LON POOLE - MARTIN McNIFF - STEVEN COOK

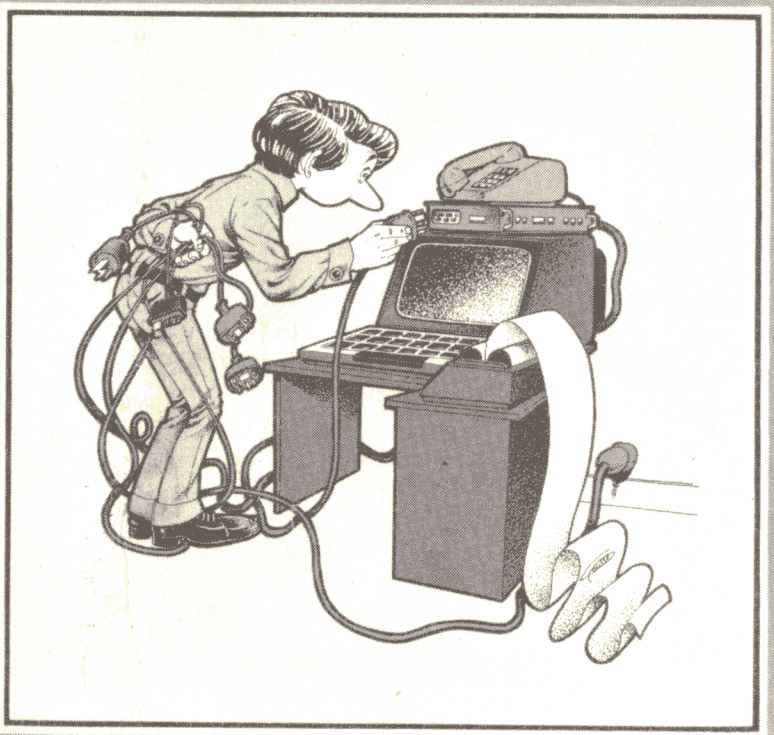
MANUEL DE L'UTILISATEUR APPLE II

S.E.C.F.
ÉDITIONS RADIO

H. LILEN

Interfaces pour MICROPROCESSEURS et MICRO-ORDINATEURS

PRATIQUE - CIRCUITS - APPLICATIONS



S. E. C. F.
ÉDITIONS RADIO

S. E. C. F.
Editions Radio

MANUEL DE L'UTILISATEUR APPLE II

LON POOLE - MARTIN McNIFF - STEVEN COOK

MANUEL DE L'UTILISATEUR APPLE II

Version française : H. Lilen

S. E. C. F.
Editions Radio
3, RUE DE L'ÉPERON 75006 PARIS
TÉL. 329.63.70-C.C.P. La Source 340.37.40 H

En première de couverture, photographie
société APPLE : système APPLE II

Imprimé en France par Berger-Levrault, Nancy
Dépôt légal : mars 1982
Éditeur n° 880. Imprimeur n° 779235
I.S.B.N. 2 7091 0880 1

INTRODUCTION

Ce livre constituera votre guide de l'ordinateur individuel Apple II. En un seul volume, il décrit cet ordinateur ainsi que ses périphériques les plus usuels, y compris les unités à disquettes et les imprimantes.

Il suppose que vous avez accès à un système Apple II complètement monté selon les instructions fournies par les manuels d'emploi livrés avec la machine. En effet, on n'expliquera pas ici comment installer le système, mais comment l'exploiter une fois qu'il a été mis en service.

Qu'est-ce qu'un Apple II ? Comment le ferez-vous fonctionner ? Les deux premiers chapitres de ce livre répondent à ces questions. Vous avez certainement remarqué que le système Apple II se compose de plusieurs éléments reliés par des câbles. Le premier chapitre vous explique ce que sont toutes ces pièces et à quoi elles servent. Le second chapitre vous indique comment faire fonctionner chacune d'entre elles. Ces connaissances acquises, vous serez prêt à faire tourner n'importe quel programme disponible sur le marché, programme de traitement de texte, d'analyse financière, de gestion, d'enseignement assisté par ordinateur ou de jeu.

Les quatre chapitres suivants vous enseignent à rédiger vos propres programmes en Basic. Le chapitre 3 commence par présenter une vue d'ensemble des deux versions de Basic disponibles avec l'Apple II, l'« Integer Basic » et le « Basic Applesoft ». Le chapitre 4 poursuit en traitant des thèmes avancés et des caractéristiques du Basic. Parmi ces thèmes, deux sont suffisamment importants pour mériter leur propre chapitre : l'unité à disques et la gestion graphique de l'écran. Le chapitre 5 expose comment employer l'unité à disquettes pour ranger des programmes et des listes de données. Le chapitre 6 vous montre comment programmer du graphique en exploitant les deux modes disponibles avec l'Apple II.

Les programmes en Basic sont exécutés sur l'Apple II sous la supervision du « Moniteur ». Le chapitre 7 explique ce qu'est ce moniteur et le moniteur « Autostart » du point de vue du programmeur. Il vous montre également comment introduire des programmes en langage assembleur dans vos programmes en Basic.
Le chapitre 8 contient une description complète de chaque instruction et de chaque fonction disponibles avec les deux versions du Basic, y comprit celles relatives aux disquettes. Avec les appendices, il vous servira de référence et vous pourrez aisément vous y reporter alors que vous serez en train de programmer votre Apple II.

CHAPITRE 1

PRÉSENTATION DE L'APPLE II

La figure 1.1. montre un système Apple II typique. Vous remarquerez qu'il se compose de plusieurs éléments interconnectés pour réaliser un système complet.
Le vôtre peut différer sensiblement, par son aspect, de celui de la figure. En effet, plusieurs éléments sont optionnels. Trois, cependant, sont à la base de tout système Apple II : l'ordinateur Apple II lui-même, son clavier incorporé, et un récepteur de télévision. Examinons chacun d'eux de plus près ; on verra aussi quels sont les éléments optionnels qui peuvent s'y adjoindre. Nous n'allons pas décire ici comment ils doivent être branchés : c'est le rôle des notices d'emploi particulières qui vous sont fournies avec chacun d'eux.

LE CLAVIER ET L'ÉCRAN TV

Le clavier et l'écran TV permettent de communiquer avec l'Apple II. Le clavier est standard, du type machine à écrire (mais avec la distribution américaine des touches, qui diffère sensiblement de la française). Il sert à transférer à l'Apple II les instructions que vous avez frappées.
L'*écran d'affichage* peut être, soit un récepteur de télévision couleur classique, soit un moniteur TV. Un récepteur de télévision en noir et blanc fonctionnera parfaitement mais naturellement, les images seront en noir et blanc. L'écran n'affiche pas seulement ce que vous frappez afin que vous puissiez vérifier vos ordres mais il affiche aussi les réactions de l'Apple II à vos instructions.
L'écran standard dispose de trois modes d'opération. L'un concerne les textes en noir et blanc uniquement et les deux autres sont principalement destinés au graphique. En mode « texte », l'écran standard est divisé en 24 lignes de 40 caractères chacune. Le mode graphique jongle avec des points et des lignes, et non plus avec des caractères, et subdivise l'écran plus finement (on en discutera au chapitre 7).

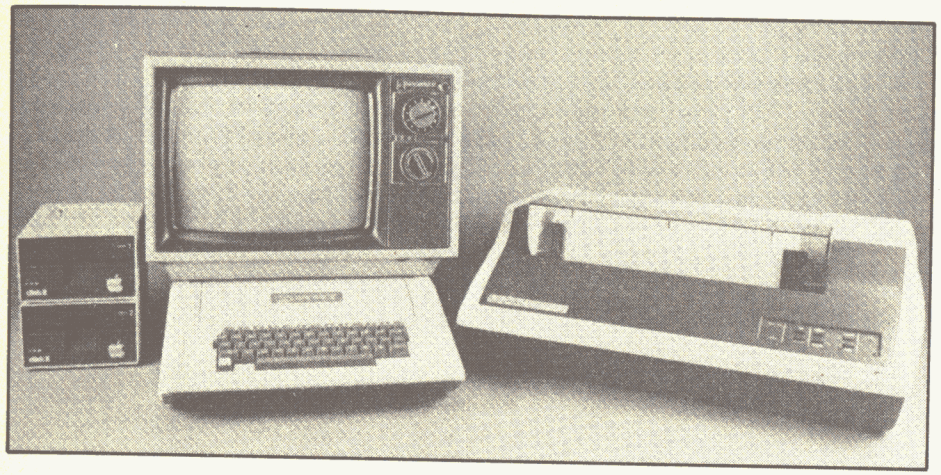


Fig. 1.1. — Un système type Apple II.

La plupart des possesseurs d'un Apple II utilisent un récepteur de télévision classique avec leur ordinateur, soit qu'ils en disposent déjà d'un, soit qu'ils trouvent là une bonne justification à son achat. Un moniteur TV produira une image plus fine qu'un récepteur TV, mais vous ne pourrez l'utiliser aussi pour voir le journal parlé de 20 h ou « Des chiffres et des lettres ». La figure 1.2 montre une connexion type entre un récepteur TV et l'ordinateur.

Ainsi qu'on peut le constater, la connexion n'est pas directe entre le téléviseur et l'Apple II. Un commutateur à glissière permet de connecter l'entrée antenne du récepteur soit sur l'antenne, pour le récepteur normal, soit sur l'Apple II. Dans ce cas, le câble qui pénètre dans l'Apple II passe par un bloc appelé « modulateur HF » chargé de convertir le signal vidéo de l'Apple II en un signal acceptable par le téléviseur. En pratique, et essentiellement pour des téléviseurs couleur aux normes françaises, on s'adressera au distributeur de l'ordinateur qui pourra efficacement conseiller l'utilisateur (et lui fournir, en particulier, la carte aux normes SECAM).

Un moniteur TV ne requiert pas de modulateur HF, la liaison est directe avec l'ordinateur, à partir des bornes prévues à cet effet à l'arrière de l'Apple II (fig. 1.3). Consultez cependant également votre distributeur ou les manuels accompagnant l'ordinateur sur le standard qu'exploite ce signal.

À L'INTÉRIEUR DE L'APPLE II

L'Apple II contient l'ordinateur commandant le reste du système — sous votre direction, bien entendu ! Juste derrière le clavier se trouve la banque des mémoires, le microprocesseur, les points de connexion pour les autres composants, et bien d'autres choses. La figure 1.4 dévoile cet intérieur, couvercle du boîtier ôté.

L'intérieur de votre propre Apple II pourra paraître différent. Vous y trouverez cependant la même grande carte imprimée supportant des dizaines de petits pavés noirs, appelés des *circuits intégrés,* en rangées régulières, et quelques cartes enfichées verticalement dans des connecteurs femelles situés à l'arrière de la grande carte principale. Le nombre de circuits intégrés ainsi que le nombre et l'emplacement des cartes additionnelles verticales varient d'un système à l'autre.

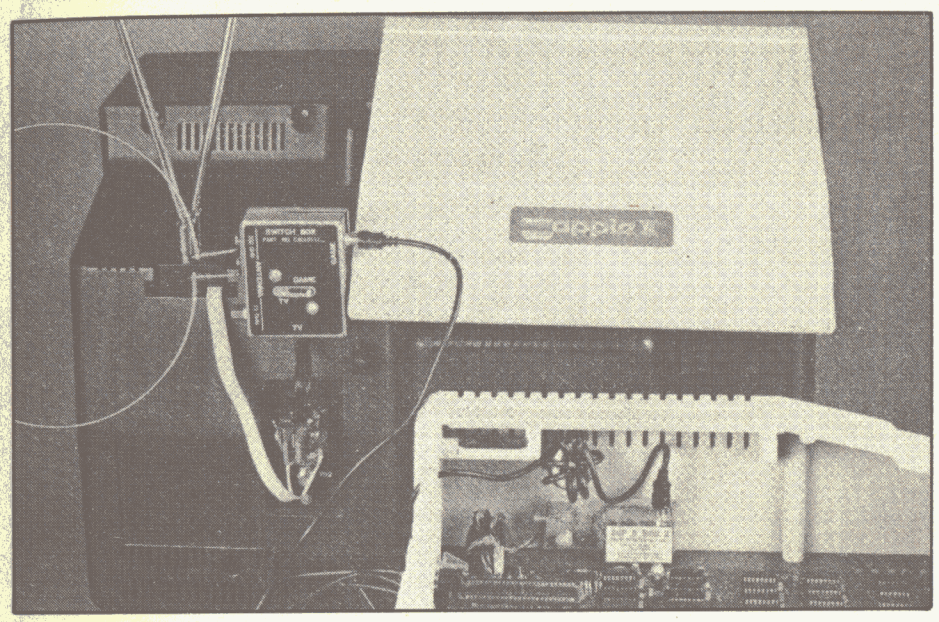


Fig. 1.2. — Connexion à un téléviseur.

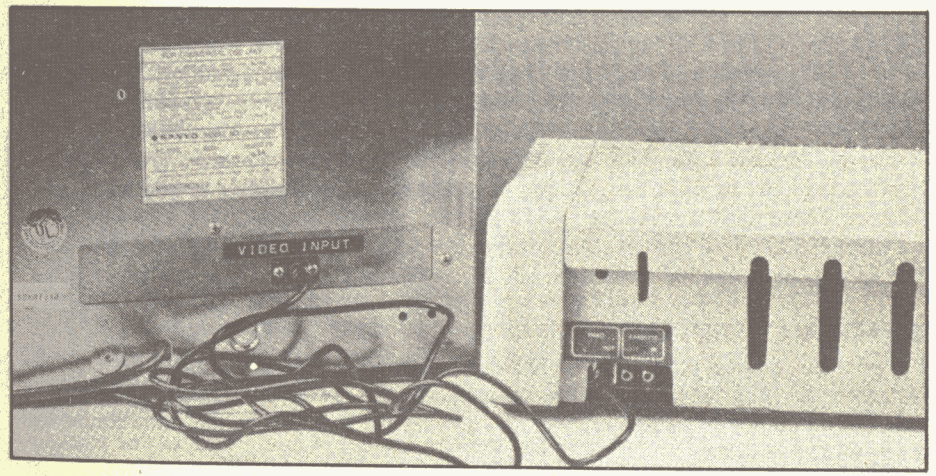


Fig. 1.3. — Connexion à un moniteur vidéo.

MÉMOIRE

L'importance d'une mémoire d'ordinateur est généralement évaluée en *octets.* Chaque octet peut stocker un caractère, ou une quantité de données équivalente. Selon le nombre de ses circuits intégrés, notre Apple II disposera de 4 096 à 65 536 octets de mémoire, ce que l'on désigne par 4 K à 64 K octets, le coefficient K valant 1 024. Le volume mémoire détermine ce que votre Apple II peut faire, ainsi qu'on le verra.

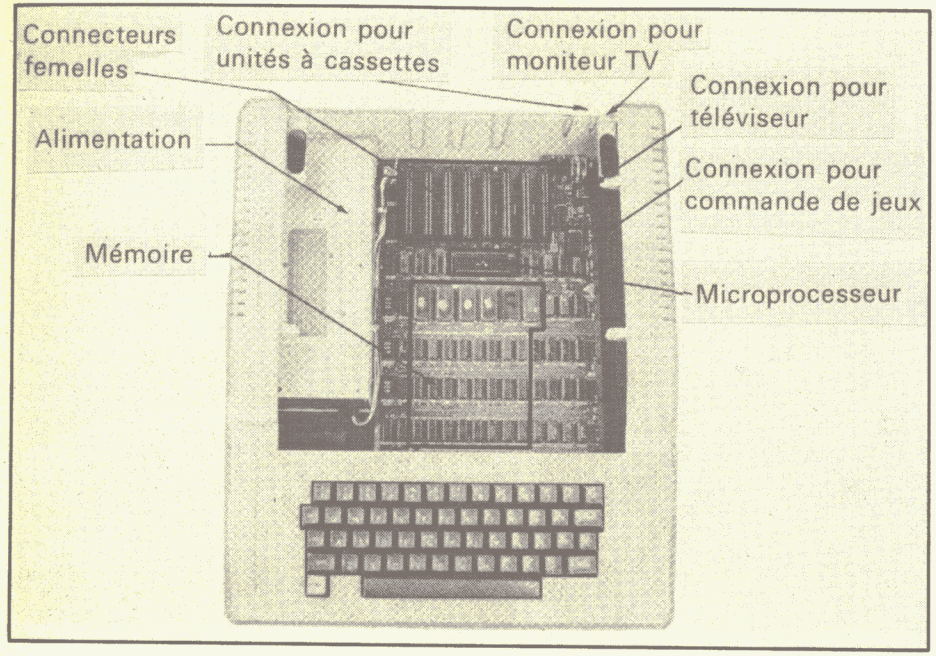


Fig. 1.4. — A l'intérieur d'un Apple II.

L'Apple II dispose réellement de deux types de mémoires. L'un est appelé *mémoire morte* (ou, en abrégé, ROM, de l'anglais « Read Only Memory », soit *mémoire à lecture seulement) ;* son contenu ne change jamais, même si vous coupez l'alimentation. La mémoire ROM contient le programme qui personnalise l'Apple II, lui permet de comprendre vos ordres et d'y répondre. L'autre type de mémoire est à *lecture et écriture ;* on l'appelle RAM, de l'anglais « Random Access Memory », soit *mémoire à accès aléatoire ;* son contenu peut être modifié. En pratique, le programme qui détermine ce que Apple II va faire est stocké dans la mémoire RAM.

La mémoire RAM ne conserve ses informations que pour autant qu'elle est alimentée. Dès lors qu'on coupe l'alimentation, elle perd son contenu.

UNITÉ À CASSETTES

Par bonheur, on peut utiliser une unité à cassettes (enregistreur-lecteur de cassettes) pour transférer des programmes de, et vers les mémoires RAM de l'ordinateur ; on constitue ainsi des bibliothèques de programmes sur cassettes. La figure 1.5 montre une connexion à une unité à cassettes.

UNITÉ À DISQUETTES

L'unité à disquettes surpasse en qualités l'unité à cassettes pour stocker des programmes. La disquette est plus fiable, mémorise davantage de données et est plus rapide. De plus, la disquette stocke facilement et rapidement des données telles que des noms ou des adresses pour des listes d'expéditions de courrier, ou de la correspondance pour un système de traitement de texte. Les unités à disquettes se présentent sous toutes formes et dimensions ; des modèles différents possèdent des capacités de stockage différents. La figure 1.6 montre deux unités « Disk II » produites par *Apple Computer Inc.*

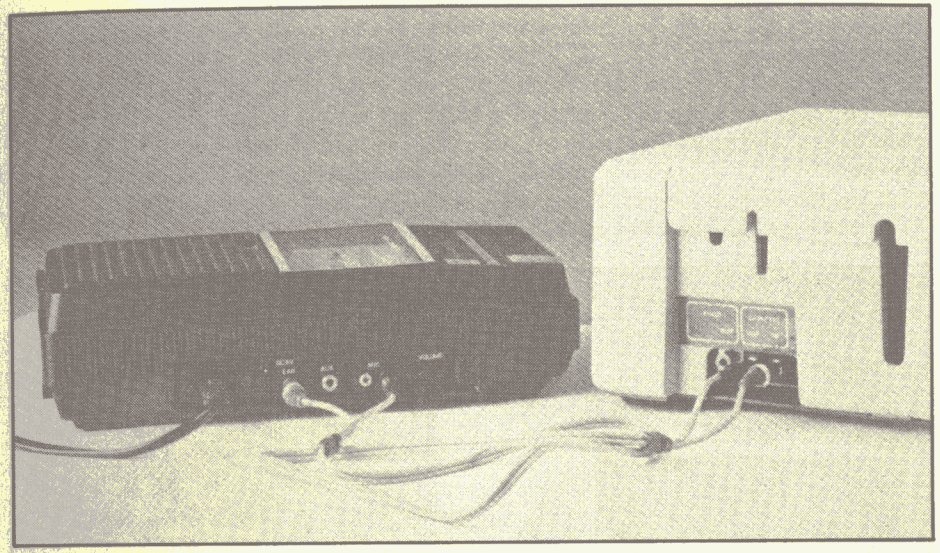


Fig. 1.5. — Connexion à une unité à cassettes.

PROGRAMMES

Faisons un petit crochet dans notre visite guidée du système Apple II afin d'examiner les diverses sortes de programmes que vous utiliserez. Nous ne voulons pas discuter des diverses applications que vous ferez avec votre ordinateur : traitement de texte, comptabilité, analyse financière, etc., mais des programmes variés qui doivent co-exister pour que l'Apple II puisse servir à ces fins. Les programmes chargés de commander des jeux, écrire des lettres, etc., sont appelées des *programmes d'application.* Ils résident toujours en mémoire RAM, jamais en ROM. Vous les transférez en RAM à partir d'une cassette ou d'une disquette. Ainsi, si vous voulez que votre Apple II devienne un outil de traitement de texte, par exemple, vous utiliserez une disquette sur laquelle le programme de traitement de texte a été enregistré et vous le transférerez en RAM, on verra comment au chapitre 2.

Le plus souvent, les programmeurs rédigent leur programme d'application dans un langage de programmation qu'ils peuvent lire aisément mais que Apple II ne pourra comprendre sans aide. Un programme spécial, appelé *interpréteur,* fera exactement ce que son nom suppose. Il traduira le programme d'application rédigé dans son propre langage en un langage que l'ordinateur comprendra. L'Apple II dispose de divers interpréteurs, chargés soit en mémoire RAM, soit en mémoire ROM.

L'interpréteur, à son tour, dépend d'un autre programme chargé de coordonner le système. Cet autre programme, souvent appelé *système d'exploitation,* exécute les opérations de base exigées par le système : transfert de programmes d'une cassette ou d'une disquette à la mémoire, renvoi en écho des touches frappées pour affichage sur l'écran, etc. Le système d'exploitation de l'Apple II est appelé le *moniteur ;* il réside toujours en mémoire ROM.

CONTRÔLEURS DE PÉRIPHÉRIQUES

Remarquez les cartes verticales de circuits imprimés enfichées dans les connecteurs femelles situés à l'arrière de la carte maître de l'Apple II. Ces cartes, appelées aussi *contrôleurs,*

contiennent l'électronique supplémentaire requise pour gérer des périphériques, tels que l'unité à disquettes.
La société *Apple Computer Inc.* propose un certain nombre de telles cartes. Certaines se voient affecter un connecteur femelle bien déterminé, mais la plupart pourront être enfichées dans n'importe lequel d'entre eux. Le marquage de ces cartes n'est pas toujours évident aussi le profane aura-t-il du mal à les différencier. Elles portent bien un nom, mais il n'est pas toujours visible lorsque la carte est enfichée. Vous pourrez identifier une carte en repérant quel est son connecteur, ou à quoi elle est connectée. La figure 1.6 montre

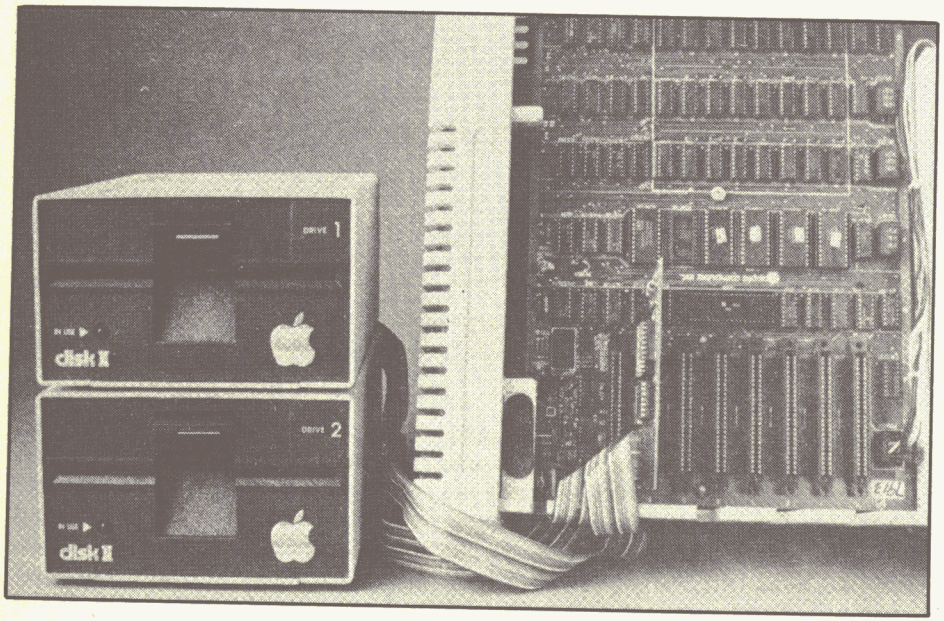


Fig. 1.6. — Connexion à une unité à disquettes.

une carte contrôleuse d'unités à disquettes « Disk II » que l'on enfichera par tradition dans le connecteur numéro 6 ; on la reliera à une ou deux unités à disquettes. Une seconde carte semblable pourra être enfichée dans le connecteur 5 pour une troisième et une quatrième unités à disquettes, et même une troisième dans le connecteur femelle 4 pour des cinquième et sixième unités à disquettes, etc.
Jetons un coup d'œil, maintenant, sur les autres cartes qui peuvent intervenir dans votre Apple II.
Chacune des trois cartes photographiées dans la figure 1.7 améliore les capacités du langage de programmation de votre ordinateur. Deux versions du langage de programmation Basic sont disponibles sur l'Apple II : le « Integer Basic » (ou Basic « entier »), et l'« Applesoft ». L'implantation de l'une de ces trois cartes, embrochée dans le connecteur femelle 0 de votre Apple II facilite le passage d'une version du Basic à l'autre. Les cartes « Language System » *(langage système)* et « Applesoft Formware » *(firmware Applesoft,* firmware se traduisant par logiciel implanté dans un circuit intégré) peuvent fonctionner sur n'importe quel type de Apple II, mais la carte « Integer Basic » n'a été conçue que pour l'*Apple II Plus.* La carte *Language System* permet l'utilisation d'autres langages de programmation, tel que le Pascal.

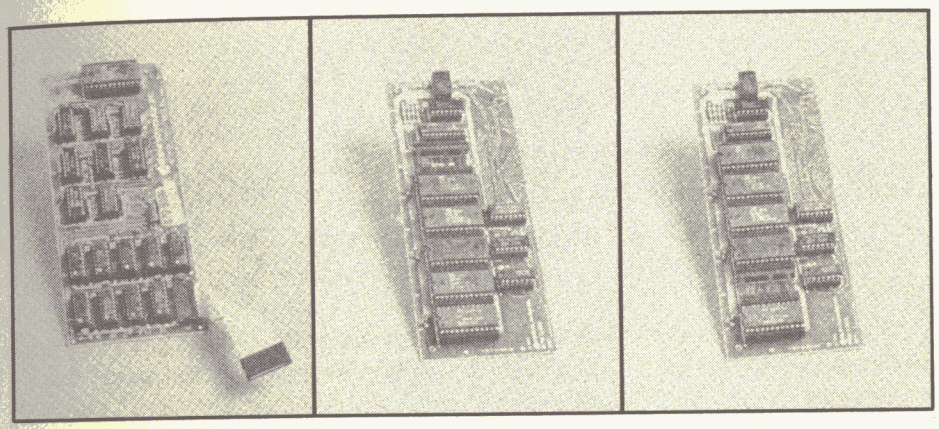


Fig. 1.7. — De gauche à droite : cartes « Language System », « Applesoft Firmware », et « Integer Basic ».

La carte d'*interface série* (fig. 1.8) est généralement montée sur le connecteur femelle numéro 1. Le périphérique le plus commun qui lui soit connecté est probablement une imprimante.
Il est assez rare, par contre, qu'on monte une carte d'*interface parallèle pour imprimante* (fig. 1.9) avec la carte d'interface série, car elle est invariablement connectée à une imprimante. Aussi la carte d'*interface parallèle pour imprimante* peut-elle être embrochée dans le connecteur 1.
La figure 1.10 montre la carte d'*interface de communications.* Avec un modem la reliant à une ligne téléphonique, elle permettra à votre Apple II de communiquer avec d'autres ordinateurs distants.

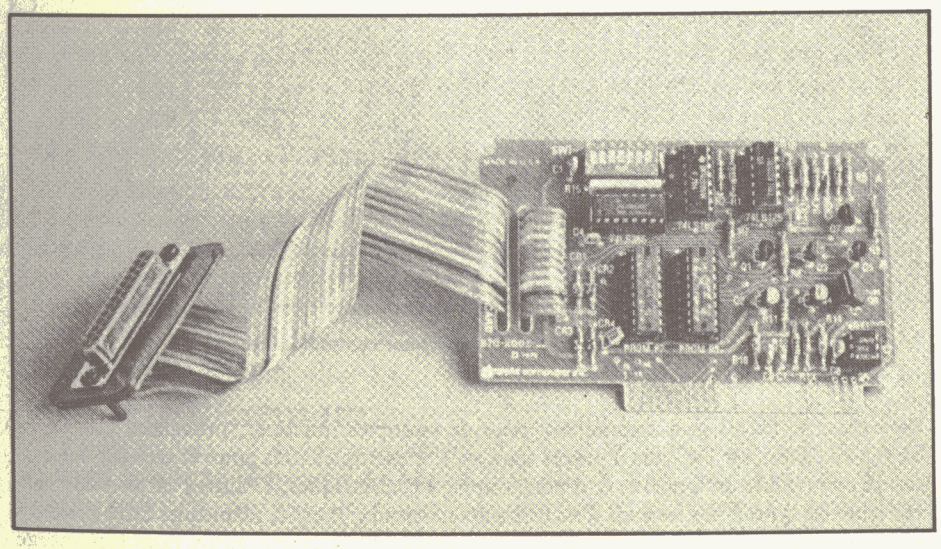


Fig. 1.8. — Carte d'interface série.

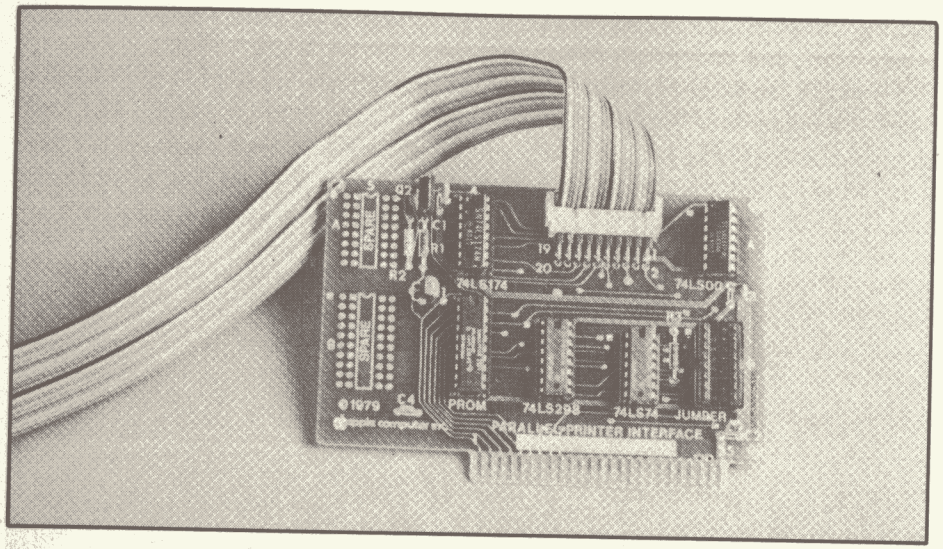


Fig. 1.9. — Carte d'interface parallèle pour imprimante.

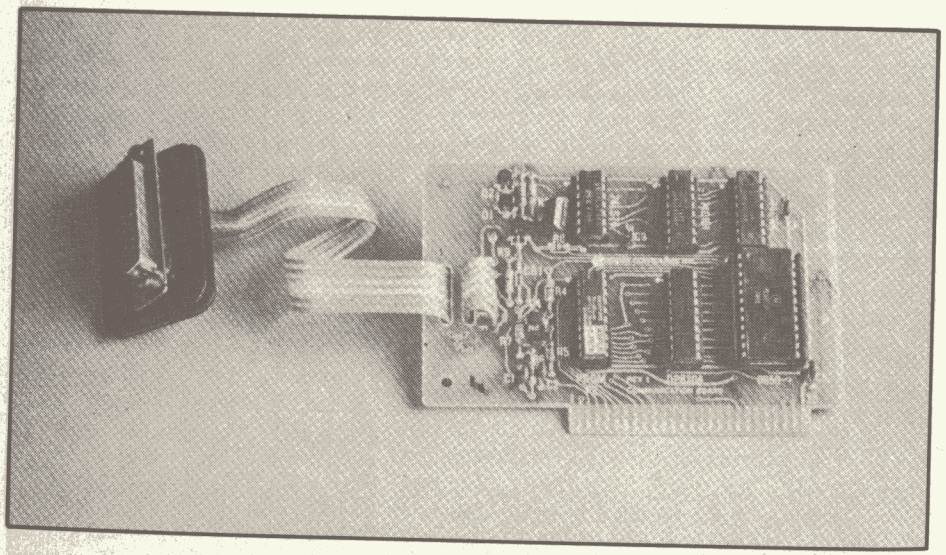


Fig. 1.10. — Carte d'interface pour communications.

Bien d'autres cartes sont disponibles, pour de multiples fonctions, provenant aussi bien de *Apple Computer Inc.* que d'autres sources. Par exemple, vous pourrez trouver des cartes de commande de lumières et d'autres appareils électriques. D'autres cartes vous permettront de synthétiser les sons d'instruments musicaux. Il existe même une carte d'extension qui, enfichée dans l'un des connecteurs femelles, en fournit d'autres pour le montage de cartes additionnelles dans un châssis séparé.

Certains systèmes Apple II disposent d'une carte spéciale qui, lorsqu'elle est installée, peut doubler le nombre de caractères de chaque ligne affichée. Dans ce cas, le moniteur TV (un récepteur TV ne suivrait plus) est connecté à cette carte et non plus à la carte principale de l'Apple II. Cette carte spéciale s'embroche dans l'un des connecteurs à l'arrière de la carte principale, généralement le connecteur femelle 3 ou 4. La figure 1.11 montre l'aspect typique d'une telle carte.

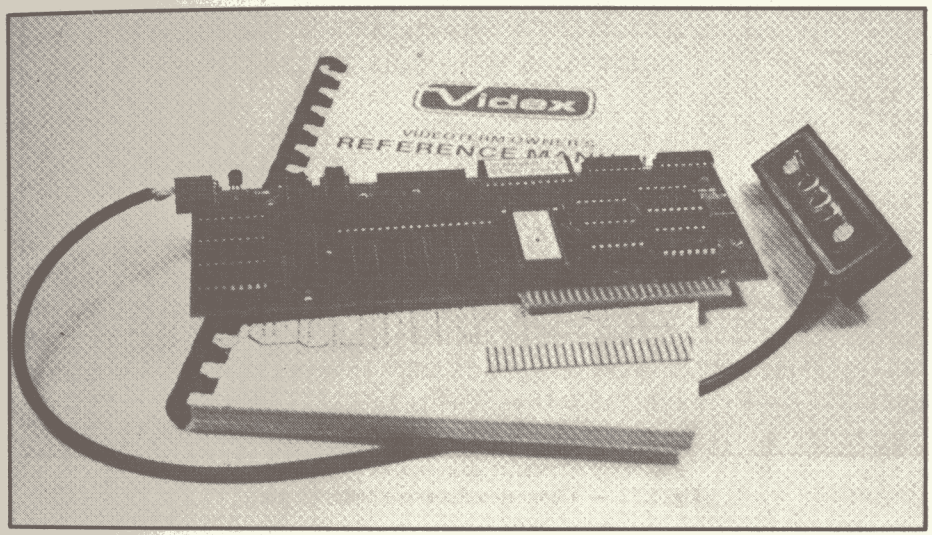


Photo courtesy of Videx

Fig. 1.11. — Carte spéciale d'affichage. (Doc. *Videx*).

COMMANDES DE JEUX

Nous allons terminer ce tour d'horizon en examinant les commandes de jeux qui se connectent à l'Apple II comme le montre la figure 1.12. Ces commandes sont employées par la plupart des programmes de jeux mais peuvent aussi intervenir pour d'autres usages. Parfois, les commandes de jeux ne seront pas connectées à l'Apple II.

IMPRIMANTE

Tournons-nous, maintenant, vers l'imprimante du système (fig. 1.13).
L'imprimante est connectée au système soit via la carte d'interface série, soit via la carte d'interface parallèle (tout dépend du type de l'imprimante), généralement enfichée dans le connecteur 1. On trouve des imprimantes de toutes tailles, prix, et types. Certaines traiteront votre correspondance aussi bien qu'une bonne machine à écrire ; d'autres imprimeront des graphiques (et même parfois en couleur). D'autres, enfin, représenteront un compromis entre celles-ci.

TABLETTE GRAPHIQUE

La tablette graphique d'*Apple Computer Inc.* (fig. 1.14) est un dispositif qui exploite à sa façon les possibilités graphiques de l'Apple II. Avec son stylet, vous pourrez créer des dessins, cartes, diagrammes... qui apparaîtront directement sur l'écran, en couleur. Sous votre direction, l'ordinateur tracera des lignes droites, des figures géométriques ou des points. Votre dessin pourra occuper tout ou partie de l'écran. Vous pourrez le déplacer,

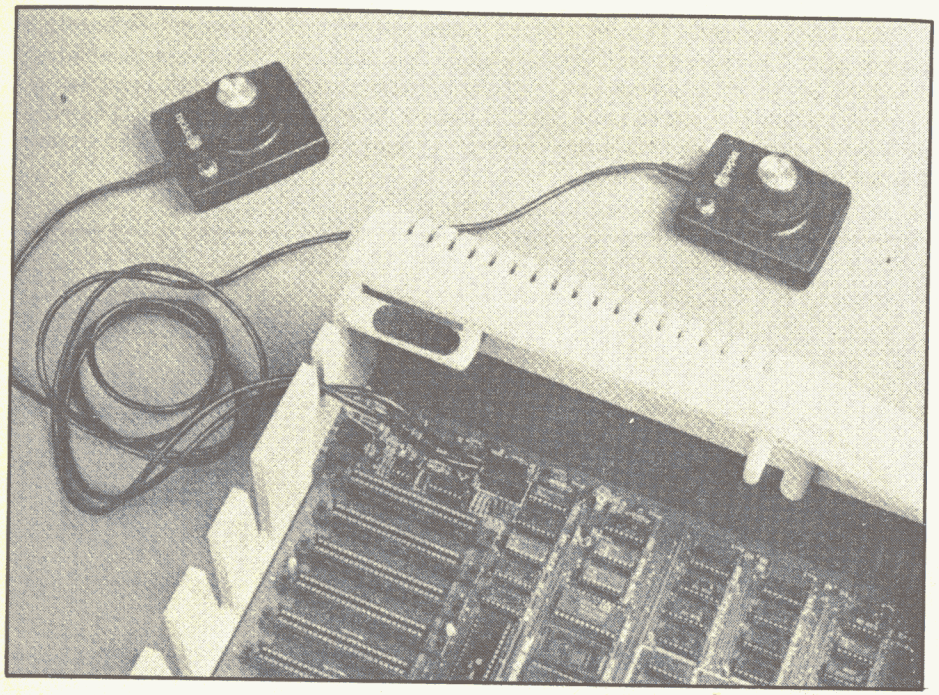


Fig. 1.12. — Connexion des commandes de jeux.

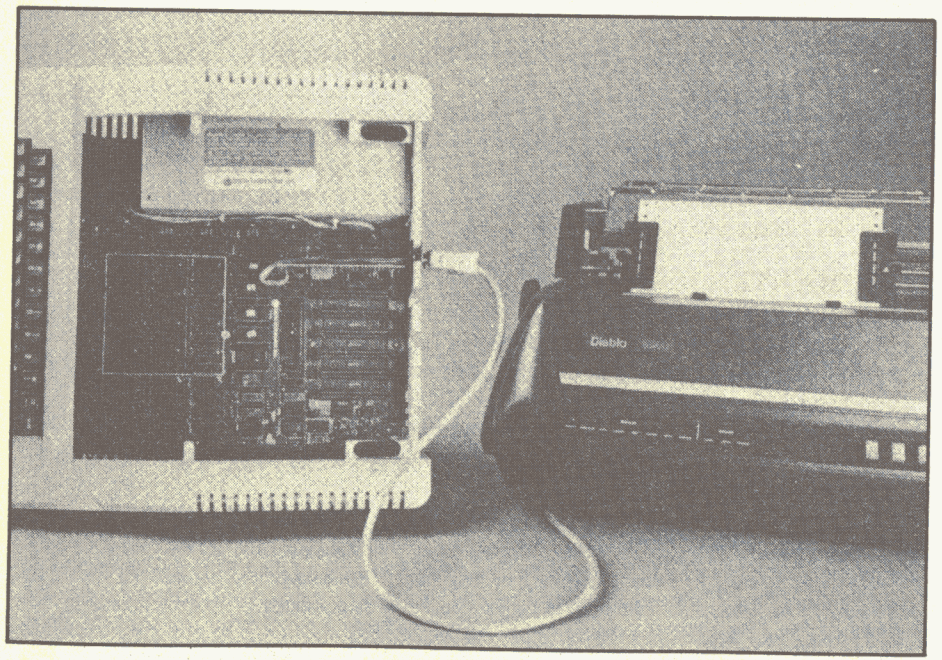


Fig. 1.13. — Comment connecter l'imprimante.

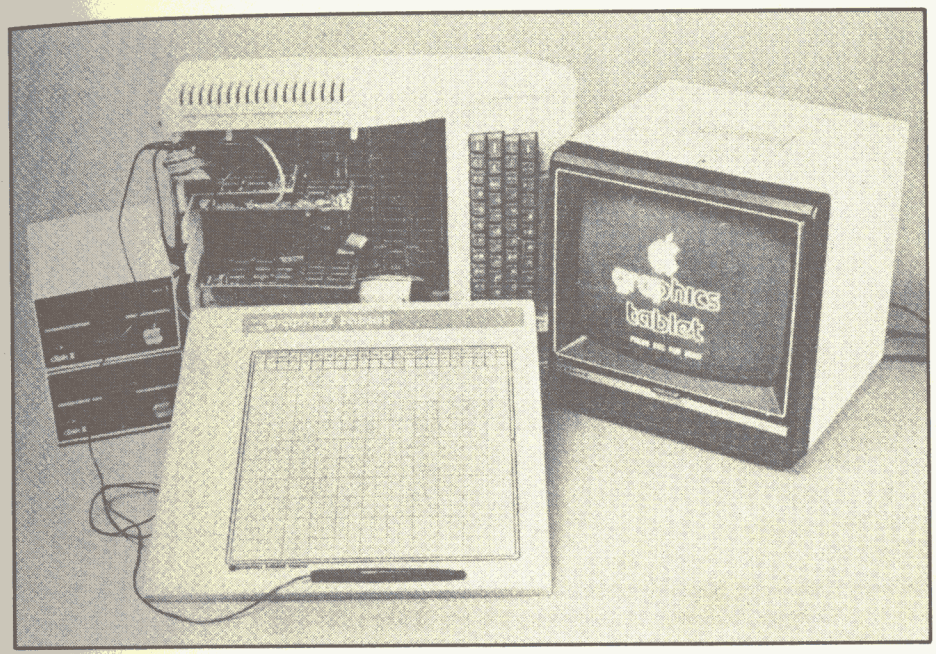


Fig. 1.14. — Connexion de la tablette graphique.

l'agrandir ou le réduire, le séparer couleur par couleur. A tout instant, vous pourrez sauvegarder l'image sur l'écran dans votre disquette pour la rappeler ultérieurement. Vous pourrez même mesurer des distances avec cette tablette graphique.

CHAPITRE 2

COMMENT FAIRE FONCTIONNER L'APPLE II

Vous pourrez être intimidé par n'importe quel ordinateur lorsque vous vous assiérez devant lui pour la première fois, même s'il est parfaitement connecté, comme ce devra être le cas avec votre Apple II avant d'aller plus avant. Ce chapitre va vous familiariser avec l'Apple II en vous expliquant comment l'utiliser, mais il ne vous indiquera pas comment le mettre en état de service. Les notices techniques qui l'accompagnent sont faites pour cela et vous procureront les instructions complètes d'installation. Si vous éprouvez le besoin d'une assistance supplémentaire pour vous assurer que vous avez tout connecté et installé correctement, faites appel à une tierce personne plus expérimentée disposant du même Apple II que vous, ou à votre distributeur Apple.

MISE SOUS TENSION

Vérifiez que tous les constituants du système sont correctement interconnectés et mettez en marche le récepteur TV. Réduisez le volume. Repérez le commutateur situé sur la sortie antenne de votre récepteur et placez-le sur la position *Jeux* (« Game ») ou *Ordinateur* (« Computer »). Sélectionnez le canal spécifié par les instructions accompagnant le modulateur HF (généralement le canal 33) ; si vous ne découvrez pas quel canal enclencher, prenez conseil auprès d'un utilisateur déjà expérimenté ou auprès du revendeur.
Repérez l'interrupteur secteur placé à l'arrière de l'Apple II, à côté du branchement du cordon secteur, et placez-le sur la position ON *(marche)*. Vous devriez entendre un « bip » provenant de l'intérieur de l'Apple II, vous signifiant que votre ordinateur est prêt. Le voyant POWER de votre clavier doit être allumé (à moins que la petite lampe ne soit grillée).
Si vous n'avez pas entendu le « bip », coupez (OFF), puis remettez sous tension (ON) votre ordinateur. Si vous n'entendez toujours rien, arrêtez-le (OFF). Le voyant POWER était-il allumé ? Si non, débranchez l'Apple II et connectez à son cordon d'alimentation, un appareil prouvant que le secteur arrive bien, que ce cordon n'est pas coupé, que la prise secteur murale est sous tension. Relisez les instructions de montage et revérifiez tout afin de vous assurer que tout est correctement branché. Si, après un nouvel essai, rien ne

marche, *coupez l'alimentation !* Débranchez votre Apple et appelez à l'aide quelqu'un de plus expérimenté, ou votre distributeur. Vous pourriez commettre bien des dégâts en voulant examiner l'intérieur de votre ordinateur ou en intervertissant les câbles.

CE QUE VOUS VOYEZ SUR L'ÉCRAN TV

Une fois sous tension, et le « bip » ayant été émis, une image doit apparaître sur l'écran TV.

L'image exacte dépend du type d'Apple II dont vous disposez, mais vous remarquerez dans tous les cas qu'un symbole va se mettre à clignoter sur l'écran. C'est un petit carré blanc, appelé un *curseur*. Il marque l'endroit où sera affiché le prochain caractère. Le texte suivant vous explique ce qu'il faut faire si vous ne voyez pas ce curseur.

Si le curseur n'apparaît pas

Votre Apple II peut déjà être équipé d'une carte « Language System », et doté d'une ou deux unités à disquettes. Dans ce cas, vous ne verrez pas de curseur. A la place, vous entendrez quelques claquements et ronflements en provenance de l'une des unités à disquettes « Disk II », et le voyant rouge qui figure sur sa face avant et qui est marqué IN USE *(en service)* sera allumé. Frappez alors la touche « Reset » *(remise à zéro) :* le curseur apparaîtra et l'unité à disquettes s'arrêtera. Dès lors, vous pouvez passer au paragraphe suivant.

D'autre part, certains systèmes Apple II sont montés pour permettre l'affichage de plus de 40 caractères sur chaque ligne de l'écran. Ici, et pour voir le curseur, vous devrez frapper une séquence de commandes :

1. Pressez la touche CTRL et maintenez-la enfoncée ; pressez alors la touche B, puis relâchez-les toutes deux.

2. Pour la commande suivante, vous devez savoir dans quel connecteur femelle votre carte spéciale (à laquelle le moniteur TV est connecté) a été enfichée ; ce sera probablement le connecteur 3 ou 4. Vous pouvez ouvrir votre Apple pour le vérifier. Les connecteurs sont numérotés de 0 à 7 de la gauche vers la droite. La figure 1.11 montre à quoi ressemble cette carte. En cas de doute, faites appel à quelqu'un de plus expérimenté, ou à votre revendeur.

3. Si votre moniteur TV est relié à une carte enfichée dans le connecteur 3, frappez PR# et pressez la touche RETURN (équivalent à un *retour chariot*). S'il s'agit du connecteur 4, frappez PR#4 ; pour le connecteur 2, faites PR#2 ; etc.

4. Le curseur doit maintenant être visible, même s'il ne clignote pas (ce qui est parfait). Poursuivez en lisant ce qui suit.

Caractères d'appel	Langage
*	Moniteur
>	Integer Basic
]	Applesoft

Tableau 2.1. — Caractères d'appel.

LE CARACTÈRE D'APPEL

A la gauche du curseur figure un autre caractère ; c'est le premier caractère de la ligne. Ce peut-être un astérisque (*), un symbole « plus grand que » (>) ou la fermeture d'un crochet (]). L'Apple II est un ordinateur multilingue et ce caractère d'appel précise dans quel langage il attend vos instructions. Le tableau 2.1 montre ces trois caractères d'appel et leur langage correspondant.

* est le moniteur

Avec de nombreuses versions de l'Apple II, le premier caractère d'appel que vous verrez en mettant votre machine sous tension est cet astérisque. Si tel n'est pas le cas, sautez ce paragraphe et passez au suivant.

L'astérisque est le caractère d'appel du *langage assembleur moniteur* de l'Apple II. Généralement, seuls les utilisateurs avertis de l'Apple II éprouveront le besoin de communiquer directement avec le moniteur. Si vous souhaitez expérimenter le moniteur, n'hésitez pas, vous ne risquez pas à cette étape de provoquer le moindre dégât. Après quoi, vous arrêterez votre Apple II puis le remettrez sous tension.

Lorsque l'astérisque d'appel réapparaît à nouveau, vous devrez ordonner à l'Apple II de passer au Basic.

CTRL-B pour passer au Basic

L'une des commandes du moniteur ordonne à l'Apple II de passer au Basic. Pour l'émettre, pressez et maintenez enfoncée la touche CTRL tandis que vous frappez puis relâchez la touche B. Relâchez ensuite CTRL et pressez RETURN. Un second caractère d'appel surgira juste en-dessous de l'astérisque. Selon le type d'Apple dont vous disposez, vous verrez le symbole > ou la fermeture d'un crochet].

Le mot Basic a été formé avec les premières lettres de *Beginner's All-purpose Symbolic Instruction Code,* soit « code d'instructions symboliques tous usages pour débutants » ; ce langage a été développé à l'université de Dartmouth (USA). L'Apple II dispose de deux versions du Basic : le Basic entier (« Integer Basic »), encore appelé Basic simple, et le Applesoft, appelé aussi Basic étendu. Ces deux versions comportent des améliorations par rapport au Basic original de Dartmouth.

> est l'« Integer Basic »

Le caractère d'appel > indique que l'Apple II est prêt à recevoir des instructions en Basic entier (« Integer Basic »). Ses règles sont différentes de celles de l'Applesoft, mais ils ont tous deux des choses en commun. Nous utiliserons par la suite des instructions que vous utiliserez avec ces deux Basic, de telle façon que vous n'ayez pas à vous soucier de celui dont vous disposez réellement.

] désigne l'Applesoft

Le caractère d'appel] vous indique que votre Apple II est prêt à recevoir des ordres en Basic Applesoft. Ne vous souciez pas, actuellement, de la version du Basic que vous employez. Lorsque ce sera nécessaire, nous évoquerons les différences entre ces deux versions.

LE CLAVIER

Le clavier de l'Apple II ressemble beaucoup au clavier d'une machine à écrire (fig. 2.1), américain cependant (clavier « QWERTY »), mais il dispose de cinq touches qu'on ne trouve guère sur les machines à écrire. Deux sont situées à gauche et mystérieusement marquées ESC et CTRL. Les trois autres se trouvent à droite et sont marquées RESET, ← et →. Deux autres touches, à droite, pourront aussi ne pas vous paraître évidentes : RETURN et REPT.
Essayez votre clavier en frappant sur les touches. Vous ne pourrez rien provoquer qui puisse abîmer votre ordinateur ; si nécessaire et pour que tout rentre dans l'ordre, coupez l'alimentation et rétablissez-la aussitôt.
Au fur et à mesure que vous frappez des lettres, vous remarquerez qu'elles sont toutes en majuscules, quelle que soit la position de la touche SHIFT (qui correspond à la touche « majuscules » de nos machines à écrire). L'Apple II standard ne connaît, en effet, que les majuscules. Vous pourriez connecter votre récepteur TV à un contrôleur spécial autorisant à la fois les majuscules et les minuscules. Par ailleurs, certains programmes savent commander l'affichage des majuscules et des minuscules.

La touche RESET

La touche RESET se voit confier une fonction très particulière. Lorsqu'elle est pressée, toute exécution s'arrête, quoi que l'ordinateur ait été en train de faire à ce moment.

Fig. 2.1. — Le clavier de l'Apple II.

Le contrôle de l'Apple II revient alors au clavier. Selon le type de votre machine, RESET provoquera la réapparition de l'un des trois caractères d'appel du moniteur, de l'Integer Basic ou de l'Applesoft.
Cette touche RESET pourra cependant vous créer des ennuis, en particulier si vous l'actionnez pendant qu'une disquette — une unité Disk II — est en service. Aussi devez-vous faire particulièrement attention *à ne pas presser* RESET accidentellement. Méfiez-vous surtout lorsque vous voudrez frapper la touche RETURN ; ne vous trompez pas de touche, RESET étant située au-dessus. Dans certaines versions de l'Apple II, on vous proposera d'utiliser la commande CTRL-RESET et non plus uniquement RESET, afin de vous prémunir contre ce risque (voyez la discussion ci-dessous).

La touche RETURN

Au fur et à mesure de votre frappe, les lettres s'alignent sur l'écran. En outre, elles sont rangées en mémoire, mais à ce stade, l'Apple II ne cherche pas à interpréter ce que vous frappez ; il ne le fera que lorsque vous agirez sur la touche RETURN. Vous lui signalez ainsi que vous avez terminé la ligne. Lorsque vous pressez RETURN, Apple II efface tout ce qui pourrait se trouver à droite du curseur, puis il examine votre ligne. Si ses caractères forment une instruction qu'il comprend, il exécutera l'action appropriée. Sinon, vous entendrez un « bip » et verrez apparaître l'un des messages d'erreurs possibles, parmi lesquels :

```
?SYNTAX ERROR
*** SYNTAX ERR
```

Ce faisant, Apple II vous signale qu'il ne comprend pas ce que vous avez voulu lui dire en frappant les caractères avant de presser RETURN. Vous aurez à refrapper toute la ligne (mais sans l'erreur que vous a annoncée votre ordinateur).

La touche SHIFT

Les lettres sont toujours des majuscules sur l'Apple II standard, aussi la touche SHIFT ne sert-elle pas à passer des minuscules aux majuscules. Elle permet à certaines touches de produire deux caractères différents, l'un lorsque l'une de ces touches est pressée alors que SHIFT est maintenue enfoncée, l'autre lorsqu'il n'y a aucune action sur SHIFT. Le caractère obtenu alors que SHIFT est enfoncée est indiqué au sommet des touches.
Nous utiliserons la notation SHIFT- pour indiquer cette double action impliquant une frappe de SHIFT. Par exemple, SHIFT-3 produira le caractère #.
Quelques touches portant des lettres n'affichent plus de lettres lorsqu'on les frappe alors que SHIFT est enfoncée. Par exemple, N affichera un ∧ si SHIFT est enfoncée, et SHIFT-P produira un *a commercial ou* @. Bien que la touche G dispose du mot BELL (sonnerie), la frappe de SHIFT-F n'affichera pas une sonnette, mais seulement un G.

La touche CTRL

CTRL est une contraction de contrôle (soit, commande). La touche CTRL sert toujours avec une autre, de la même façon que SHIFT. Vous maintiendrez CTRL enfoncée tandis que vous presserez une autre touche. Cette double action sera indiquée par CTRL-. Par exemple, CTRL-B définira une double action sur CTRL et sur B simultanément.
Cette touche, tout comme SHIFT, permet à d'autres clés de disposer d'une fonction supplémentaire. La touche G est la seule sur laquelle on ait inscrit sa fonction lorsqu'elle est pressée tandis que CTRL est enfoncée : BELL (sonnerie). En effet, si vous faites CTRL-G, vous entendrez le « bip » de la sonnerie de l'Apple II. Vous avez déclenché la sonnette.
Les autres combinaisons de CTRL provoqueront diverses réactions de votre ordinateur. Nous en avons déjà évoqué une, CTRL-B, qui place l'Apple II en Basic.
Une autre combinaison pratique est CTRL-X : elle indique à Apple II d'ignorer tout ce que vous venez de frapper sur la ligne courante, car vous désirez l'annuler et recommencer. Essayez de frapper quelque chose, puis faites CTRL-X : le curseur revient au début de la ligne suivante. Les anciens caractères subsistent sur l'écran mais Apple II n'en conserve plus trace en mémoire ; pour lui, tout se passe comme si vous n'aviez rien frappé.

La touche ESC

L'abréviation ESC signifie « escape » : c'est un caractère d'*échappement,* dont l'origine remonte aux âges héroïques où le télé-imprimeur était le terminal informatique le plus

courant. La touche ESC servira à maints usages, qu'on examinera dans ce chapitre et dans le suivant.
A l'inverse de SHIFT et CTRL, ESC n'est jamais employée en la maintenant enfoncée alors qu'on frappe une autre clé. ESC sera toujours pressée, puis relâchée, avant une action sur une autre touche. Une telle opération double constitue une *séquence d'échappement*.
Une telle séquence sera constituée, par exemple, par ESC-@. Vous frappez ESC puis relâchez cette touche, puis vous frappez SHIFT et maintenez cette clé enfoncée tandis que vous actionnez puis relâchez la touche P (SHIFT-P = @). Le résultat ? Tout ce qui se trouvait sur l'écran est effacé et le curseur est revenu dans le coin supérieur gauche. (Dans le jargon informatique, l'angle supérieur gauche est « la maison », le « *home* » : ESC-@ commande un retour à la maison.)

Les touches ← et →

Ces deux touches porteuses de flèches s'appellent *flèche gauche* et *flèche droite*.
Vous trouverez ces touches très utiles car elles vous permettent la correction des fautes de frappe et la modification des informations déjà introduites. La touche ← agit comme la touche de retour (espace arrière) d'une machine à écrire. Chaque fois que vous l'actionnerez, vous effacerez de la mémoire de l'Apple II le caractère présent sous le curseur ; ce dernier reviendra en arrière de un caractère. Vérifiez-le maintenant. Frappez n'importe quel mot, par exemple BASIC ; actionnez ensuite ← plusieurs fois et observez le déplacement du curseur. Les caractères ne disparaîtront pas de l'écran, mais soyez assuré qu'ils auront été effacés de la mémoire de l'Apple II. Ramenez ainsi le curseur au bord gauche de l'écran, et actionnez encore une fois ←. Le curseur passera à la ligne suivante sur laquelle un nouveau caractère d'appel est apparu.

Ainsi que vous pourriez vous en douter, la touche → déplace le curseur à droite sur la ligne affichée. Chaque fois qu'il passe sur un caractère affiché, celui-ci est mis en mémoire exactement comme si vous aviez actionné la touche correspondante. Pour le vérifier, frappez un autre mot, ramenez le curseur en arrière avec ←, puis pressez → plusieurs fois, chaque fois que le curseur passera sur un caractère, celui-ci sera rechargé en mémoire comme si vous l'aviez refrappé.

La touche REPT

Le mot REPT est l'abrégé de « répétition ». Si vous maintenez la touche REPT enfoncée avec une autre touche simultanément, le caractère de cette dernière sera répété jusqu'à ce que vous relâchiez ces deux clés. C'est très pratique, surtout lorsque la seconde touche est une flèche et que vous avez beaucoup de caractères à effacer (←) ou à recopier (→).
Pour un fonctionnement correct, la séquence consiste d'abord à enfoncer la touche qu'on veut répéter, puis à presser REPT. On relâchera ensuite REPT, puis la touche du caractère utile.

Les autres touches

Les autres touches de votre clavier vous sont sans aucun doute familières. Ce sont les lettres de l'alphabet, les nombres de 0 à 9, et un jeu standard de symboles.
Souvent, on ne fait guère la distinction entre le nombre zéro et la lettre O, ou entre le nombre 1 et la lettre l en minuscule. L'Apple II ne peut accepter une telle ambiguïté. Vous devrez frapper un nombre si vous voulez un nombre. Pour différencier le zéro de la lettre O, Apple II affiche un zéro avec une barre oblique à travers, comme il apparaît sur la touche.

L'UNITÉ À CASSETTES

Si votre système Apple comprend une unité à cassettes, vous pourrez charger vos programmes à partir de bandes magnétiques. Vous pourrez obtenir des programmes sur cassettes chez *Apple Computer Inc.* ou chez d'autres fournisseurs, pour votre machine, mais vous pouvez aussi les réaliser vous-même.

Manipulation des cassettes

Soyez prudent en manipulant les cassettes. On peut facilement les endommager et elles seront alors difficiles à remplacer.
Ne touchez pas la surface sensible de la bande magnétique. Même si vos mains vous paraissent très propres, vous la contaminerez. Remettez les cassettes dans leurs boîtes après usage. Ne les stocker jamais dans un endroit où la température est élevée, au soleil, ou à proximité d'aimants (de moteurs électriques, par exemple).

Fig. 2.2. — Les deux encoches servant de protection contre l'écriture dans une cassette.

Etiquetez chaque cassette

N'omettez pas d'inscrire, sur chaque cassette, ce qu'elle contient. Vous vous éviterez ainsi des migraines et des heures de recherche du programme souhaité.

Protection en écriture

Chaque cassette dispose, à l'arrière, de deux encoches. L'unité à cassettes, très généralement, vérifiera leur présence et dans ce cas, interdira tout enregistrement. Dans les cassettes vierges, des languettes ferment ces encoches afin d'autoriser les enregistrements. Vous pourrez protéger vos programmes en les faisant sauter.
Pour trouver la languette correspondant à une piste, retirez la cassette et tenez-la de façon que la bande soit à l'opposé de vous, comme dans la figure ; la demi-bande à protéger étant celle du dessus, la languette à ôter sera celle de droite. Il suffira de coller une bande adhésive sur l'encoche pour pouvoir à nouveau enregistrer.

Le réglage du volume

Vous devez régler la commande de volume sonore de votre unité à cassettes de façon à obtenir le niveau requis par votre ordinateur. S'il est trop bas ou trop haut, des distorsions rendront les informations incompréhensibles.
La seule façon de procéder est expérimentale. Essayez de charger un programme avec un réglage de volume très bas ; si ça ne marche pas, augmenter le volume et essayez à nouveau, et ce, jusqu'à ce que vous réussissiez le chargement.
Vous pourrez utiliser l'une des bandes livrées avec l'Apple II. Si le caractère d'appel précédant le curseur est], prenez la cassette marquée « Color Demosoft ». Si le caractère d'appel est >, prenez la « Color Graphics ».
Insérez la cassette dans l'unité à cassettes (dans le bon sens). Pour chaque réglage du volume :

1. Rebobinez complètement la cassette.
2. Sur le clavier de l'Apple II, tapez le mot LOAD (« charger »).
3. Pressez la touche PLAY (« lecture ») de l'unité à cassettes.
4. Pressez la touche RETURN sur le clavier.

Dès que vous aurez frappé RETURN, le curseur disparaît. Après 15 ou 20 secondes, vous saurez si vous avez réussi.
Si vous obtenez le message ?SYNTAX ERROR ou le message ***SYNTAX ERR, ne retouchez plus le volume mais revenez à l'étape 1 et recommencez. Si le même message réapparaît ensuite, essayez de nettoyez la tête de lecture de l'unité à cassettes, ou prenez une autre cassette.
Si rien ne se passe, ou si le message ERR ou ERRERR est affiché, pressez RESET, augmentez le volume et recommencez.
Si vous entendez un « bip » sans qu'aucun message n'apparaisse, tout va bien. Votre ordinateur a trouvé le début du programme et commence à le charger. Après 15 secondes ou davantage (cela dépend de la longueur du programme), vous entendez un second « bip », le caractère d'appel et le curseur feront leur réapparition. Vous pouvez alors arrêter la bande. Notez le réglage du volume correspondant de façon à ne pas répéter cette procédure à chaque fois.
Lorsque vous apercevez sur l'écran le caractère d'appel du Basic et le curseur, c'est que le programme a bel et bien été chargé.

L'UNITÉ À DISQUETTES DISK II

Si une ou deux unités à disquettes sont connectées à votre Apple II, vous pourrez charger vos programmes à partir de disquettes et non plus de cassettes. La disquette « System Master Diskette » fournie avec l'unité à disquettes DISK II dispose de la plupart des programmes qu'*Apple Computer Inc.* propose en cassettes, plus quelques programmes spéciaux conçus pour l'unité Disk II.

Manipulation des disquettes

Manipulez les disquettes avec précaution. Elles sont bien plus fragiles que les cassettes. *Ne les pliez jamais ; ne touchez jamais* la surface d'une disquette (celle qui apparaît dans la fenêtre de son enveloppe), et *ne forcez jamais* une disquette dans son unité Disk II.

Remettez toujours les disquettes dans leurs enveloppes lorsque vous les extrayez de l'unité Disk II et protégez-les de la chaleur, du soleil et des champs magnétiques (tels que ceux créés par des moteurs électriques). Soyez spécialement attentif à la disquette maître « System Master Diskette ».

Comment insérer une disquette dans l'unité Disk II

La meilleure façon d'introduire une disquette dans son unité « Disk II » est montrée figure 2.3. Tenez la disquette entre le pouce et l'index de façon que le pouce couvre l'étiquette. Ouvez le volet de l'unité Disk II et glissez délicatement la disquette dans l'unité. Vous ne devrez pas rencontrer de résistance. Si la disquette ne pénètre pas facilement, retirez-la et recommencez. Assurez-vous que vous maintenez bien la disquette horizontalement. Une fois la disquette en place, refermez doucement le volet de l'unité Disk II, ce qui devrait s'effectuer très facilement. Si vous rencontrez la moindre résistance, relâchez le volet et enfoncez complètement la disquette, puis essayez à nouveau. Si vous forcez le volet, vous risquez de détruire la disquette. Parfois, le lancement de la disquette (mise en marche de l'unité) facilitera son centrage ; vous pourrez alors attendre ce moment pour rabattre le volet.

LE SYSTÈME D'EXPLOITATION DISQUETTES

Avant que vous puissiez utiliser votre unité à disquettes, vous devrez charger en mémoire un programme spécial appelé « Disk Operating System », soit *système d'exploitation disques* . Ce programme, désigné par les lettres DOS, en abrégé, contrôle toutes les activités liées aux disques.
Vous devrez procéder à ce chargement à chaque remise sous tension de l'Apple II, pour amorcer le système. On dit qu'on « charge », ou qu'on « appelle » le DOS.

Fig. 2.3. — Insertion d'une disquette dans l'unité « Disk II ».

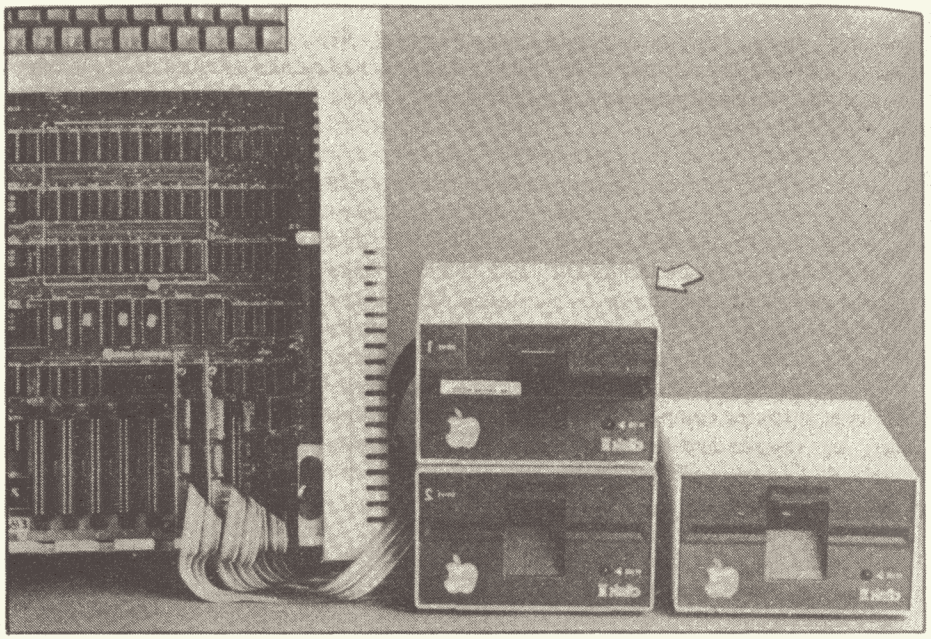


Fig. 2.4. — Unité à disquettes standard, lors du chargement du DOS.

Chargement du DOS

Il existe plusieurs façons de charger le DOS ; tout dépend de la configuration de votre système et du langage utilisé pour procéder à ce chargement. Chaque méthode suppose que l'unité à disquettes (ou « DRIVE », en anglais), et ici la première (ou « DRIVE 1 ») est reliée au connecteur « drive 1 » de la carte contrôleur, elle-même enfichée dans le connecteur femelle numéro 6.

Introduisez la « System Master Diskette » dans le bon « drive » et rabattez son volet. Dans certains cas, lorsque le « Langage system » est présent, vous aurez à employer une disquette spéciale « Integer and Applesoft II », avant la disquette DOS. Cette situation est décrite ci-dessous.

Après que vous ayez réussi à charger le DOS selon l'une des méthodes décrites ci-après, l'écran devrait offrir l'un des aspects de la figure 2.5.

Appel en « autostart »

La façon la plus simple de charger le DOS est l'auto-démarrage (« autostart »). Comme l'indique ce nom, le chargement est alors automatique. Pour qu'il soit possible, il faut que votre ordinateur dispose du moniteur « Autostart ». Si, lorsque vous mettez votre Apple II sous tension, l'unité à disquettes fait entendre des bruits divers, et si son voyant rouge « en service » (« IN USE ») est allumé, c'est que vous possédez un moniteur Autostart.

Si le moniteur Autostart est présent sans le « Système langage », le chargement du DOS est une procédure en un seul pas. L'Apple II n'étant pas sous tension, placez la disquette « System Master Diskette » dans son unité Disk II et rabattez le volet. Mettez sous tension. Après quelques secondes, la disquette s'arrêtera et l'écran présentera l'aspect de l'une des photos de la figure 2.5.

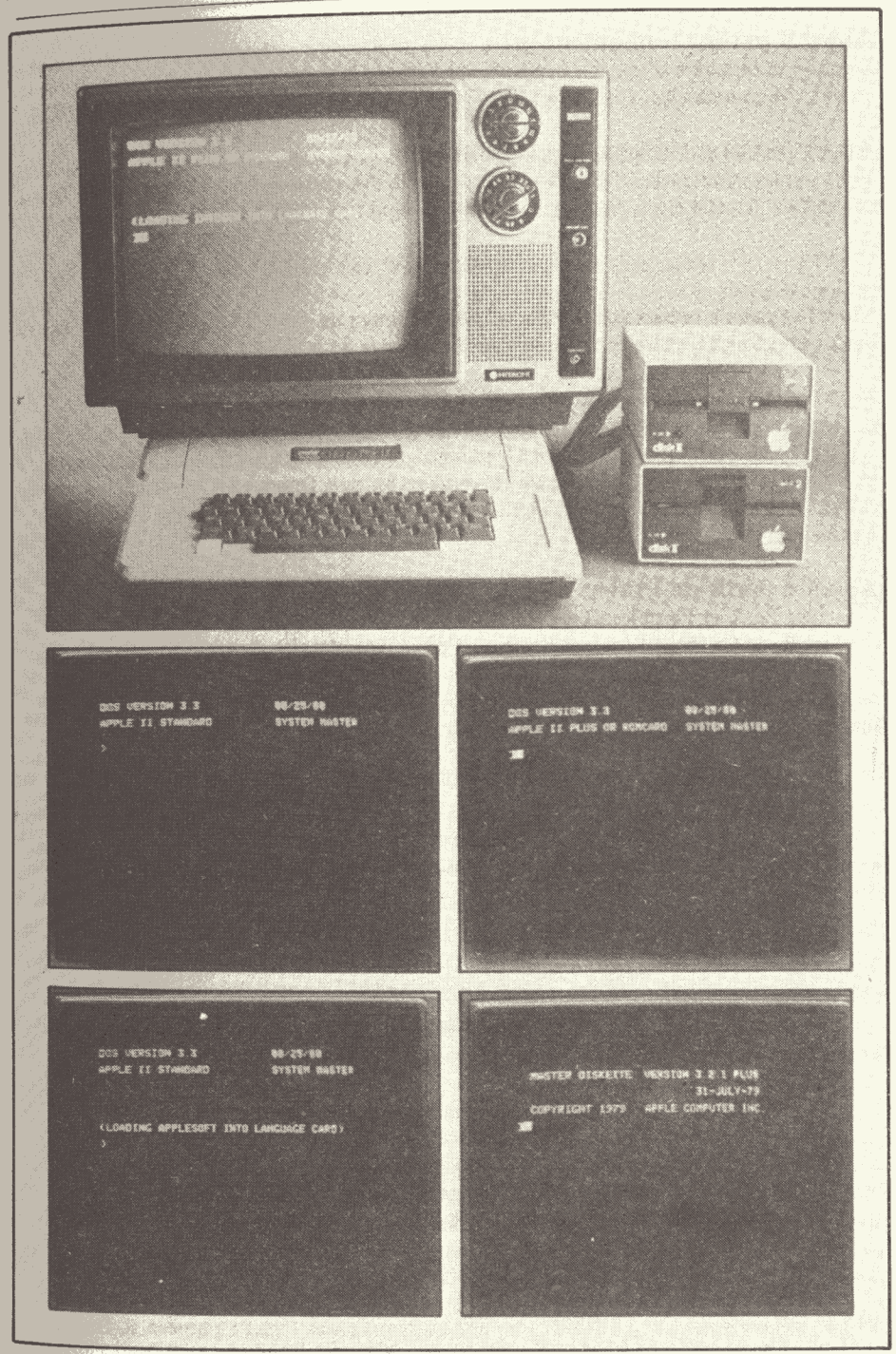


Fig. 2.5. — Aspects possibles de l'écran après un chargement réussi du DOS.

Appel à partir du moniteur

Lorsque le caractère d'appel * apparaît sur l'écran, le moniteur du langage assembleur attend vos commandes. Plusieurs méthodes vous permettront alors de charger le DOS.

Appel par branchement via le moniteur

Vous pouvez charger le DOS à partir de l'unité à disquettes 1 (« DRIVE 1 »), reliée au connecteur femelle 6 (cf. fig. 2.4) avec l'instruction moniteur suivante :

```
*C600G
```

N'oubliez pas de presser RETURN. Le voyant rouge du DRIVE 1 s'allumera et, après quelques secondes, votre écran aura l'un des aspects de la figure 2.5.

Appel à partir du moniteur par CTRL-P

La seconde commande moniteur que vous pourrez employer pour charger le DOS est CTRL-P. Pour cela, frappez d'abord le numéro du connecteur femelle auquel est connectée l'unité à disquettes (généralement, le numéro 6), puis frappez CTRL-P (rien n'apparaît sur l'écran). Pressez alors RETURN. Après quelques secondes, l'écran est semblable à l'une des photos de la figure 2.5.

Appel à partir de l'Integer Basic ou de l'Applesoft

L'Integer Basic et l'Applesoft reconnaissent les mêmes commandes. Après le caractère d'appel Basic (> pour l'Integer Basic ou] pour l'Applesoft), frappez les lettres IN ou PR, puis le caractère *dièze* (#), et enfin le numéro du connecteur femelle auquel est connectée l'unité à disquettes que vous allez utiliser (généralement, le connecteur 6). La commande devrait ressembler à l'une des deux suivantes :

```
IN#6
PR#6
```

Pressez RETURN. Après quelques secondes, votre écran affichera ce que montre l'une des photos de la figure 2.5.

Appel avec le « Système Langage »

L'appel du DOS avec le « Système Langage » (« Language System ») est, sous certaines conditions, une opération en deux pas. La version employée du DOS détermine la procédure. Chaque version est numérotée, sur la « System Master Diskette », avec des références telles que 3.3, ou 3.2.1, ou 3.2.

L'appel avec la version 3.3 du DOS est virtuellement le même avec ou sans le « Système Langage ». Utilisez la « System Master Diskette » avec l'une des procédures décrites ci-dessus. Le DOS est alors chargé en mémoire. Une autre étape se déroule automatiquement et elle rend l'Integer Basic ou l'Applesoft immédiatement disponible, via le « Système Langage ». Lorsque la disquette s'arrête, le chargement est terminé et votre écran présente l'un des aspects de la figure 2.5.

Avec les versions 3.2.1 et 3.2 du DOS, le chargement requiert deux disquettes. Vous devez d'abord placer la disquette marquée « Integer and Applesoft » dans l'unité à disquettes choisie, généralement le « DRIVE 1 » relié au connecteur femelle numéro 6. Exécutez alors l'une des procédures de chargement décrites ci-dessus. L'unité à disquettes va émettre des bruits divers, le voyant rouge « IN USE » va s'allumer, et quelques secondes

après, vous verrez apparaître sur votre écran le message suivant :

```
INSERT BASIC DISK AND PRESS RETURN
```

Relevez le volet de l'unité à disquettes, retirez la disquette et remplacez-la par la « System Master Diskette » ; rabattez le volet et pressez RETURN. Après quelques secondes, votre écran ressemblera à ce que montre l'une des photos de la figure 2.5.

Pour afficher le catalogue des programmes de la disquette

Si vous avez chargé avec succès la « System Master Diskette », vous pourrez demander l'affichage de la liste des programmes qu'elle contient. Sur le clavier, frappez :

```
CATALOG
```

L'écran devrait alors afficher quelque chose ressemblant à ce qui suit. (Au fait, n'avez-vous pas omis de frapper RETURN ?)

```
DISK VOLUME 254

*I 002 HELLO
*I 053 APPLE-TREK
*I 018 ANIMALS
*B 009 UPDATE 3.2.1
*I 014 COPY
*I 009 COLOR DEMO
*I 053 BRICK OUT
*I 026 SPACE WAR
*I 050 THE INFINITE NO. OF MONKEYS
*I 051 COLOR SKETCH
*I 053 SUPERMATH
*I 026 APPLEVISION
*I 017 BIORHYTHM
*I 027 PINBALL
```

Chargement d'autres disquettes

On vient d'expliquer comment charger dans le système le DOS de la « System Master Diskette ». La procédure serait exactement la même pour toute autre disquette insérée dans une unité Disk II. Des disquettes préparées pour d'autres ordinateurs, ou pour d'autres unités à disquettes, ne fonctionneront généralement pas avec l'unité Disk II.

PRÉPARATION DE DISQUETTES VIERGES

Parfois, vous aurez besoin de disquettes supplémentaires pour vos propres programmes. Avant d'utiliser une disquette avec une unité Disk II, pour la première fois, vous devrez l'*initialiser*. Si le programme d'application que vous employez comprend déjà des instructions spécifiques pour initialiser les disquettes, employez-les. Sinon, vous pouvez respecter les instructions suivantes, générales pour préparer les disquettes.

Le processus d'initialisation prépare la disquette pour son usage ultérieur. Pendant l'initialisation, le programme courant, présent dans la mémoire de l'Apple II est sauvegardé dans la disquette et devient le programme *de présentation* de la disquette. Ce programme

```
NEW
10      REM INITIALISATION EN DATE DU date
20      REM MEMOIRE DE octets
30      REM VERSION numéro DU DOS
40      PRINT " nom de la disquette "
50      END
```

Fig. 2.6. — Programme modèle de présentation d'une disquette.

de « salutations » est automatiquement chargé (de la disquette dans la mémoire centrale : on utilisera généralement le mot *charger* pour ce sens, le mot *sauvegarder* pour un rangement du programme de la mémoire vers la disquette) et exécuté avec la séquence de chargement. Si vous avez un programme spécial de présentation à l'esprit, fort bien ; sinon, pourquoi ne pas reprendre celui de la figure 2.6 comme modèle ? Utilisez ce programme exactement comme il a été rédigé, mais bien sûr, remplacez ce qui est écrit en italique par les informations correspondantes. La *date* est celle d'initialisation ; les *octets* sont le nombre d'octets de votre système (32 K, 36 K, etc.) ; le *numéro* est celui de la version du DOS que vous employez (quelque chose comme 3.2, ou 3.2.1, ou 3.3) ; le *nom de la disquette* est celui que vous lui attribuerez. Voici ce que pourrait donner, par exemple :

```
NEW
10 REM INITIALISATION EN DATE DU 1/4/82
20 REM MEMOIRE DE 48K
30 REM VERSION 3.3 DU DOS
40 PRINT "DIVERS, VOL. 3"
50 END
```

Pour initialiser une disquette, commencez donc par préparer ce programme de présentation, introduisez la disquette dans son unité, et frappez la commande suivante :

```
INIT HELLO
```

Assurez-vous que le volet de l'unité à disquettes a été rabaissé et pressez RETURN. Le voyant rouge s'allume, des sons bizarres et habituels se manifestent. Le processus complet d'initialisation demande environ deux minutes, aussi soyez patient. Lorsque le voyant s'éteint, l'initialisation est terminée.
Préparez maintenant une étiquette pour cette nouvelle disquette ; enlevez cette dernière de son unité et appliquez-lui l'étiquette.

CHARGEMENT ET LANCEMENT D'UN PROGRAMME

De nombreux programmes ont déjà été rédigés pour votre Apple II. Certains sont sur cassette, d'autres sur disquette, et parfois, vous les trouverez sur les deux types de supports. Certains sont écrits en Integer Basic, d'autres en Applesoft. Généralement, un programme en Applesoft ne « tournera » pas (ne fonctionnera pas) si le caractère d'appel est celui de l'Integer Basic, et les programmes en Integer Basic ne tourneront pas si le caractère d'appel est celui de l'Applesoft.

UTILISEZ LE BON BASIC

La plupart des Apple II disposent des deux versions du Basic, mais pas tous. L'Apple II Plus n'a pas l'Integer Basic, sauf si la carte « Système Langage » ou « Integer Basic » est enfichée. Par ailleurs, l'Applesoft n'est pas disponible sur de nombreuses versions de l'Apple II tant que vous n'avez pas chargé son interpréteur à partir d'une cassette ou d'une disquette.
Si votre système Apple II est équipé de la carte « Language System » (« Système Langage ») ou de la carte « Applesoft Firmware », il emploiera automatiquement la version Basic convenant au programme que vous avez chargé. Sinon, il faudra vérifier la conformité entre le caractère d'appel et le langage du programme sur disque ou cassette que vous allez charger.
Sur un Apple II standard, il est facile d'obtenir le caractère d'appel de l'Integer Basic (>). Pressez RESET, puis frappez CTRL-B (sans oublier de terminer avec un RETURN). Il est plus difficile d'obtenir le caractère d'appel de l'Applesoft (]) sur un Apple II standard, car ce langage réside en disque ou en cassette. Vous devrez commander à votre machine le chargement de l'interpréteur Applesoft dans sa mémoire centrale.
Avant de faire fonctionner l'unité à disquettes, il vous faudra charger le DOS comme décrit antérieurement dans ce chapitre. Dès lors, vous pourrez charger l'interpréteur Applesoft en plaçant la « System Master Diskette » dans l'unité Disk II puis en frappant la commande.

```
FP
```

L'unité à disquettes va émettre des bruits pendant quelques secondes, puis le caractère d'appel de l'Applesoft (]) apparaîtra sur l'écran.
Vous pouvez aussi disposer de l'interpréteur Applesoft sur cassette. Employez la cassette marquée « Applesoft II » (de *Apple Computer Inc.*) ; placez-la dans l'unité à cassettes, ré-embobinez-la, puis frappez la commande :

```
LOAD
```

Avant de presser RETURN, mettez l'unité à cassettes en position de lecture, en enfonçant la touche PLAY. Dès que vous entendez un « bip » de l'Apple II, vous saurez que le chargement a commencé. Il demande entre 1,5 à 2 minutes. Dès que ce chargement est terminé, un second « bip » se fait entendre. Stoppez l'unité à cassettes. Le caractère d'appel de l'Applesoft devrait être sur l'écran.
A partir de l'Applesoft, vous pouvez passer à l'Integer Basic en frappant la commande :

```
INT
```

Si vous vouliez ensuite revenir à l'Applesoft, vous devriez recharger son interpréteur à partir de la disquette ou de la cassette.

CHARGEMENT D'UN PROGRAMME À PARTIR D'UNE CASSETTE

Dès lors que vous avez choisi la version du Basic vous convenant (à moins, donc, que votre Apple II ne s'aligne automatiquement sur celle de votre programme d'application), voici les pas à respecter pour charger un programme à partir d'une cassette :

1. Positionnez la bande au départ du programme. Ce sera généralement au début de la bande, dans lequel cas il vous faudra la ré-embobiner complètement. Si votre programme n'est pas le premier de la cassette, vous devrez charger successivement tous les programmes qui précèdent. Répétez les pas suivants pour tous les programmes que vous devrez sauter.

2. Sur le clavier de l'Apple II, frappez la commande :

```
LOAD
```

3. Enfoncez la touche de lecture PLAY de l'unité à cassettes.

4. Pressez la touche RETURN.

Le curseur va alors disparaître. Après quelques secondes, l'Apple II va émettre un « bip » signifiant qu'il a commencé le chargement. Peu de temps après, il va en émettre un second, signalant ainsi qu'il a terminé cette opération. Enfoncez la touche STOP de l'unité à cassettes.
Le programme est maintenant chargé. Si vous n'obtenez aucun « bip » ou si l'écran affiche un message d'erreur, vérifiez le réglage de volume comme indiqué précédemment dans ce chapitre. Si cela ne marche toujours pas, c'est probablement que la bande magnétique s'est détériorée ; il ne vous restera plus qu'à la remplacer.

CHARGEMENT D'UN PROGRAMME À PARTIR D'UNE DISQUETTE

Une fois le DOS chargé, vous pouvez charger un programme stocké sur une disquette en frappant une commande telle que :

LOAD *nom du programme*

Bien entendu, remplacez *nom du programme* par son nom. Il faudra, aussi, qu'un programme de ce nom soit présent sur la disquette introduite dans l'unité à disquettes.

LANCEMENT D'UN PROGRAMME

Lorsque votre programme est chargé, utilisez la commande suivante pour en lancer l'exécution :

```
RUN
```

Ce programme prend en charge la commande de l'Apple II, incluant le clavier et l'écran. Pour reprendre la direction de votre ordinateur, vous pourrez le plus souvent frapper CTRL-C, puis RETURN également. Si rien ne se produit, vérifiez les instructions spécifiques du programme en cours d'exécution. En cas d'urgence, il vous reste toujours la possibilité de frapper RESET ou de couper un instant l'alimentation de votre ordinateur, mais dans les deux cas, vous perdrez votre programme.

RÉGLAGE DE LA COULEUR AVEC UN RÉCEPTEUR TV

L'Apple II peut traiter du graphique en couleur. Si l'un de vos programmes fait appel à cette possibilité, vous devrez ajuster les couleurs sur votre récepteur TV ou sur votre moniteur pour en obtenir une bonne distribution. La société *Apple Computer Inc.* a réalisé un programme spécial pour vous assister dans cette tâche. Avec l'Integer Basic (caractère d'appel >), employez la cassette « Color Graphics » ; avec l'Applesof (]), la « Color Demosoft » ; en disquette, utilisez le programme COLOR DEMOSOFT. Dans les deux cas, le chargement et le lancement du programme se fait comme indiqué plus haut. L'écran devrait afficher ce que montre la figure 2.7.

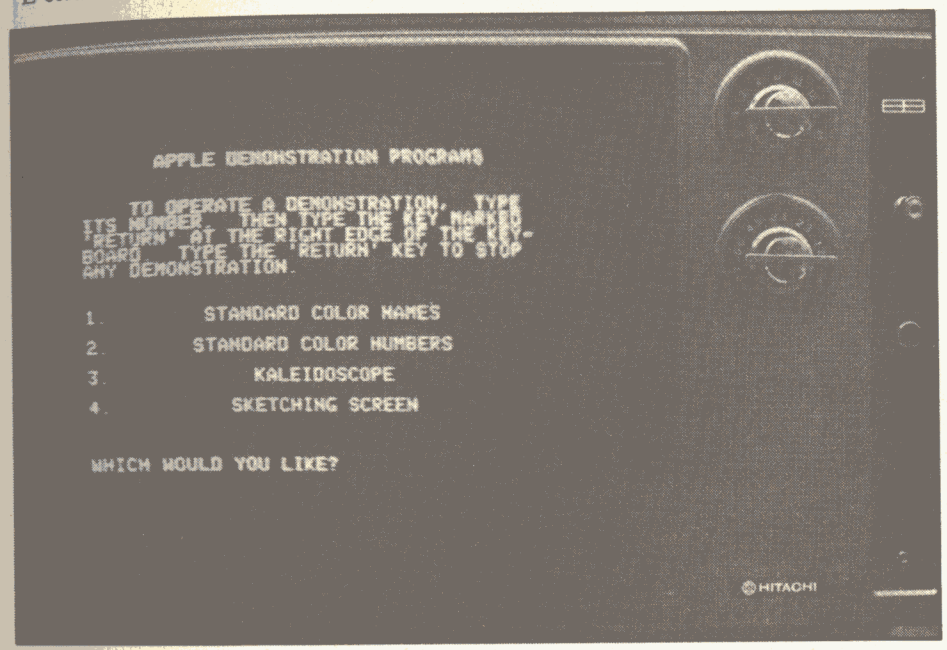


Fig. 2.7. — Le menu du programme COLOR DEMOSOFT.

Ce que vous voyez sur cette figure s'appelle un *menu*. C'est une liste offrant un choix (ici, de quatre possibilités) et vous proposant d'exprimer ce choix. Pour régler votre récepteur couleur, choisissez 1 et frappez la touche 1, puis RETURN. L'écran ressemblera alors à ce que montre (mais en noir !) la figure 2.8 et exposera des bandes verticales colorées. Comme vous pouvez le constater, chaque couleur se voit affecter d'un nom, en bas de l'écran. De gauche à droite, ces couleurs sont :

Code	Anglais*	Français	Code	Anglais*	Français
BLAK	Black	Noir	BROWN	Brown	Brun
MGTA	Magenta	Magenta	ORNG	Orange	Orange
DBLU	Dark Blue	Bleu foncé	GREY	Grey	Gris
PURP	Purple	Pourpre léger	PINK	Pink	Rose-crème
DGRN	Dark Green	Vert foncé	LGRN	Light Green	Vert clair
GREY	Grey	Gris	YELO	Yellow	Jaune
MBLU	Medium Blue	Bleu moyen	AQUA	Aqua	Bleu-vert
LBLU	Light Blue	Bleu clair	WITE	White	Blanc

* Note : Les noms anglais explicitent les codes.

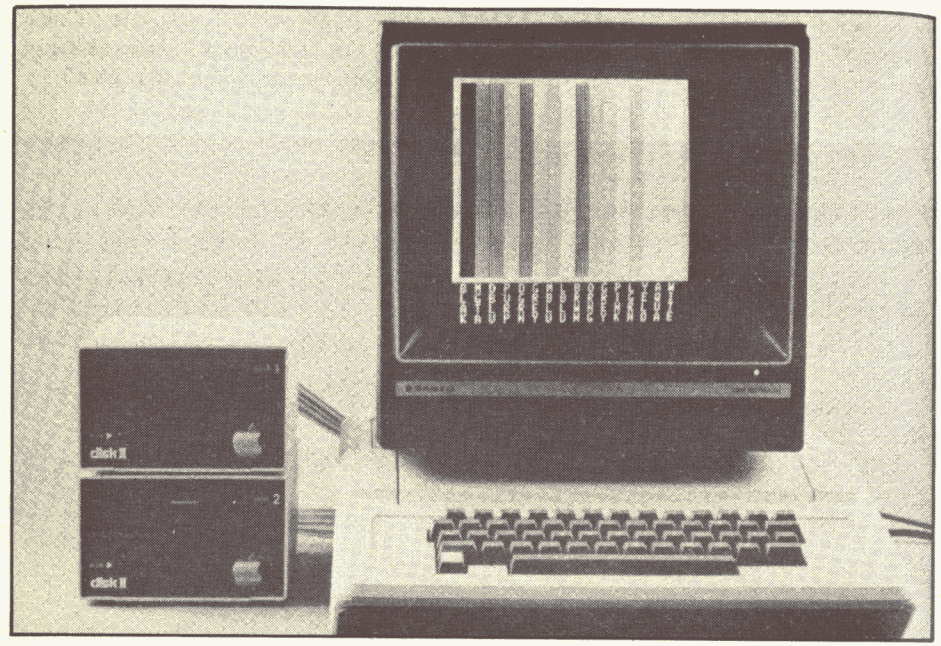


Fig. 2.8. — Pour régler les couleurs sur votre écran.

Ajustez maintenant le contraste, la luminosité et les commandes de teintes de votre téléviseur jusqu'à ce que vous obteniez des couleurs répondant à leurs noms et acceptables. Soignez particulièrement le pourpre, le rose, le jaune, et les trois bleus.
Quand vous en aurez terminé, pressez RETURN ; le menu réapparaît. Vous pourrez alors choisir une autre option que vous souhaiterez expérimenter. Pour « sortir » de ce programme, frappez CTRL-C (il vous faudra conclure par RETURN).

MATÉRIELS DIVERS

On vient ainsi d'en terminer avec les instructions spécifiques, relatives aux matériels constituant votre système. Si vous en possédez d'autres, référez-vous à leurs manuels d'utilisation. Si, par exemple, vous disposez d'une imprimante, il vous faudra absolument vous en tenir à son manuel d'emploi car les modes d'emploi des imprimantes varient grandement d'un type à l'autre.

ET LES ERREURS ?

L'Apple II est un merveilleux outil, mais il partage un même problème avec tous les ordinateurs : il manque d'imagination. Chaque instruction que vous lui prodiguez doit être parfaitement correcte, faute de quoi il n'exécutera pas ce que vous attendez de lui. Le résultat d'une erreur va d'un ennui passager au saccage du programme.

MESSAGE D'ERREUR

Si vous vous trompez en frappant un ordre, et pressez RETURN, votre ordinateur vous répondra généralement par un « bip » et quelque chose à décrypter mais qui est un *message d'erreur*. Souvent, ce message contiendra une indication sur le type d'erreur que vous avez commise ; mais ce n'est nullement obligatoire. Le remède général est le même dans les deux cas : retapez la ligne. Vous trouverez la liste complète des messages d'erreur dans l'appendice C.

CORRECTION DES ERREURS DE FRAPPE

Parce que vous tapez sur les touches du clavier, vous risquez de commettre des erreurs de frappe. Certaines des touches que nous avons décrites antérieurement faciliteront la correction de ces erreurs, si vous les avez décelées avant de presser RETURN. Ce sont les touches ← , →, REPT, CTROL-X et ESC-@ :

- La touche ← déplace le curseur en arrière et efface de la mémoire les caractères sur lesquels le curseur passe, bien que ceux-ci restent affichés.
- La touche → déplace le curseur vers la droite, recopiant les caractères sur lesquels le curseur passe.
- La touche REPT, utilisée avec ← ou →, permet des déplacements rapides, en avant et en arrière.
- La double commande CTRL-X annule la ligne que vous venez de frapper.
- La séquence ESC-@ efface l'écran et ramène le curseur en haut et à gauche.

Examinons comment vous pourriez recourir à ces caractéristiques d'édition. Supposons que vous vouliez frapper :

```
LOAD COLOR DEMOSOFT
```

mais par mégarde, vous commettez :

```
LOAS COLOR DEMOSOFT
```

Vous avez deux possibilités. Vous frappez CTRL-X pour annuler la ligne et vous recommencez la frappe, ou vous utilisez ← pour effectuer la correction. Pressez et maintenez enfoncée la touche ←, puis pressez REPT. Le curseur fait marche arrière ; dès qu'il arrive sur la lettre erronée, lâchez la touche REPT. Rappelez-vous qu'au passage, il aura effacé de la mémoire tous les caractères qu'il aura croisés. Si vous dépassez la lettre erronée, repartez de l'avant avec → pour ramener le curseur sur ce malencontreux S. Pressez alors un D, le curseur se déplace et *vous* lisez une ligne correcte :

```
LOAD COLOR DEMOSOFT
     ↑(c'est le curseur)
```

Mais savez-vous ce qui se passerait si vous pressiez maintenant RETURN ? Tous les caractères à la droite du curseur s'évanouiraient et il ne subsisterait sur l'écran que :

```
LOAD
```

Cela ne va plus ! Avant de frapper RETURN, vous devez ramener le curseur à la fin de la ligne (ou refrapper le reste, mais vous risqueriez de commettre une nouvelle erreur !). La

touche →, conjointement avec REPT, déplacera le curseur vers la droite en recopiant en mémoire tous les caractères traversés.

RESET ACCIDENTEL

Tôt ou tard, vous frapperez accidentellement sur RESET alors que vous n'en n'aviez nullement l'intention. C'est inévitable (mais votre Apple II peut fort bien exiger une double frappe, CTRL-RESET, au lieu de simplement RESET). Pour éliminer ce risque, ôtez le capuchon en plastique de la touche RESET afin que l'axe de cette touche apparaisse (fig. 2.9). Dès lors, une frappe accidentelle restera moins probable.
Vous pouvez aller plus loin encore, en mettant une petite bague élastique (de joint) autour de cet axe afin qu'une résistance s'offre à la pression du doigt (fig. 2.10) ; puis, replacez le capuchon.

Après un RESET accidentel

Ce que vous devrez faire après un RESET accidentel dépend de ce que vous faisiez lorsque vous l'avez inopportunément provoqué, et du type de votre Apple II.
Chaque programme que vous lancez sur votre machine devrait disposer d'instructions spécifiant ce qu'il convient de faire dans ce cas. N'oubliez pas, alors, de lire ces instructions avant de lancer le programme. Si vous pressez RESET en exécutant un programme en Basic, il vous faudra le reprendre à son début. Mais si vous traitiez une gestion, par exemple, vous aurez peut-être perdu des données que vous aviez déjà traitées.

Fig. 2.9. — Pour vous protéger d'un RESET accidentel : phase 1.

Lorsque vous pressez RESET, Apple II stoppe immédiatement ce qu'il était en train de faire. Le contrôle revient au clavier ; vous verrez apparaître le caractère d'appel et le curseur clignotant au bas de l'écran. Si le caractère d'appel est celui du Basic, vous pourrez redémarrer le programme qui était en cours d'exécution avec une commande RUN. Une partie de ce que vous aviez fait depuis le RUN précédent se sera probablement perdu.
Si le caractère d'appel apparaissant après votre malencontreux RESET est celui du moniteur (*), vous pourrez revenir au Basic en frappant CTRL-C, *sauf* si vous étiez en train d'utiliser l'Applesoft en cassette ou en disquette. Pour passer du moniteur à l'Applesoft de la disquette, frappez la commande moniteur :

```
*3D0G
```

Pour passer du moniteur à l'Applesoft de la cassette, frappez la commande :

```
*0G
```

ATTENTION : Assurez-vous que vous employez bien la bonne commande. Si vous hésitez, prenez conseil auprès de quelqu'un habitué à ce genre d'exercice. Une mauvaise commande aggravera votre cas : il vous faudra éventuellement rechercher votre programme, l'Applesoft, et peut-être aussi le DOS.

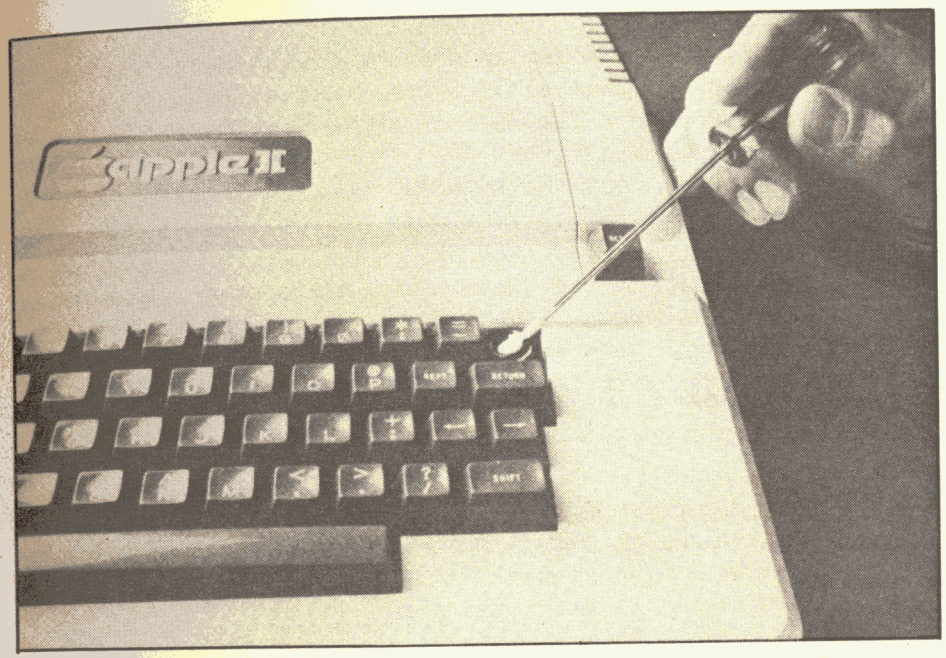


Fig. 2.10. — Pour vous protéger d'un RESET accidentel : phase 2.

CHAPITRE 3

PROGRAMMATION EN BASIC

Ce chapitre vous enseigne comment commencer à rédiger vos propres programmes en Basic pour l'Apple II.
Le Basic est un langage de programmation. Comme tout autre langage, il constitué par un jeu d'instructions que vous pouvez combiner pour créer des programmes. Un programme définit une tâche que votre ordinateur exécutera.
Nous pourrions vous apprendre le Basic en vous faisant apprendre ces instructions, une par une. Mais vous abandonneriez alors probablement très vite, car ces instructions individuelles n'ont guère de signification. L'étude des instructions Basic distinctes dégénère rapidement et devient l'étude d'un recueil de règles qui ne vous disent rien de la programmation et de sa bonne pratique. Aussi, l'énoncé rigoureux et la définition de toutes les instructions Basic ont-elles été reléguées au chapitre 8. Reportez-vous à ce chapitre si nécessaire, mais ne l'abordez pas avant d'avoir lu celui-ci.

POUR DÉMARRER AVEC LE BASIC

Deux versions du Basic sont disponibles sur l'Apple II. Certains modèles disposent de l'Integer Basic (Basic entier), d'autres de l'Applesoft, et d'autres, enfin, des deux. Pour l'instant, vous n'avez pas à vous soucier de savoir quel Basic traite votre machine. Ultérieurement, lorsque les différences deviendront significatives, nous en discuterons en détail.

Au début était le caractère d'appel

L'Apple II est un ordinateur multilingue. Si vous voulez programmer en Basic, il doit pouvoir accepter les instructions dans ce langage. Vous trouverez, dans le chapitre 2, comment démarrer votre système avec le Basic ; rappelons ces règles.
Avec un Apple II équipé d'un moniteur « Autostart », vous n'avez qu'à mettre la machine sous tension et presser RESET (ou CTRL-RESET sur certaines versions).
Si votre Apple II ne possède pas de moniteur « Autostart », mettez-le sous tension puis pressez CTRL-B, et RETURN.
Le caractère d'appel, à gauche de l'écran et à côté du curseur, vous indique quel Basic est disponible. C'est l'Integer Basic pour un « > », l'Applesoft pour un «] ».

MODES IMMÉDIAT ET PROGRAMMÉ

Avant de nous préoccuper de savoir comment passer d'un Basic à l'autre, examinons quelques instructions simples que vous pourrez employer avec les deux versions, Integer Basic et Applesoft. Certains des exemples qui suivent montrent, comme caractère d'appel, « > », et d'autres, «] ». Vous pourrez expérimenter ces exemples avec l'une et l'autre version du Basic, quel que soit le caractère d'appel indiqué, *sauf* si cela est expressément spécifié. Les exemples sans caractère d'appel s'appliquent aussi bien aux deux Basic.

L'AFFICHAGE DE CARACTÈRES

Lorsque vous démarrez avec votre Apple II en Basic, l'ordinateur est en *mode immédiat,* appelé aussi mode direct, mode commande, ou mode calculette. Il répond immédiatement aux ordres reçus. Essayez de frapper :

```
PRINT "CHIENS ET CHATS"
```

N'oubliez pas de presser RETURN après avoir fermé les guillemets. Votre Apple II répond en affichant :

```
CHIENS ET CHATS
```

S'il affiche un message ?SYNTAX ERROR ou *** SYNTAX ERR, c'est pour vous indiquer que votre commande était indéchiffrable. Vous avez probablement mal frappé le mot PRINT. Si Apple II affiche le nombre 0 au lieu d'un message, cela signifie que vous avez omis d'ouvrir les parenthèses. Dans les deux cas, vous pourrez refrapper votre ordre mais avec plus d'attention, cette fois ! Les ordinateurs sont très sensibles à l'orthographe et à la ponctuation ; la plus légère erreur peut contrarier votre machine ou l'entraîner à exécuter un ordre inattendu.
Une commande telle que la précédente ordonne à la machine d'afficher sur l'écran tout ce qui est inclus dans les guillemets.
Il y a une limite à la longueur de ce message. Cette limite diffère avec l'Integer Basic et l'Applesoft, mais dans les deux cas, excède la largeur de l'écran. Cela signifie qu'une commande peut occuper plus d'une ligne d'affichage. Des commandes trop longues, comme celle qui suit, repartiront à la ligne automatiquement. Essayez cette commande :

```
PRINT "CET HOMME DEVRAIT NORMALEMENT ETR
E ESTIME FOU"
CET HOMME DEVRAIT NORMALEMENT ETRE ESTIM
E FOU
```

L'Integer Basic autorise jusqu'à 120 caractères environ. Si vous dépassez cette limite, vous vous verrez adresser, après avoir pressé RETURN, le message ***TOO LONG ERR (« TOO LONG » signifie *trop long*).
L'Applesoft accepte jusqu'à 255 caractères. Lorsque vous approcherez cette limite, la machine émet des « bip ». Si vous la franchissez, une barre oblique inversée (\ ) est affichée et elle annule automatiquement votre entrée, comme si vous aviez frappé CTRL-X.

L'AFFICHAGE DE FORMULES

Vous pouvez utiliser votre Apple II en mode immédiate comme une calculatrice. Il répond directement à vos ordres. Vérifiez-le avec ces exemples.

```
PRINT 4+6                    Addition
10

PRINT 500-437                Soustraction
63

PRINT 100*23                 Multiplication
2300

PRINT 96/12                  Division
8

PRINT 3^2                    Exponentiation
9

PRINT 3*4*10-800             Expression
-680
```

Les réponses correctes figurent sur la ligne immédiatement en-dessous de la question. Remarquez qu'on n'utilise pas de guillemets ici. Savez-vous ce qui se passerait si vous placiez des guillemets ? Vérifiez-le si vous n'êtes pas sûr de la réponse.
L'Integer Basic limite les maximums et les minimums dans les calculs. Si la valeur calculée excède, à n'importe quel moment, 32767, le message d'erreur ***>32767 ERR est affiché ; si la valeur est inférieure à −32767, c'est le message −***>32767 ERR qui apparaît. Voici quelques exemples d'erreurs, le dernier résultant d'une division par zéro :

```
>PRINT -32766-2
-*** >32767 ERR
>PRINT 2^15-1
*** >32767 ERR
>PRINT10/0
*** >32767 ERR
```

Les fractions décimales ne sont pas admises en Integer Basic (qui tire son nom de ce fait). Aussi, si vous exécutez une division qui ne tombe pas juste, le reste sera perdu. Par exemple, essayez de faire :

```
>PRINT 9/2
4
```

L'Applesoft, lui, accepte les fractions. Les valeurs numériques peuvent avoir jusqu'à 9 *chiffres significatifs,* y inclus les parties entières et décimales. Cela signifie que des valeurs avec plus de neuf chiffres significatifs seront arrondies et ramenées à neuf chiffres, ou moins :

```
]PRINT 12.34567896           Arrondi supérieur
12.345679

]PRINT 12.34567894           Arrondi inférieur
12.3456789
```

Si vous faites d'autres essais en Applesoft, en mode immédiat, vous remarquerez que le résultat est parfois donné en notation scientifique :

```
JPRINT 123456789123
1.23456789E+11
```

Si la notation scientifique ne vous est pas familière, bornez-vous à des calculs simples pour l'instant. Nous y reviendrons.

L'ordre PRINT abrégé

L'Applesoft vous permet de remplacer l'ordre PRINT par le symbole dollar ($). En voici quelques exemples que vous pouvez essayer :

```
J?"LE TEMPS S'ECOULE"
LE TEMPS S'ECOULE

J?13-46*6
-263
```

MESSAGES D'ERREUR

On a déjà fait état de divers messages d'erreur que l'Apple II délivre lorsqu'il ne comprend pas un ordre. Il émet un « bip » pour attirer votre attention sur le fait que quelque chose ne va pas, et il tente d'en établir le diagnostic pour vous. Mais ses possibilités de diagnostic sont limitées, aussi vous ne devrez pas en attendre une analyse définitive de votre erreur. Votre Apple II dispose de moins de 35 messages d'erreur possibles, pour des myriades de possibilités d'erreurs, ou de combinaisons d'erreurs.
Tout au long de ce chapitre, et dans les suivants, nous noterons les erreurs pouvant survenir dans des situations spécifiques, surtout les plus insidieuses. Tous les messages d'erreur sont listés dans l'appendice C.

Format des messages d'erreur

Les messages d'erreur se présentent sous un format sensiblement différent en Integer Basic et en Applesoft, comme cela apparaît ci-dessous :

```
JPIRNT "AU CLAIR DE LA LUNE"

PIRNT "AU CLAIR DE LA LUNE"
*** SYNTAX ERR
>
```

Espaces supplémentaires

Êtes-vous en train de vous débattre avec le problème des espaces, à savoir : où faut-il mettre des « blancs » (des espaces) dans une ligne, et où ne le faut-il pas ? Ne vous en inquiétez pas. Apple II interprète une ligne à partir de ses éléments, qu'il y ait ou non des espaces. Le seul endroit où il vous faudra en introduire obligatoirement, c'est dans des guillemets si vous voulez que votre message en comprenne.

INSTRUCTIONS, LIGNES ET PROGRAMMES

Un programme se compose d'une ou plusieurs instructions qui fournissent à l'Apple II une définition complète de la tâche à accomplir. Si cette tâche est courte et simple, le programme qui en résultera pourra être court et simple. Les ordres en mode immédiat que nous venons d'expérimenter constituent chacun un programme, simple et court, constitué par une seule instruction à l'Apple II. Mais la plupart des programmes sont composés de 10, 100, 1000 instructions, ou même davantage. Considérez les instructions suivantes :

```
PRINT "AU CLAIR"
AU CLAIR

PRINT "DE LA LUNE"
DE LA LUNE

PRINT "MON AMI PIERROT"
MON AMI PIERROT
```

Chacune de ces instructions en mode immédiat imprime une ligne de texte sur l'écran. Chaque programme est ainsi composé d'une instruction, sur une ligne.
Applesoft vous permet d'écrire plusieurs instructions sur une ligne. Vous séparerez ces instructions par un double point (:). Comparez ce programme en mode immédiat avec le précédent.

```
JPRINT "AU CLAIR" : PRINT "DE LA LUNE" :
 PRINT = MON AMI PIERROT
AU CLAIR
DE LA LUNE
MON AMI PIERROT
```

Cette triple instruction en une seule ligne affiche exactement le même texte que les trois précédentes.
Il n'y a pas de limite au nombre d'instructions qu'on peut porter dans une ligne de programme. Rappelez-vous cependant qu'une ligne ne peut excéder 255 caractères. Si vous frappez une telle ligne, votre ordinateur va commencer à émettre des « bip » à partir du 248e caractère afin de vous mettre en garde ; si, néanmoins, vous franchissez cette limite, il annule automatiquement toute la ligne, tout comme si vous aviez composé CTRL-X ; vous devrez tout recommencer. Apple II n'exécutera aucune des instructions de cette ligne excédentaire annulée. Ainsi, telle sera la limite d'une série d'instructions sur une ligne en mode immédiat.

Une ligne de programme en Applesoft

On peut cumuler beaucoup de choses dans une ligne de programme en mode immédiat, grâce à cette faculté de l'Appelsoft d'accepter 255 caractères par ligne. Considérez, par exemple, l'instruction suivant§e.

```
JFOR I=1 TO 800:?"A";:NEXT:?" BOF!"
```

Ignorons, pour l'instant, la signification de ce mini-programme ; frappez-le très exactement tel qu'il est, terminez-le par un RETURN, et vous verrez apparaître 20 lignes et 40 colonnes de A, suivis par un « BOF ! » sur la 21e ligne :

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BOF !AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

La ligne du programme est restée en haut de l'écran, cela parce que l'écran affiche juste assez de lignes pour tout loger. Lorsque l'exécution de ce programme est terminée, le caractère d'appel de l'Applesoft avec son curseur réapparaît à la ligne inférieure.

MODE PROGRAMMÉ

Les programmes précédents étaient didactiques et, nous l'espérons, intéressants, mais limités par le mode immédiat. C'est d'autant plus vrai avec l'Integer Basic qui, lui, n'accepte qu'une instruction par ligne. Un autre problème surgit avec les programmes en mode immédiat : vous devez les refrapper à chaque exécution. Des méthodes évoluées d'éditeur, dont nous discuterons plus loin, vous permettraient de ré-utiliser le programme tant qu'il apparaît sur l'écran, mais là encore, on est vite limité.

Ce dont nous avons besoin, c'est d'une méthode permettant l'introduction de plusieurs lignes de programme et leur stockage. De cette façon, on pourra rédiger des programmes capables d'exécuter des tâches trop complexes pour être compactées en une seule ligne. Pour cela, on va passer du mode immédiat au *mode programme,* appelé aussi mode indirect. En mode programme, l'ordinateur accepte et range les instructions dans sa mémoire, mais n'en exécute aucune tant qu'on ne lui en a pas donné l'ordre. Vous pouvez introduire autant de lignes de programme que vous le désirez. Puis, vous frapperez la commande appropriée et l'ordinateur exécutera les opérations spécifiées par le programme.

Exécution du programme

On dit qu'un ordinateur exécute un programme lorsqu'il accomplit les opérations spécifiées par ce programme. On dit alors que « le programme tourne sur la machine ». En mode immédiat, le programme est exécuté dès qu'on frappe RETURN.

En mode programme, vous devrez frapper la commande RUN pour lancer l'exécution. Chaque fois que vous frapperez RUN, vous relancerez l'exécution du programme.

Élimination des anciens programmes

Parce que l'Apple II stocke tous les programmes dans sa mémoire, vous devrez lui ordonner d'effacer les anciens programmes avant d'en introduire un nouveau. L'ordre est, ici, NEW (pour « nouveau »). Si vous omettez de frapper NEW, votre nouveau programme risque de se mélanger avec les anciens.

Pour terminer correctement un programme

La fin d'un programme en mode immédiat est évidente, mais elle ne l'est plus en mode programme, comme on va le voir. Aussi emploiera-t-on l'instruction END (« fin ») pour en stopper l'exécution et revenir au mode immédiat. END sera, ainsi, la dernière instruction exécutée par votre programme.

L'Applesoft peut se passer de END ; il termine un programme automatiquement dès lors qu'il n'y a plus d'instructions.

Numérotation des lignes

La *numérotation des lignes* fait entrer la machine en mode programme. Un numéro de ligne est simplement un nombre à un, deux, trois, quatre ou cinq chiffres placé au début d'une ligne. Ce numéro constitue la seule différence entre le mode immédiat et le mode programme. Vous trouverez des instructions qu'on ne peut employer qu'en mode immédiat, et d'autres qu'en mode programmé ; nous en discuterons ultérieurement.

Essayez ce programme en mode programmé :

```
>NEW

>10 PRINT "ADIEU COCHON, COUVEE"
>20 END
>RUN
ADIEU COCHON, COUVEE
```

Chaque numéro de ligne ne doit être utilisé qu'une seule fois ; on n'a pas le droit d'écrire deux lignes de programmes avec le même numéro. Si vous reprenez le même numéro une seconde fois, c'est cette seule ligne dont se souviendra l'ordinateur ; la précédente (avec le même numéro) sera perdue. Pour le vérifier, faites :

```
>NEW

>10 PRINT "PREMIERE LIGNE 10"
>10 PRINT "DEUXIEME LIGNE 10"
>20 END
>RUN
DEUXIEME LIGNE 10
```

Les numéros de lignes déterminent l'ordre de la séquence à exécuter dans un programme en Basic. La première instruction doit posséder le numéro le plus faible, et la dernière, le plus fort. Si vous frappez des lignes dans le désordre, Apple II les ré-organisera et les classera dans l'ordre. Faites, par exemple :

```
>NEW

>30 PRINT "RATON"
>10 PRINT "ET"
>20 PRINT "UN"
>40 PRINT "LAVEUR"
>50 END
>RUN
ET
UN
RATON
LAVEUR
```

Pour vous prouver qu'Apple II n'oublie pas les instructions en mode programme, effacez l'écran avec ESC-@ et relancez le programme :

```
>    (Pressez ESC - @, puis RETURN)

>RUN
ET
UN
RATON
LAVEUR
```

C'est très simple d'ajouter des lignes à un programme déjà en mémoire. Vous pouvez ajouter une ligne au début, à la fin, ou n'importe où dans le corps du programme et la frapper, précédée par un numéro qui la placera à l'endroit voulu. Supposez que vous vouliez ajouter une ligne au début du programme précédent. Tant que vous n'avez pas fait NEW, il est resté en mémoire. Puisque le numéro le plus faible était 10, frappez un nombre inférieur à 10 en début de la nouvelle ligne ; par exemple, essayez ceci :

```
>5 PRINT "MAIS C'EST DU PREVERT"
>RUN
MAIS C'EST DU PREVERT
ET
UN
RATON
LAVEUR
```

Vous apprécierez ainsi que le programme commence en 10 et non en zéro ! C'est une bonne habitude à prendre que de commencer les programmes avec un numéro de ligne assez élevé, qui laissera assez de place à des lignes supplémentaires éventuelles.

Lignes multi-instructions

A nouveau, on peut inclure plusieurs instructions dans une ligne de programme. La première instruction suit le numéro de la ligne ; la seconde intervient, mais après un double point (:) de séparation. Les doubles points séparent les instructions d'une même ligne. L'Integer Basic vous permet d'utiliser plusieurs instructions par ligne, en mode programme (mais pas en mode immédiat). La ligne est limitée à 150 caractères environ, cette limite dépendant du contenu de la ligne. Si vous voulez introduire une ligne trop longue, Apple II vous retourne un message d'erreur ***TOO LONG ERR (avec « TOO LONG » pour *trop long*) et vous devez refrapper la ligne.

Les lignes multi-instructions sont autorisées en Applesoft, aussi bien en mode immédiat qu'en mode programmé. Dans les deux cas, la longueur maximale de caractères de 255, ainsi qu'on l'a déjà indiquée.

Listage des lignes du programme

Vous pourrez afficher la liste des instructions de votre programme, rangé en mémoire, à tout moment, en frappant la commande LIST. Essayez. Si vous n'avez pas introduit un NEW entre-temps, vous devriez voir sur l'écran :

```
LIST
5 PRINT "MAIS C'EST DU PREVERT:"
10 PRINT "ET"
20 PRINT "UN"
30 PRINT "RATON"
40 PRINT "LAVEUR"
50 END
```

On appelle cela un *listage* (en anglais : *listing*). Des variantes de la commande LIST vous serviront à n'afficher qu'une ligne, ou un groupe de lignes. Cette dernière option est intéressante lorsque le programme est trop long pour tenir en totalité sur l'écran. Si le programme ci-dessus est toujours en mémoire, frappez la commande :

```
LIST 10
```

et vous verrez apparaître sur l'écran la seule ligne 10.

```
10 PRINT "ET"
```

Si vous spécifiez le numéro d'une ligne de départ et celui d'une ligne d'arrivée, vous provoquerez l'affichage de la tranche comprise entre ces deux numéros, ceux-ci compris :

```
LIST 20, 40
20 PRINT "UN"
30 PRINT "RATON"
40 PRINT "LAVEUR"
```

En Applesoft, on peut afficher les lignes du programme jusqu'à (et y compris) une ligne donnée. Ou encore, à partir de, et jusqu'à la fin du programme. En voici deux exemples :

```
]LIST,10

5 PRINT "MAIS C'EST DU PREVERT:"
10 PRINT "ET"

]LIST30,

30 PRINT "RATON"
40 PRINT "LAVEUR"
50 END
```

Pour interrompre un listage

On peut interrompre un listage en cours d'affichage sur l'écran en frappant CTRL-C. Ce sera particulièrement utile pour stopper le déroulement d'un programme très long.
Si votre Apple II dispose d'un moniteur Autostart, vous pourrez suspendre, ou geler temporairement le listage d'un programme en frappant CTRL-S. Le listage reprendra dès que vous presserez la barre d'espacement. Ainsi, CTRL-S vous servira à revoir le listage d'un long programme à l'allure qui vous convient.

Numérotation automatique des lignes

Votre Apple II pourra se charger de numéroter les lignes pour vous, en Integer Basic. Utilisez la commande AUTO pour ce faire. L'ordinateur fournira alors le prochain numéro à chaque fois que vous presserez RETURN. Il ne passera pas à la ligne suivante s'il détecte une erreur dans la ligne que vous venez de frapper, ou si vous n'avez rien frappé du tout, sauf RETURN.
Pour sortir de la séquence de numérotation automatique, frappez CTRL-X. Cette commande annulera le numéro de ligne que vient de fournir l'ordinateur. A la suite de quoi, frappez la commande MAN (pour « numérotation MANuelle »).
Pour examiner le fonctionnement de ces commandes, faites :

```
>AUTO 1000

>1000 PRINT, "COMBIEN Y A-T-IL DE YARDS
DANS UN MILE?"
>1010                    (Pressez simplement RETURN)
>1010 PRIMT 5280/3
*** SYNTAX ERR
>1010 PRINT 5280/3
>1020 \                  (Pressez CTRL-X)
>MAN

>
```

Ainsi que vous pouvez le deviner, AUTO demande que vous précisiez à quel numéro de ligne la numérotation automatique doit commencer. Vous pourriez également indiquer l'incrément entre lignes :

```
>AUTO 1000,100
>1000 PRINT "CHAIR OU POISSON?"
>1100 \                  (Pressez CTRL-X)
>MAN
```

Dans cet exemple, l'incrément entre lignes a été fixé à 100. Si vous n'indiquez rien, la valeur par défaut sera de 10, comme dans l'exemple précédent.
L'Applesoft ne dispose pas de la numérotation automatique des lignes.

SAUVEGARDE DES PROGRAMMES SUR CASSETTES

Une unité à cassettes vous donne le moyen de sauvegarder de nombreux programmes, de les ranger à l'extérieur de l'ordinateur. Ultérieurement, vous pourrez les recharger en mémoire centrale. Supposons que vous ayez le programme suivant en mémoire :


```
10 PRINT "TOTO"
20 PRINT "JE"
30 PRINT "NE CROIS PAS"
40 PRINT "QUE NOUS SOYONS"
50 PRINT "AMERICAINS"
60 END
```

Pour sauvegarder ce programme en cassette, placez une cassette dans votre unité, ré-embobinez-la, pressez à la fois RECORD et PLAY pour la mettre en état d'enregistrer, et frappez enfin au clavier la commande :

```
SAVE
```

L'Apple II émet un « bip » et commence à enregistrer le programme sur la cassette. Lorsqu'il a terminé, il émet un second « bip ». Pressez la touche STOP de votre unité à cassettes.
Maintenant, faites NEW pour effacer la mémoire de l'ordinateur, puis LIST pour vérifier qu'effectivement, elle ne contient plus rien. Pour recharger le programme en mémoire, ré-embobinez la bande, pressez PLAY et frappez la commande suivante au clavier :

```
LOAD
```

Un « bip » vous informe que votre Apple II a commencé le chargement, et un second « bip » qu'il l'a achevé. Arrêtez la lecture de la bande. Faites LIST pour vérifier que le programme est à nouveau en mémoire.
Dans le chapitre 5, nous verrons comment utiliser des disquettes, plus pratiques pour sauvegarder et recharger des programmes.

Sauvegarde de plusieurs programmes sur une cassette

Vous aurez pu remarquer que la sauvegarde d'un programme ne dure pas très longtemps. Un programme plus long demanderait davantage de temps, mais généralement, une unique bande en cassette pourra stocker plusieurs programmes Basic. Vous pourrez les ranger séquentiellement, dans l'ordre et l'un après l'autre, le premier programme, le second, le troisième, etc.
Ce sera moins commode pour les recharger en mémoire centrale, car vous devrez sélectionner le programme voulu et si ce n'est pas le premier, sauter les précédents. Une méthode consiste à faire des LOAD successifs pour recharger les programmes l'un après l'autre... jusqu'au bon ; ce processus est lent mais il marche !
Pour aller plus vite, servez-vous du compteur de votre unité à cassettes, si elle en possède un. Remettez-le à zéro après avoir rebobiné la cassette et avant de sauvegarder un programme ; exécutez cette sauvegarde, puis notez où en est le compteur : c'est le point de départ de la sauvegarde suivante. Et ainsi de suite.
Désormais, pour recharger le second programme, il vous suffira de ré-embobiner la cassette, de remettre le compteur à zéro puis, en utilisant la touche FAST-F (pour FAST FORWARD, *enroulement rapide*), d'amener le compteur à la position que vous avez notée et à laquelle début le second programme. Faites LOAD et vous rechargerez directement ce programme.

D'UN BASIC À L'AUTRE

Sur de nombreuses versions de l'Apple II, vous avez le choix entre programmer en Integer Basic ou en Applesoft. Les critères de choix vous deviendront évidents au cours de ce chapitre. Nous avons déjà vu, par exemple, que l'Integer Basic dispose de la numérotation automatique, mais pas l'Applesoft. Par ailleurs, l'Applesoft sait traiter des valeurs décimales mais pas l'Integer Basic.

Toutes les versions de l'Apple II ne disposent cependant pas des deux Basic. L'Apple II Plus standard n'a que l'Applesoft, par exemple. La procédure permettant de passer d'un Basic à l'autre dépend du modèle de machine que vous possédez, et des options que vous avez implantées dans sa carte-mère.

Avec l'option « Apple Language System », vous avez immédiatement accès à l'un ou l'autre des Basic. Si vous êtes en Integer Basic, frappez FP pour passer en Applesoft. Si vous êtes en Applesoft, frappez INT pour passer en Integer Basic.

La carte « Applesoft Firmware » vous donne également immédiatement accès à l'un ou l'autre des Basic. Elle est équipée d'un commutateur qui émerge à l'arrière de votre Apple II et qui détermine la version du Basic voulue. Vous aurez l'Integer Basic si vous mettez l'ordinateur sous tension lorsque le commutateur est vers le bas ; s'il est alors vers le haut, c'est l'Applesoft qui sera disponible. Indépendamment de la position de ce commutateur, vous pourrez frapper FP pour l'Applesoft ou INT pour l'Integer Basic si la carte Applesoft Firmware est installée dans la machine.

Avec un Apple II standard, il est facile d'obtenir l'Integer Basic. Pressez la touche RESET et frappez CTRL-B. Ce sera plus difficile d'acquérir l'Applesoft du fait qu'il réside en cassette ou disquette. Vous devrez lui commander le chargement de ce programme.

Avant d'employer l'unité à disquettes, vous devrez amorcer le système avec le système d'exploitation sur disquette, ou DOS (de « Disk Operating System ») comme on l'a décrit dans le chapitre 2. Vous aurez à répéter cette opération chaque fois que vous remettrez votre ordinateur sous tension. Voici, très rapidement, un rappel de la séquence standard de chargement du DOS à partir du moniteur et de l'Integer Basic :

```
*6      (Pressez CTRL-P, puis RETURN)


>PR#6
```

Une fois le DOS chargé, vous pourrez charger l'Applesoft à partir de votre Integer en insérant une disquette avec l'Applesoft dans son unité de lecture, puis en frappant FP. Après quelques secondes, le caractère d'appel de l'Applesoft (]) apparaîtra sur l'écran.

Vous pouvez aussi charger l'Applesoft à partir d'une cassette. La procédure complète est décrite dans le chapitre 2. En résumé, placez la cassette dans son unité, pressez sa touche PLAY, et frappez la commande LOAD à partir de l'Integer Basic. En 1,5 à 2 minutes, la référence de l'Applesoft et son caractère d'appel seront affichés.

A partir de l'Applesoft, vous reviendrez à l'Integer Basic en commandant INT. Si vous voulez ensuite repasser à l'Applesoft, utilisez la commande FP avec l'unité à disquettes, ou la commande LOAD avec l'unité à cassettes.

PROCÉDURES ÉVOLUÉES D'ÉDITION

Dans le chapitre 2, nous avons examiné comment corriger des erreurs dans une ligne que vous êtes en train de frapper, et ce, avant d'avoir actionné RETURN. Revoyons rapidement ces quelques règles :

La touche ← d'espace arrière fait reculer le curseur et efface de la mémoire (mais pas de l'écran) les caractères sur lesquels le curseur passe.
La touche → déplace le curseur vers l'avant (la droite) et recopie (« refrappe ») les caractères sur lesquels le curseur est passé.
La touche REPT, utilisée conjointement avec ← ou →, déplace rapidement le curseur dans ces deux sens.
La commande CTRL-X annule la ligne que vous êtes en train de frapper.
La séquence ESC-@ efface l'écran et ramène le curseur dans l'angle supérieur gauche.

Nous allons maintenant découvrir d'autres méthodes d'édition des lignes du programme. Elles seront particulièrement utiles lorsque vous voudrez introduire des modifications dans les lignes en mode programme (donc, numérotées).

SUPPRESSION DE LIGNES

Pour supprimer une ligne complète, frappez son numéro puis faites RETURN. En listant ensuite le programme, vous constaterez que la ligne et son numéro ont bien disparu. En voici un exemple :

```
>NEW

>100 PRINT "LA VERTU EST SA PROPRE RECOM
     PENSE"

>110 PRINT "SI LA CHAUSSURE VOUS VA, POR
     TEZ-LA"

>120 PRINT "IL N'Y A PAS DE FUMEE SANS F
     EU"

>130 PRINT "C'EST A BOIRE, A BOIRE, A BO
     IRE"

>140 END

>110

>130

>LIST

  100 PRINT "LA VERTU EST SA PROPRE REC
      OMPENSE"
  120 PRINT "IL N'Y A PAS DE FUMEE SANS
      FEU"
  140 END
```

Vous pourriez aussi employer la commande DEL (pour « Delete »), pour effacer tout un bloc ; à partir du programme complet, faites :

```
>DEL 110, 130

>LIST

  100 PRINT "LA VERTU EST SA PROPRE REC
      OMPENSE"
  140 END
```

La commande DEL 110, 130 supprime toutes les lignes à partir de celle numérotée 110 jusqu'à celle numérotée 130, même si l'une d'entre elles, la 110 par exemple, manque.

AJOUT DE LIGNES

Vous pouvez ajouter de nouvelles lignes, dans n'importe quel ordre et à tout moment, dans votre programme. Le numéro de ligne déterminera leur emplacement. Apple II les introduira automatiquement en bonne place dans le programme déjà en mémoire. Essayez de ré-introduire la ligne 110 que vous aviez supprimée :

```
>110 PRINT "SI LA CHAUSSURE VOUS VA, POR
     TEZ-LA"

>LIST

  100 PRINT "LA VERTU EST SA PROPRE REC
      OMPENSE"
  110 PRINT "SI LA CHAUSSURE VOUS VA, P
      ORTEZ-LA"
  140 END
```

MODIFICATIONS DANS LES LIGNES DU PROGRAMME

La façon la plus simple de modifier une ligne consiste à la refrapper. Mais cette formule n'est guère satisfaisante pour plusieurs raisons. Une nouvelle frappe prend du temps, et accroît les risques d'erreurs de frappe. Par bonheur, il existe une façon de modifier les lignes du programme déjà en mémoire sans les refrapper complètement. L'Integer Basic et l'Applesoft font que ce qui est affiché sur l'écran est *actif*. Vous pouvez éditer n'importe quoi sur l'écran. En utilisant la touche ESC avec d'autres conjointement, vous pouvez déplacer le curseur à volonté sur l'écran. Vous pourrez ainsi le positionner au début de n'importe quelle ligne affichée. A ce moment, vous utiliserez → pour recopier ce qui doit rester inchangé dans la ligne. Vous pourrez remplacer, insérer, ou supprimer des caractères n'importe où dans cette même ligne.
Voici le mode d'emploi. Tout d'abord, frappez la commande LIST pour afficher la, ou les lignes que vous souhaitez modifier. Vous aurez peut-être remarqué que cette commande LIST incorpore des espaces supplémentaires en affichant le listage (pour rendre le programme plus lisible) ; ces espaces additionnels peuvent contrarier l'édition de programmes longs. Pour interdire à LIST d'introduire de tels espaces, effacez l'écran (avec ESC-@) et frappez la mystérieuse commande suivante :

```
POKE 33,33
```


Outre qu'elle supprime les espaces additionnels, cette commande réduit la largeur de l'affichage à 33 caractères au lieu de 40. Nous traiterons de cette instruction POKE plus en détail dans le chapitre 4. Pour revenir à l'affichage normal, frappez :

```
POKE 33,40
```

Pour déplacer le curseur

Pour déplacer le curseur sur l'écran, vous devez presser deux touches en séquence, d'abord ESC, puis et au choix, A, B, C ou D. Vous devez refaire cette séquence chaque fois que vous voudrez déplacer le curseur d'une unique position, à droite, gauche, haut ou bas. La figure 3.1 illustre les quatre séquences et les mouvements résultants.
Avec le moniteur Autostart, vous pourrez employer les lettres I, J, K, M, toujours conjointement avec ESC, pour déplacer le curseur. En fonction de la position de ces lettres sur le clavier, (fig. 3.2), le mouvement résultant sera vers la gauche (J), la droite (K), le haut (I), le bas (M).

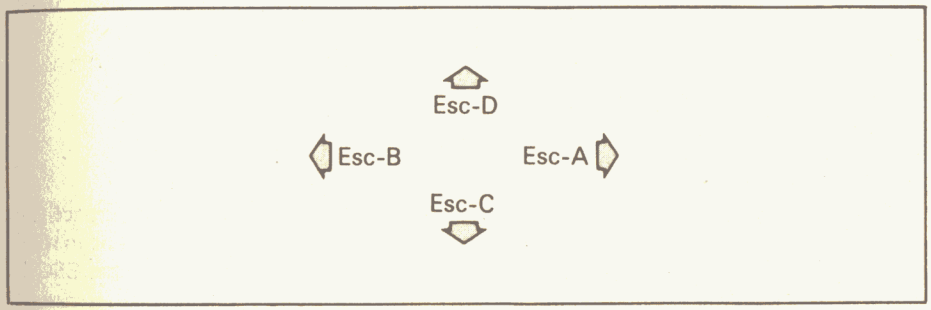


Fig. 3.1. — Les mouvements du curseur (séquence de deux touches).

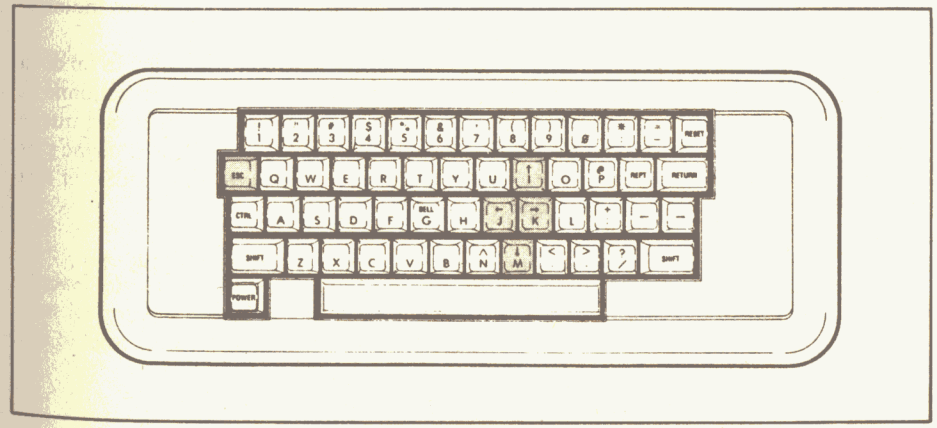


Fig. 3.2. — Les mouvements du curseur (avec le moniteur Autostart).

Il existe une différence importante dans la façon dont ESC agit avec I, J, K et M. Pressez ESC : l'Apple II entre dans le *mode éditeur*. Actionnez maintenant I, J, K ou M pour promener le curseur sur l'écran : vous n'avez plus besoin de frapper ESC à chaque fois. L'Apple II restera en mode éditeur tant que vous utiliserez les touches I, J, K, M ou REPT ;

cela vous permet de déplacer le curseur autant que vous le voulez sans avoir à refrapper ESC à chaque déplacement d'une position. Pour sortir du mode éditeur, frappez sur n'importe quelle touche, sauf I, J, K, M, REPT, CTRL ou SHIFT.
Vous pouvez recourir à REPT, dans le mode éditeur, pour déplacer rapidement le curseur sur de longues distances sans frappes superflues.

Remplacement de caractères

Remplacer un caractère par un autre est la simplicité même. Positionnez simplement le curseur sur le caractère à remplacer, et frappez le nouveau caractère immédiatement. Par exemple, si le curseur se trouve comme indiqué ici :

```
100 PRINT "HEURE DE DEPART"
```

vous pourrez remplacer DE DÉPART par D'ARRIVÉE, en frappant ce texte, et vous obtiendrez :

```
100 PRINT "HEURE D'ARRIVEE"
```

Pressez RETURN pour enregistrer la modification.

Suppression de caractères

Vous pouvez supprimer des caractères individuels en les surchargeant avec un espace (un « blanc »). Rappelez-vous qu'en Basic, les espaces supplémentaires ne jouent aucun rôle, sauf s'ils sont inclus dans des guillemets. Vous pourrez aussi employer les touches ESC et A (la touche K en mode éditeur) pour déplacer le curseur vers l'avant. A la différence de la touche →, ESC-A et ESC-K ne recopient pas les caractères sur lesquels ils passent. Si les caractères que vous voulez supprimer sont à l'intérieur de guillemets, il sera plus facile d'employer ESC-A, ou la touche J en mode éditeur, pour sauter les caractères indésirables.
Pour supprimer tous les caractères à partir de la position du curseur et jusqu'à la fin de la ligne *affichée,* pressez ESC, puis E. L'effet sera le même que si vous aviez actionné la barre d'espacement de multiples fois, jusqu'à arriver à la fin de la ligne, mais sans que le curseur se déplace. Les caractères de la ligne suivante ne seront pas supprimés, même s'ils appartiennent à la même ligne du programme. Par exemple, si vous pressez ESC et E à partir de la position suivante du curseur :

```
100 PRINT "L'OISIVETE EST  LA MERE DE TO
US LES VICES"
```

vous obtiendrez :

```
100 PRINT "L'OISIVETE EST
US LES VICES"
```

Attention ! Si vous pressez maintenant RETURN, la ligne 100 se terminera là où se trouve le curseur.
Vous pouvez aussi effacer l'écran à partir de la position du curseur et jusqu'au bas en pressant ESC et F.

L'insertion de caractères

L'insertion de caractères dans une ligne est simple. Elle pourra vous troubler, au début, car le résultat final n'apparaît pas aussitôt. L'Apple II ne peut pas décaler les caractères sur une ligne pour faire place à d'autres caractères. La figure 3.3 montre comment l'Apple II vous laissera insérer des caractères. Vous frappez le texte au-dessus de la ligne avec l'aide du curseur (et de ses mouvements : ESC, etc.). Vous devrez vous souvenir que ce qui est affiché sur l'écran n'est pas la réplique exacte de ce qui sera rangé en mémoire. Voici un exemple d'édition faisant appel à ce processus d'édition. On va employer l'Integer Basic (>), mais cela se passerait de la même façon en Applesoft. Examinez ce qui suit :

```
>NEW

>10 PRINT "SUR LE COTE"
>
```

Pour insérer le mot BAS- avant côté, listez d'abord la ligne du programme :

```
>LIST
   10 PRINT "SUR LE COTE"

>
```

En utilisant les mouvements du curseur avec ESC, etc., et *uniquement ainsi,* positionnez le curseur de façon qu'il se superpose au premier caractère de la ligne, ainsi :

```
>LIST
   10 PRINT "SUR LE COTE"

>
```

Employez maintenant la touche → pour recopier la première partie de la ligne ; arrêtez le curseur sur le C de côté :

```
>LIST
   10 PRINT "SUR LE COTE"
```

Pressez ESC, puis D pour faire passer le curseur à la ligne supérieure. (Si des caractères sont ici présents à la droite du curseur, supprimez-les en faisant ESC, puis E, pour dégager la ligne.) Vous obtenez :

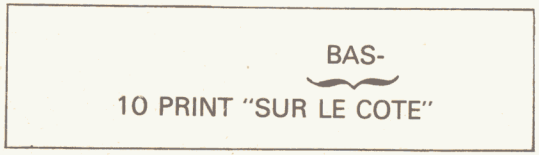


Fig. 3.3. — L'insertion de caractères.

```
>LIST
   10 PRINT "SUR LE COTE"

>
```

Frappez alors le mot BAS-, ce qui donne :

```
>LIST                BAS-
   10 PRINT "SUR LE COTE"

>
```

En utilisant *exclusivement* les touches de mouvement du curseur, placer celui-ci sur la première lettre de l'insertion. *Ne vous servez pas* de la touche ← car, bien qu'il semble que le résultat soit le même qu'avec ESC-B ou ESC-J sur l'écran, la touche ← efface les caractères sur lesquels elle passe ; elle annulerait vos insertions ! Vous obtenez :

```
>LIST                BAS-
   10 PRINT "SUR LE COTE"

>
```

Pressez ESC, puis C, pour ramener le curseur sur la ligne d'origine ;

```
>LIST                BAS-
   10 PRINT "SUR LE COTE"

>
```

Finalement, utilisez la touche → pour recopier le reste de la ligne, puis pressez RETURN. Une nouvelle commande LIST vous donnera :

```
>LIST                BAS-
   10 PRINT "SUR LE COTE"

>LIST 10
   10 PRINT "SUR LE BAS-COTE"

>
```

L'appendice B contient une liste des commandes d'édition.

RÉ-EXÉCUTION EN MODE IMMÉDIAT

Le fait qu'avec l'Apple II, tout ce qui apparaît sur l'écran est actif, vivant, vous permet de ré-exécuter toute instruction en mode immédiat encore affichée. Vous pourrez la ré-exécuter en mode immédiat telle qu'elle se présente, ou encore l'éditer d'abord.
Dans les deux cas, la première chose à faire consiste à positionner le curseur au début de la ligne en mode immédiat. Employez ESC et A, B, C et D en séquence comme décrit antérieurement (ou alors, ESC suivi de I, J, K, et M si vous disposez d'un moniteur Autostart). Puis, frappez → pour recopier l'instruction en mode immédiat. Vous pourrez, bien sûr, introduire des modifications dans la ligne en appliquant les techniques décrites pour remplacer, supprimer, ou insérer des caractères.

Pour voir comment cela fonctionne, examinez le programme suivant en mode immédiat qui calcule un volume en mètres cubes ;

```
>PRINT "VOLUME EN M3= ";10*25*8
VOLUME EN M3 = 2000
```

Pour calculer un volume différent, par exemple de $10 \times 25 \times 14$, positionnez d'abord le curseur au début de la ligne en mode immédiat (en pressant alternativement ESC et D trois fois). Maintenant, actionnez → et REPT ; le curseur va se déplacer sur la ligne jusqu'à arriver sur 8 ; vous relâchez alors les deux touches. Si vous n'êtes pas assez rapide et dépassez l'objectif, revenez en arrière avec ←, ou repartez de l'avant avec →. Vous auriez d'ailleurs aussi bien pu emmener le curseur sur le 8 en frappant 33 fois sur → sans employer REPT.
Lorsque le curseur est sur le 8, frappez la nouvelle dimension, 14, puis pressez RETURN :

```
>PRINT "VOLUME EN M3= ";10*25*14
VOLUME EN M3 = 3500
```

LANGAGES DE PROGRAMMATION

Un langage de programmation est un moyen de communication entre vous et l'ordinateur. Il existe de nombreux langages de programmation. Certains, comme le Basic, sont d'usage général alors que d'autres ont été conçus pour des données spécifiques telles que la gestion, le scientifique, le graphique, le traitement de texte, etc. Les langages de programmation sont aussi différents que les langues parlées. Outre le Basic, les langages les plus courants sont Pascal, C, Fortran, Cobol, APL, PL/M, PL1 et Forth.
Les machines Apple II peuvent employer plusieurs languages, Basic et Pascal, entre autres. Ce livre est consacré à la programmation en Basic.
Quel que soit le langage, chaque instruction doit être rédigée en respectant un jeu de règles parfaitement définies. Regroupées, ces règles s'appellent la *syntaxe*. Chaque language de programmation dispose de sa propre syntaxe.
Les langages de programmation, tout comme les langues parlées, possèdent des *dialectes* qui se manifestent par des variantes mineures de la syntaxe. Apple II dispose de deux dialectes en Basic : l'Integer Basic et l'Applesoft. En raison de leurs différences, des programmes écrits dans l'un ne tourneront souvent pas sur une machine attendant des ordres dans l'autre. De plus, un programme Basic rédigé pour l'Apple II ne fonctionnera probablement pas sur un autre ordinateur, même s'il se réclame aussi du Basic. Cependant, une fois que vous aurez appris à programmer votre Apple II dans l'un de ses dialectes Basic, vous n'éprouverez guère de difficultés à passer à un autre.
Certaines règles de syntaxe sont évidentes. Les exemples d'addition et de soustraction donnés au début de ce chapitre le prouvent. Il n'est nullement indispensable d'être un programmeur pour les comprendre. Mais la plupart des règles de syntaxe sont hautement arbitraires et sans signification, tant que vous n'avez pas étudié cette syntaxe. Ne recherchez donc pas d'explications rationnelles à la syntaxe ; généralement, il n'y en a pas. Par exemple, pourquoi adopter le symbole * pour la multiplication ? Normalement, vous

écririez × pour le signe de la multiplication et pour représenter la lettre x. Cependant, presque tous les langages informatiques ont adopté l'astérisque (*) comme symbole de la multiplication. La division est universellement représentée par une barre oblique (/) ; il n'y a aucune raison particulière à ce choix, n'importe quel autre caractère aurait pu convenir.

ÉLÉMENTS DE BASIC

La plupart des règles de syntaxe, en Basic, concernent des instructions individuelles. Elles traitent séparément des trois groupes principaux d'éléments : numérotation des lignes, instructions à l'ordinateur, et données. Nous allons les décrire chacune à leur tour. Il existe aussi un petit nombre de règles qui appartiennent au programme, considéré comme un tout. Nous en traiterons en temps utile dans ce chapitre.

REVUE DE LA NUMÉROTATION DES LIGNES

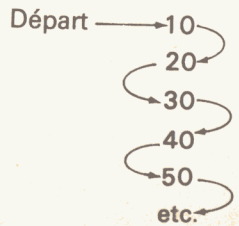
Nous avons déjà abordé le thème de la numérotation des lignes. Après un bref rappel, nous allons entrer dans plus de détails. En mode programmé, chaque ligne d'un programme Basic doit disposer d'un numéro qui lui soit propre et unique. Ces numéros déterminent l'ordre d'exécution des instructions ; la ligne avec le numéro le plus petit est exécutée la première, et celle avec le numéro le plus grand, la dernière.
L'Integer Basic permet l'emploi de numéros allant de un à 5 chiffres, avec des valeurs entières comprises entre 0 et 32767.
L'Applesoft autorise des numéros de un à cinq chiffres également, mais allant de 0 à 63999.

Les numéros en tant qu'adresses

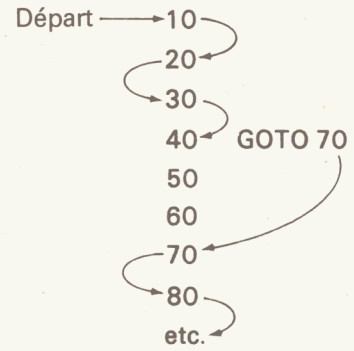
Par essence, les numéros de lignes constituent un moyen d'adresser les lignes du programme. Ce concept est important, car chaque programme comprend deux types d'instructions :

1. Les instructions qui créent ou modifient les données ;
2. Celles qui contrôlent la séquence, l'ordre dans lequel les opérations sont exécutées.

L'idée que les opérations spécifiées par un programme doivent être exécutées en respectant une séquence bien définie est un concept simple. Normalement, l'exécution du programme commence avec la première instruction, et se poursuit séquentiellement comme illustré ci-dessous :



Cependant, nous allons vite découvrir que la plupart des programmes incluent des exécutions non séquentielles. C'est pourquoi cette numérotation des lignes est très importante ; vous pourrez, en effet, identifier alors une ligne par son numéro pour modifier la séquence d'exécution. On peut illustrer ce processus ainsi :



ESPACES

De manière générale, vous pouvez introduire des espaces supplémentaires (des « blancs ») librement afin d'améliorer la lisibilité de vos programmes. Mais parce que ces espaces consommeraient de la mémoire, Apple II compacte ceux qui ne sont pas indispensables hors de la ligne lorsque vous pressez RETURN. Lorsque vous listez ensuite le programme (commande LIST), Apple II les ré-introduit pour la lisibilité en fonction d'un plan pré-déterminé. Souvenez-vous que vous pouvez supprimer cette ré-introduction en frappant la commande POKE 33,33 avant d'émettre la commande LIST. L'ordre POKE 33,40 normalise l'affichage sur l'écran.
Utilisez avec prudence des espaces supplémentaires inclus dans des guillemets. Comparez les deux commandes suivantes :

```
PRINT "DONNEZ LA DATE"
DONNEZ LA DATE

PR INT "   DONNEZ LA D ATE"
       DONNEZ LA D ATE
```

DONNÉES

Le travail principal d'un ordinateur consiste à introduire, traiter, et extraire des données. Aussi, la façon dont le langage de programmation manipule les données est très importante, qu'il s'agisse de textes ou de chiffres. Nous allons maintenant décrire les types de données que vous rencontrerez dans les programmes d'Apple II.

Chaînes

Une *chaîne* est constituée par un caractère, ou n'importe quelle séquence de caractères introduite dans des guillemets. Nous avons déjà employé des chaînes avec l'ordre PRINT, et elles représentaient des messages à afficher.

En voici d'autres exemples :

```
"L'IGNORANCE EST COUPABLE"
"TOTAUX = 4019 - 181 - 324 - 837"
"LOUIS DURAND"
"AU GRAND CAFE"
"LE 1ER AVRIL 1982"
```

A quelques exceptions près, une chaîne peut contenir n'importe quel caractère disponible sur le clavier, utilisant les touches alphabétiques et numériques courantes, avec ou sans CTRL ou SHIFT. Ces exceptions sont constituées par ←, →, RETURN, ESC, CTRL-H, CTRL-M, CTRL-U et CTRL-X, qui, soit déplacent le curseur, soit terminent la ligne que vous frappez, ou les deux.
Une chaîne peut atteindre n'importe quelle longueur, de 0 à 255 caractères. Une chaîne sans caractère est appelée une chaîne *vide* (ou nulle).
Il existe quelques caractères invisibles que vous pouvez obtenir avec des combinaisons de touches. Par exemple, si vous frappez simultanément CTRL et G, l'ordinateur émet un « bip ». Vous pourrez placer ce caractère dans une chaîne. Essayez d'employer ces caractères dans un ordre PRINT :

```
PRINT ""  (Pressez CTRL-G plusieurs fois dans les guillemets)
```

Ici, vous entendez le « caractère » dans le voir. D'autres caractères sont à la fois invisibles et inaudibles ; ils serviront à commander les fonctions de l'imprimante, des dispositifs de communication, et des autres matériels que vous connecterez à votre Apple II.
L'appendice I liste le jeu entier des caractères de l'Apple II et vous indique quelle touche produit quoi.
Avec l'Applesoft, vous disposez d'une méthode vous permettant d'inclure, dans une chaîne, des caractères que vous ne pouvez pas générer directement avec les touches. Vous utiliserez, pour cela, la fonction CHR$, que nous décrirons dans le chapitre 4.

Nombres

Deux types de nombres peuvent être stockés dans l'Apple II : les *entiers,* nombres sans partie fractionnaire, et les *nombres réels* (appelés aussi nombres en virgule flottante) qui peuvent disposer d'une partie fractionnaire. Comme vous pourriez le supposer, l'Integer Basic ne traite que les entiers (« Integer », en anglais, signifie « entier »). L'Applesoft s'adresse aux entiers et aux réels.
Vous devez exprimer les nombres avec un point pour séparer la partie décimale (et non une virgule), et ne jamais employer de virgule ; ne séparez jamais, non plus, les tranches de trois chiffres avec une virgule ou un point. Par exemple, écrivez 65432.1 et non 65.432,1 ou 65432,1 ou...

Entiers

Un nombre entier ne comporte pas de partie fractionnaire ni donc de point décimal. Il peut être négatif (−) ou positif (+). Un nombre non signé est supposé positif. La gamme des entiers va de −32767 à 32767.

Les nombres suivants sont des entiers :

```
0
1
44
32699
-15
```

Nombres réels

Un nombre réel est un nombre complet, pouvant contenir une partie décimale, ou constitué uniquement de cette partie décimale. Il peut être négatif (−) ou positif (+). En l'absence de signe, il est réputé positif. Le plus petit (le plus négatif) des nombres réels est :

```
-1000000000000000000000000000000000000000
```

Et le plus grand est :

```
1000000000000000000000000000000000000000
```

Voici quelques exemples de nombres réels :

```
5
-15
65000
161
0
0.5
0.0165432
-0.0000009
1.6
24.0055
-64.2
```

Lorsqu'un nombre décimal approche du zéro de moins de :

```
.00000000000000000000000000000000000000003,
```

il est converti en zéro.

Notation scientifique

Des nombres très grands ou très petits seront représentés, par l'Applesoft, en *notation scientifique.* Chaque nombre de plus de neuf chiffres avant le point décimal sera représenté en notation scientifique. Chaque nombre fractionnaire plus près du zéro que ±.01 sera exprimé en notation scientifique.

Un nombre en notation scientifique à la forme :

nombre E ± *ee*

où : *nombre* est un entier, une fraction, ou la combinaison des deux, comme illustré ci-dessus. La section *nombre* contient les chiffres significatifs et est appelée *coefficient*. S'il n'y a pas de point décimal, on suppose que ce point se trouve à la droite du nombre.

E est toujours la lettre E, qui signifie *exposant*.

± est un signe optionnel, plus, ou moins.

ee est un exposant sur un ou deux chiffres. Il spécifie la valeur du nombre, c'est-à-dire de combien de places à droite (exposant positif) ou à gauche (exposant négatif) il faut déplacer le point décimal pour lui attribuer sa vraie position.

Voici quelques exemples de notation scientifique :

Notation standard	Notation scientifique
1000000000	1E + 09
.000000001	1E – 09
200	2E + 02
–123456789	–1.23456789E + 09
–.00000123456789	–1.23456789E – 06

Ainsi que vous pouvez le constater, la notation scientifique est un moyen commode pour exprimer de très grands ou de très petits nombres. Les valeurs maximales et minimales de nombres réels, qu'on écrivait avec de nombreux zéros, peuvent être ramenés de 1E + 38 à 1E – 38, ce qui est bien plus compact. De même, le nombre le plus proche de zéro qu'on puisse obtenir est 3E-38.

Arrondi

Nous avons déjà indiqué dans ce chapitre que les nombres réels peuvent posséder jusqu'à neuf chiffres de précision. Pour un nombre supérieur à 1, ou inférieur à – 1, cela signifie simplement que les neufs digits les plus à gauche peuvent être autres que des zéros. L'Apple II supprimera les digits excédents et arrondira le nombre, pour le ramener à neuf chiffres. En voici quelques exemples (vous remarquerez que les grands nombres sont en notation scientifique) :

```
]PRINT 1234567891
1.23456789E+09

]?-123456789123456789
-1.23456789E+17

]?-150000475.75
-150000476

]?90000000.7558
90000000.8
```

Les nombres fractionnaires (compris entre 1 et – 1) sont soumis aux mêmes limitations.

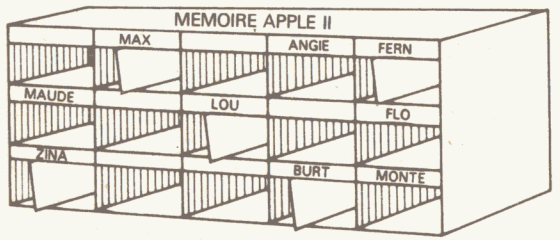
Dans ce cas, cependant, les neuf chiffres de précision commencent avec le premier digit, non à zéro à la droite du point décimal. En voici quelques exemples :

```
]PRINT .1234567891
.123456789

]?-123456789123456789
-1.23456789E+17

]?-123456789 123456789
-1.23456789E+17

]?.000000000900000007558
9.00000008E-10
```

VARIABLES

Jusqu'à présent, notre discussion sur les données a porté sur ce que l'on considère comme des valeurs constantes. Or, il est souvent pratique de se référer aux données en les appelant par leur nom plutôt qu'en donnant leur valeur. C'est pourquoi elles sont appelées des *variables*.

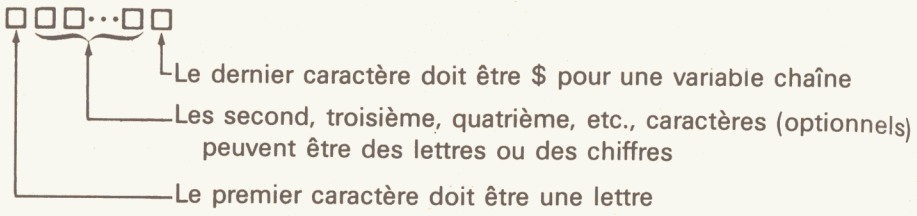
Si vous avez étudié l'algèbre élémentaire, vous n'aurez aucune peine à assimiler ce concept de variables, et le nom de variable. Dans le cas contraire, supposez que le nom d'une variable est un nom attribué à une boîte aux lettres. Ce que vous mettrez dans cette boîte aux lettres deviendra la valeur associée au nom de cette boîte, et cela, jusqu'à ce que vous y mettiez autre chose qui se substituera à la précédente valeur. Dans le jargon informatique, on dit qu'on attribue une valeur à la variable.



Une variable ne se voit pas toujours affecter la même valeur. C'est ce qui fait sa puissance : elle peut représenter n'importe quelle valeur autorisée. Vous pourrez changer cette valeur au cours du déroulement du programme. Basic dispose d'un certain nombre d'instructions pour cela ; nous les examinerons ultérieurement.

Noms de variables en Integer Basic

Avec l'Integer Basic, les noms de variables peuvent comprendre de 1 à 100 caractères. Les règles générales sont les suivantes :



Ainsi, le dernier caractère du nom de la variable précise à l'Integer Basic le type de la variable, chaîne ou numérique.
Les *variables chaînes* sont des variables dont la chaîne va de 0 à 255 caractères. Les espaces de la chaîne comptent pour des caractères. Avant d'utiliser une variable chaîne, vous devrez spécifier sa longueur maximale. Pour cela, vous emploierez la commande DIM (pour « dimension »), que nous décrirons plus loin. Si vous l'omettez, vous recevrez un message d'erreur ***STR OVFL ERR. Voici quelques exemples de noms de variables chaînes, légaux et illégaux (interdits) :

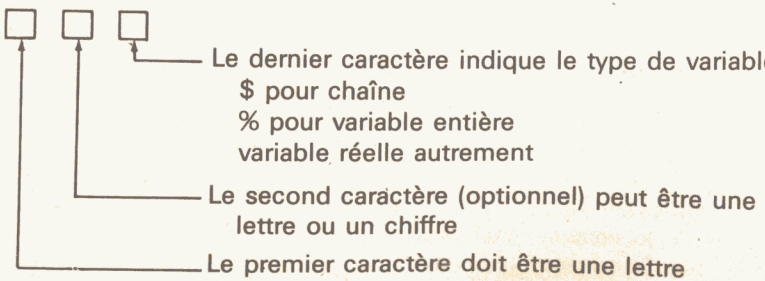
Légal	Illégal
A$	$
NOMCLIENT$	9$
PART1$	DECLAR.VOL$
DEPOT$	
X8$	

Les variables numériques, en Integer Basic, doivent se voir affecter des valeurs comprises entre −32767 et +32767. Si vous excédez ces limites, vous obtiendrez le message d'erreur ***>32767 ERR. Voici quelques exemples de noms de variables numériques en Integer Basic :

Légal	Illégal
A	QU'EST-CE
CODE PTT	3X4Z
X0	$TOTAL

Noms de variables en Applesoft

Un nom de variable, en Applesoft, peut avoir un, deux ou trois caractères, avec les règles suivantes :



Ainsi, le dernier caractère du nom de la variable indique le type de donnée qu'elle représente.
En Applesoft, une *variable chaîne* peut stocker une chaîne allant de 0 à 255 caractères. Vous n'avez plus besoin d'en préciser la longueur maximale, comme en Integer Basic ; Applesoft s'en passe. Voici quelques exemples de noms de variables, autorisés et interdits :

Légal	Illégal
A$	0$
MN$	M!$
F6$	77$

Les *variables entières* se réfèrent à des nombres compris entre −32767 et 32767. Si vous dépassez ces limites, vous recevrez le message d'erreur ?ILLEGAL QUANTITY ERROR. Si vous tentez de ranger une valeur réelle comme une variables entière, Applesoft convertira la valeur réelle en une valeur entière avant de l'accepter. Nous allons examiner rapidement les règles de conversion. Voici quelques exemples de noms de variables entières en Applesoft :

Légal	Illégal
A%	A$%
B%	31%
A1%	3D%
X4%	

Les *variables réelles* se réfèrent à des valeurs généralement restreintes à la gamme -10^{38} à $+10^{38}$, bien que vous puissiez exécuter des calculs sur des valeurs aussi élevées que $1{,}7 \times 10^{38}$, en valeur absolue (donc, en − ou +) sous certaines conditions. Si vous tentez de ranger dans une variable réelle une valeur excédant cette gamme, vous obtiendrez le message d'erreur ?OVERFLOW ERROR.
Lorsque la valeur d'une variable réelle se rapproche du zéro à moins de $\pm 2.9388 \times 10^{-39}$, Applesoft la ramène à zéro.
Rappelez-vous que les variables réelles peuvent recevoir des valeurs entières, puisqu'un nombre entier n'est autre chose qu'un nombre fractionnaire avec la partie décimale à zéro. Voici quelques exemples de noms de variables réelles :

Légal	Illégal
A	0
B	7B
A1	A#
AA	
Z5	

Noms de variables plus longs en Applesoft

Les noms de variables peuvent disposer de plus de deux caractères alphanumériques, auxquels s'ajoutent les suffixes % ou $ pour des entiers ou des chaînes, en Applesoft ; mais seuls, les deux premiers caractères comptent réellement. Ainsi PRIX1 et PRIX2 sont un

seul nom pour l'Applesoft, puisqu'ils commencent tous deux par PR. Cependant, PRIX1 et PRIX1% sont différents car leur suffixe est différent.
L'Applesoft autorise des noms de variables jusqu'à 238 caractères. Voici quelques exemples de noms de variables à plus de deux caractères :

Légal	Illégal
COMPTEUR %	ITEM= %
BALANCE	2IEME
NOMS$	CLIENT, ADRESSE$

Si vous employez des noms de variables avec plus de deux caractères, souvenez-vous que :

1. Seuls, les deux premiers caractères et le suffixe ($ ou%) sont significatifs. N'utilisez pas des noms tels que BOUCLE1% et BOUCLE2% qui se réfèrent à la même variable BO.
2. Les caractères additionnels demandent eux aussi de la mémoire, que vous pourriez peut-être mieux utiliser pour vos programmes. Leur avantage, c'est qu'ils rendent les programmes plus faciles à lire. PRODUITNO est ainsi plus aisé à comprendre que PN, par exemple, s'il constitue un nom de variable décrivant le numéro d'un produit au cours d'un inventaire.

Mots réservés

Tous les mots servant aux instructions Basic sont appelés des *mots réservés.* L'appendice F en donne la liste pour l'Integer Basic et l'Applesoft. Vous en avez déjà rencontré plusieurs dans ce chapitre, les autres seront décrits ailleurs.
Lorsqu'il exécute des programmes en Basic, Apple II scrute tous les ordres, recherchant si une chaîne ne constitue pas un mot réservé. La seule exception réside dans les chaînes incluses dans des guillemets. Si un mot réservé est compris dans la chaîne, dans un nom de variable, cela pourra constituer une sérieuse gêne car Apple II n'est pas suffisamment intelligent pour identifier un nom de variable par sa position dans l'instruction. De ce fait, *vous devrez soigneusement vous garder d'employer des mots réservés dans les noms de variables ;* ce sera particulièrement important avec les mots réservés courts, qui peuvent aisément se glisser dans un nom de variable plus long.

TABLEAUX

On appelle *tableau,* un grand nombre de variables organisées de façon systématique. On utilisera fréquemment des tableaux dans les programmes informatiques. Si vous ne comprenez pas de qu'est un tableau, ou si vous ne savez pas comment l'exploiter, poursuivez votre lecture car ce qui suit est très important.
Par conception, les tableaux sont des choses très simples. Lorsque vous disposez de deux objets (items) ou davantage, au lieu de leur attribuer un nom différent de variable, vous donnerez à cette « collection » d'objets un unique nom général. Cette collection est appelée un *tableau,* son nom est un *nom de tableau.* Les objets, ou items, individuels sont souvent appelés les *éléments du tableau* et ils sont numérotés. Vous choisirez un élément quelconque en le désignant par le numéro indiquant sa position ; ce numéro est appelé un *index.*
Les tableaux constituent un moyen plus court pour décrire un grand nombre de variables possédant un lien quelconque. Considérez, par exemple, une table de 200 numéros. Comment voudriez-vous assigner un nom de variable à chacun de ces 200 nombres ? Il sera

plus simple de les introduire dans un tableau qui se verra affecter d'un nom global, puis d'identifier chacun de ces nombres par leur position dans la table. C'est précisément ce que fait le tableau.
Considérez par exemple, comment un motel de dix chambres peut suivre leur occupation. On pourrait attribuer un nom de variable à chaque chambre :

JONES	SMITH	DOE		LITTKE	ALTON	DAVIS	HANSON	SHORTEN	
R1$	R2$	R3$	R4$	R5$	R6$	R7$	R8$	R9$	R10$

Ou alors, on peut considérer les clients du motel comme appartenant à un tableau :

JONES	SMITH	DOE		LITTKE	ALTON	DAVIS	HANSON	SHORTEN	
R$(1)	R$(2)	R$(3)	R$(4)	R$(5)	R$(6)	R$(7)	R$(8)	R$(9)	R$(10)

Dans cet exemple, R$ est le nom du tableau. Celui-ci se compose de dix éléments, dont chacun est le nom du client. Un index (donné entre parenthèses) suit chaque nom de la variable. Ainsi, un élément quelconque du tableau (le nom de l'occupant de la chambre) est identifié par le nom de la variable et par l'index. L'occupant de la chambre 3 est la valeur de R$(3), soit DOE.
L'exemple de tableau ci-dessus s'appuie sur des chaînes, car les chaînes sont aisément, et même intuitivement comprises par beaucoup. En réalité, l'Integer Basic ne permet que des tableaux numériques.
En Applesoft, les tableaux peuvent représenter des variables entières, des variables réelles, ou des variables chaînes ; cependant, une simple variable tableau ne peut représenter qu'un seul type de donnée. Autrement dit, une unique variable ne peut mêler des entiers et des nombres réels ; mais bien sûr, une variable réelle pourra toujours se voir affecter une valeur entière (mais non l'inverse). Chaque type de tableau occupe un volume mémoire différent ; consultez l'appendice G pour plus de détails.

Dimension(s) d'un tableau

Vous devez spécifier le nombre d'éléments d'un tableau avant de l'exploiter, en Integer Basic. Vous le ferez avec l'instruction DIM *(dimension).* On l'étudiera plus loin dans ce chapitre.
Applesoft vous laisse utiliser des tableaux avec dix éléments maximum sans que vous ayez à les dimensionner.
En Applesoft, les tableaux peuvent disposer de plus d'une dimension, ce qui signifie qu'il faudra plus d'un index pour en sélectionner un élément. Un tableau à une seule dimension est équivalent à une liste, à une seule rangée de nombres. L'index identifie un nombre dans une unique rangée. Un tableau à deux dimensions mène à une table ordinaire de nombres, avec des rangées et des colonnes ; un index identifie la rangée, un autre la colonne. Vous pourriez visualiser un tableau à trois dimensions sous forme d'un cube, ou encore d'un empilement de tables à deux dimensions. Des tableaux à quatre dimensions, ou davantage, sont difficiles à visualiser, mais ne sont pas mathématiquement plus complexes qu'un tableau à nombre moindre de dimensions.
On peut créer un exemple de tableau à deux dimensions en développant l'exemple précédent du motel avec sa liste de clients. Supposons que l'hôtel soit de huit étages de dix chambres chacun. On dispose de quatre options pour suivre les occupations de chambres avec les noms des 80 clients. D'abord, chaque chambre peut se voir affecter un nom de

variable. Deuxièmement, l'hôtel peut être représenté par un tableau à 80 éléments. Troisièmement, chaque étage peut se voir affecter un tableau à dix éléments. Quatrièmement, l'hôtel peut être représenté par un tableau à deux dimensions. Illustrons ainsi ce dernier choix :

H$(8,1)	H$(8,2)	H$(8,3)	H$(8,4)	H$(8,5)	H$(8,6)	H$(8,7)	H$(8,8)	H$(8,9)	H$(8,10)
H$(7,1)	H$(7,2)	H$(7,3)	H$(7,4)	H$(7,5)	H$(7,6)	H$(7,7)	H$(7,8)	H$(7,9)	H$(7,10)
H$(6,1)	H$(6,2)	H$(6,3)	H$(6,4)	H$(6,5)	H$(6,6)	H$(6,7)	H$(6,8)	H$(6,9)	H$(6,10)
H$(5,1)	H$(5,2)	H$(5,3)	H$(5,4)	H$(5,5)	H$(5,6)	H$(5,7)	H$(5,8)	H$(5,9)	H$(5,10)
H$(4,1)	H$(4,2)	H$(4,3)	H$(4,4)	H$(4,5)	H$(4,6)	H$(4,7)	H$(4,8)	H$(4,9)	H$(4,10)
H$(3,1)	H$(3,2)	H$(3,3)	H$(3,4)	H$(3,5)	H$(3,6)	H$(3,7)	H$(3,8)	H$(3,9)	H$(3,10)
H$(2,1)	H$(2,2)	H$(2,3)	H$(2,4)	H$(2,5)	H$(2,6)	H$(2,7)	H$(2,8)	H$(2,9)	H$(2,10)
H$(1,1)	H$(1,2)	H$(1,3)	H$(1,4)	H$(1,5)	H$(1,6)	H$(1,7)	H$(1,8)	H$(1,9)	H$(1,10)

Comme vous pouvez le constater, le premier index de ce tableau à deux dimensions est l'étage et le second index, le numéro de chambre sur cet étage. Ainsi, R$(3,2) serait le nom de l'occupant de la seconde chambre du troisième étage.
Les tableaux peuvent atteindre 88 dimensions en Applesoft. Il n'y a pas de limites spécifiques sur le nombre d'éléments dans chaque dimensions. Le volume mémoire disponible limitera le nombre total des éléments, naturellement, puisque chacun de ceux-ci occupe un certain espace en mémoire.

EXPRESSIONS

Dans la section qui suit, nous allons explorer les méthodes permettant de combiner les valeurs de variables et de constantes en utilisant des *expressions*. Nous avons déjà employé des expressions pour calculer la valeur résultant de problèmes arithmétiques simples, en mode immédiat. Rappelez-vous l'instruction :

```
PRINT 4+6
10
```

Elle ordonne à l'Apple II d'ajouter 4 à 6 et d'afficher la somme. L'instruction :

```
PRINT A+B
0
```

commande à l'ordinateur d'ajouter les valeurs de deux variables numériques, A et B, puis d'afficher leur somme.
Le signe plus (+) est celui de l'addition. En jargon informatique, on dit que c'est un *opérateur*. Le signe plus est ainsi en *opérateur arithmétique* car il spécifie une addition, qui est une opération arithmétique.
Les opérateurs arithmétiques sont suffisamment faciles à comprendre ; nous avons tous appris à additionner, soustraire, multiplier et diviser dans notre jeune âge. Mais il existe d'autres types d'opérateurs : *opérateurs de chaînes, opérateurs relationnels,* et *opérateurs booléens*. Ils sont aussi faciles à comprendre, mais parce qu'ils impliquent des notions

plus abstraites, il faut leur consacrer davantage d'explications.
Chaque catégorie d'opérateurs définit un type d'expression. Il existe des expressions arithmétiques, de chaînes, relationnelles et booléennes.

Préséance entre opérateurs dans les expressions

Les expressions peuvent faire appel à plusieurs opérateurs. Par exemple :

```
PRINT A+B/10
0
```

demande à la fois une addition et une division. Il existe un schéma standard servant à déterminer dans quel ordre évaluer une telle expression. Nous allons établir les règles de *préséance* pour chaque type d'expression, en commençant par la concaténation de chaînes, et en poursuivant par les expressions entières, réelles, relationnelles et booléennes, puis en mélangeant tout cela dans le même ordre. Pour commencer, examinons comment se débarrasser des règles standard de préséance.

Pour éviter les règles standard de préséance

Vous pouvez modifier l'ordre dans lequel Apple II évalue les expressions en usant de parenthèses. Chaque opération comprise dans les parenthèses est exécutée en premier lieu. Lorsque plusieurs jeux de parenthèses co-existent, l'Apple II les évalue de la gauche vers la droite.
Introduire une seconde paire de parenthèses dans une première s'appelle l'*emboîtage*. Dans ce cas, Apple II évalue d'abord le contenu de la paire de parenthèses la plus interne, puis celle qui suit immédiatement (et qui est devenue, à son tour, la plus interne), etc. Les parenthèses peuvent être emboîtées jusqu'à n'importe quel niveau. Utilisez-les librement pour clarifier l'ordre des opérations à exécuter dans une expression.
Voici quelques exemples d'expressions avec des parenthèses, en mode immédiat :

```
PRINT (2+10)*3
36

>
PRINT ((2+10)*3+31)*10
670

>
PRINT -(2^(3+8/4))
-32
```

Concaténation de chaînes

Vous pouvez relier des chaînes en les associant bout à bout pour créer une chaîne plus longue. C'est ce qu'on appelle la *concaténation*. Elle peut se visualiser ainsi :

CHAINE 1 = CHAINE 2 = CHAINE 3

Devient

CHAINE 1 CHAINE 2 CHAINE 3

En concaténant des chaînes, on peut en créer de nouvelles avec un maximum de 255 caractères.
L'Integer Basic ne dispose pas d'opérateur de concaténation. Vous pourrez alors concaténer des chaînes en Integer Basic à l'aide d'une technique exposée à la fin de ce chapitre (sa présence, ici, aurait été prématurée).
L'Applesoft recourt au signe plus (+) comme opérateur de concaténation. Voici quelques exemples de concaténation de chaînes en Applesoft ;

"SUR"+"BOUM"	*donne* "SURBOUM"
"REPORT"+" "+"MENSUEL"	*donne* "REPORT MENSUEL"
"TELE"+R$	*donne* les caractères TELE suivis par la valeur de R$
A1$+YA$+C$(1)	*donne* la valeur de A1$, suivie par la valeur de YA$, suivie par la valeur de C$(1)

Expressions entières

Les expressions entières sont des expressions arithmétiques n'impliquant que des variables entières et des constantes entières. Nous traiterons des expressions arithmétiques incluant à la fois des valeurs entières et réelles dans le paragraphe intitulé « Expressions mixtes ». Les opérateurs pour expressions entières sont l'addition (+), la soustraction (−), la multiplication (*), la division (/) et l'exponentiation, ou élévation à une puissance ( ). Vous pourrez aussi employer le moins unaire (−) pour indiquer une valeur numérique négative. Les opérations sont exécutées dans l'ordre suivant : le moins unaire d'abord, suivi par l'exponentiation, puis la multiplication et la division, et finalement l'addition et la soustraction. Les opérations de même préséance sont exécutées de gauche à droite.
Voici quelques exemples d'expressions entières en Integer Basic :

100 − 30*2	donne	40
−9∧2	donne	81
A/B*C	donne	la valeur A divisée par B, et l'entier du résultat multiplié par C
D+X*3	donne	trois fois la valeur de X, plus D
5/2*2	donne	4 (le quotient de 5/2 est converti en entier 2)

Voici quelques exemples d'expressions entières en Applesoft :

−120/2+100	donne	40
2∧3*2	donne	16
N1%*N2%/N3%	donne	le produit des valeurs de N1% par N2%, le tout divisé par N3%
AA%/AB%/AC%	donne	la valeur de AA% divisée par celle de AB%, puis ce quotient divisé par AC%
5/2*2	donne	5 (le quotient de 5/2 n'est pas converti en entier)

L'Integer Basic dispose d'un opérateur supplémentaire que vous pourrez employer dans les expressions entières. Il fournit le reste d'une division, lorsque le dividende n'est pas exactement divisible par le diviseur. Cet opérateur est MOD (pour « Modulo »). Il a le même niveau de préséance que la multiplication et la division. Voici quelques exemples de MOD :

4 MOD 3	donne	1
3*5 MOD 4	donne	3
(41+2)/25 MODA	donne	le reste de la division de 18 par A
3 MOD 4	donne	3

Expressions réelles

L'Applesoft dispose d'une autre forme d'expressions arithmétiques, avec des valeurs réelles. Ses opérateurs sont les mêmes : addition (+), soustraction (−), multiplication (*), division (/), exponentiation (∧) et moins unaire (−). La préséance des opérations est la même : moins unaire d'abord, puis exponentiation, multiplication et division, et finalement addition et soustraction. Voici quelques exemples d'expressions réelles.

87.5−4.25*2	donne	79
1.51∧(3/2/2)	donne	1.35540301
AL*(PL−3.1*CB)	donne	la valeur de AL fois le résultat de la soustraction du produit de 3.1 par CB de PL.
7.5*2/5	donne	3

Expressions de relations

Les opérateurs de relations comparent deux valeurs afin de déterminer leurs relations, l'une avec l'autre. Vous pouvez vérifier si la première est plus grande, plus petite, égale, différente, plus grande ou égale, plus petite ou égale à la seconde. Les comparaisons peuvent porter sur des constantes, des variables, ou toutes sortes d'expressions (avec quelques restrictions en Integer Basic). Si la valeur d'un nombre de la comparaison est une chaîne, celle du second nombre doit être une chaîne également. Autrement dit, vous pouvez comparer des types semblables de valeurs en utilisant des opérateurs de relations.
Si la relation est vraie, l'expression relationnelle a pour valeur numérique 1. Si elle est fausse, le résultat est 0.
Les opérateurs de relations en Applesoft et Integer Basic sont les mêmes, à une exception près, comme le montre le tableau 3.1 :

Integer Basic	Opération	Applesoft
<	Plus petit que (*)	<
>	Plus grand que (*)	>
=	Egal à	=
#	Différent de	<> ou ><
>=	Plus grand ou égal à *	>= ou =<
<=	Plus petit ou égal à *	<= ou <

(*) — Non admis avec des chaînes en Integer Basic

Tableau 3.1. — Opérateurs relationnels.

Tous les opérateurs relationnels ont la même préséance ; ils sont évalués dans l'ordre, de gauche à droite. Voici quelques exemples d'expressions relationnelles :

1=5-4	donne	1 (vrai)
14>66	donne	0 (faux)
15>=15	donne	1 (vrai)
"AA">"AA"	donne	0 (faux)
"ANDERSON" > "ASTON"	donne	1 (vrai)
(A=B)=(A$>B$)	dépend	de la valeur des variables. Si A est égal à B et si la valeur de A$ est supérieure à B$, alors l'expression donne 1 (vrai)

Le concept de relation est aisé à comprendre. Les valeurs arbitraires 0 et 1, en Basic, attribuées aux conditions fausse et vraie, peuvent servir dans des expressions entières ou réelles. C'est moins facile à comprendre, car il s'agit là de quelque chose d'arbitraire. Par exemple, quelle signification donner à l'expression (1=1)*4? Hormis en Basic, cette expression n'a pas de sens ; mais en Basic, (1=1) est vrai et donne 1, et ainsi, l'expression est la même que 1*4, qui donne 4. Vous pourrez incorporer des expressions relationnelles dans d'autres expressions Basic. En voici des exemples :

25+(14>66)	revient à faire	25+0
(A+(1=5-4))*(15>=15)	revient à faire	(A+1)*(1)

Comparaison de chaînes

Vous pouvez vous demander quelles règles l'Apple II applique pour comparer des chaînes. Deux considérations sont à prendre en compte. Tout d'abord, la longueur de la chaîne. Des chaînes d'inégale longueur (rappelez-vous que les espaces comptent) ne sont pas égales. Si une chaîne plus courte est identique au début d'une chaîne plus longue, la chaîne plus longue est plus grande que la plus courte (ce qui n'est vrai qu'en Applesoft). La seconde considération est : les chaînes contiennent-elles les mêmes caractères et dans le même ordre ? En Integer Basic, vous pouvez uniquement comparer deux chaînes avec les opérateurs = et #. Les chaînes sont comparées un caractère après l'autre, à partir de

la gauche — le premier caractère de la première chaîne avec le premier caractère de la seconde, le second caractère de l'une avec le second caractère de l'autre, le troisième au troisième, et ainsi de suite jusqu'à épuisement de l'une des chaînes ou jusqu'à ce qu'une différence de caractères se manifeste.
L'Applesoft compare l'ordre relatif des caractères, un par un. Pour cette comparaison, les lettres de l'alphabet donnent, dans l'ordre : A<B, B<C, C<D, etc. Les nombres apparaissant dans des chaînes donnent 0<1, 1<2, 2<3, etc. Les autres caractères, tels que +, −, $, etc., sont arbitrairement classés comme le montre l'appendice I.

Expressions booléennes

Les opérateurs booléens fournissent à un programme l'aptitude de prendre des décisions logiques. C'est pourquoi on les appelle souvent des *opérateurs logiques*. Il existe quatre opérateurs booléens standard : AND (ET logique), OR (OU logique), OR exclusif (OU exclusif) et NOT (inversion). Le Basic d'Apple II n'en reconnaît que trois : AND, OR et NOT.
Si vous ne comprenez pas le rôle des opérateurs booléens, une simple analogie avec un supermarché va illustrer cette logique. Supposons que vous achetiez des fruits pour vos deux enfants ; l'opérateur AND dira que vous achèterez des bananes pour tous les deux si c'est ce qu'ils préfèrent ; l'opérateur OR, que vous prendrez des bananes pour l'un d'entre eux si un seul enfant les aime. L'opérateur NOT inverse la donnée : si l'enfant A est toujours en désaccord avec l'enfant B, les décisions de l'enfant B seront toujours l'inverse de celles de l'enfant A.
Les ordinateurs ne fonctionnent pas sur des analogies mais sur des nombres. Aussi, les opérateurs booléens réduisent-ils les valeurs sur lesquelles ils travaillent à des 1 ou des 0 (vrai ou faux). Puisqu'ils traitent des 0 et des 1, ils seront le plus souvent utilisés dans des expressions relationnelles (rappelez-vous qu'elles fournissent un 0 ou un 1). Les opérateurs booléens fonctionnent aussi avec d'autres types d'opérandes, comme nous allons le voir aussitôt après.

L'opération AND donne un 1 seulement si les deux valeurs sont à 1.
(C'est notre ET logique.)

1 AND 1=1	1 AND 0=0
0 AND 1=0	0 AND 0=0

L'opération OR donne 1 si au moins l'une des valeurs est à 1.
(C'est notre OU logique.)

1 OR 1=1	1 OR 0=1
0 OR 1=1	0 OR 0=0

L'opération NOT complémente logiquement chaque valeur.
(C'est notre « inversion ».)

NOT 1=0
NOT 0=1

Tableau 3.2. — Table des opérations booléennes.

Le tableau 3.2 résume la façon dont les expressions booléennes sont évaluées. Un tel tableau est appelé *table de vérité*. La préséance est la même pour tous les opérateurs booléens. Si plusieurs opérateurs booléens sont présents dans la même expression, ils sont évalués de la gauche vers la droite. En voici quelques exemples :

```
NOT ((3+4)>=6)                          donne    0 (faux)
("AA"="AB") OR ((8*2)=4^2)              donne    1 (vrai)
NOT ("POMME "="ORANGE") AND (A$=B$)
                                        donne    1 si A$ et B$ sont égaux,
                                                 0 (faux) autrement
```

Expressions mixtes

Très souvent, les expressions n'incluront pas des valeurs d'un seul type. C'est particulièrement vrai en Applesoft, puisqu'il dispose des types numériques réels et des entiers. Nous avons déjà évoqué la possibilité de mixer des expressions dans notre discussion sur les expressions relationnelles et booléennes. Vous pouvez mêler librement les types dans toute expression, avec une exception : les chaînes ne peuvent pas participer à des expressions entières réelles ou booléennes. Les chaînes ne seront présentes que dans les expressions de chaînes et relationnelles. Voici quelques exemples d'expressions mixtes :

Légal	Illégal
3.1416 * (R ^ 2)	1600 + "PENNSYLVANIA AVENUE"
A% >= B/3	ST$ <A%
43 AND 137	A$ AND B$
1 OR 4E + 10	NOT (A$) = B$
(A$ = B$) AND −6.25	NOT(A = B) OR C$

Apple II doit résoudre plusieurs choses quand il évalue une expression mixte. La première concerne la préséance des opérateurs. La table 3.3 donne la liste des opérateurs pour tous les types d'expressions dans l'ordre de leur préséance, de la plus haute à la plus basse. Elle montre que tout ce qui est entre parenthèses est évalué en premier. Si plus d'un niveau de parenthèses existe, Apple II commence par évaluer le contenu des parenthèses les plus internes, puis de proche en proche, jusqu'aux plus externes. (Rappelez-vous que nous avons déjà traité de ce concept et de l'*emboîtage*.) Puis, les expressions arithmétiques sont évaluées. Après quoi, ce sont les expressions relationnelles et enfin, les expressions booléennes.

Ainsi que nous l'avons noté antérieurement, les expressions relationnelles retournent une valeur de 0 ou de 1 selon que la relation testée est fausse ou vraie. Ainsi, une expression relationnelle peut être introduite dans une expression entière ou réelle.

Vous pouvez mêler les expressions booléennes. Tout ce qu'elles contiennent sera, avant toute opération, converti en 0 et 1. Les valeurs numériques sont converties selon les règles suivantes : si la valeur est 0, elle reste à zéro ; toute valeur autre que zéro est convertie en 1.

Le Basic ne peut pas convertir automatiquement des chaînes en valeurs numériques, aussi sont-elles interdites dans des expressions entières, réelles et booléennes, sauf si elles font partie d'une expression relationnelle.

En Applesoft, les valeurs entières et réelles peuvent être toutes deux présentes dans la même expression réelle, relationnelle ou booléenne.

	Préséance	Integer Basic	Applesoft	Signification
	Haut 9	( )	( )	Les parenthèses déterminent l'ordre de l'évaluation
Opérateurs arithmétiques	8	^	^	Exponentiation
	7	−	−	Moins unaire
	6	*	*	Multiplication
	6	/	/	Division
	6	MOD	Non disponible	Reste de la division (**)
	5	+	+	Addition
	5	−	−	Soustraction
Opérateurs relationnels	4	=	=	Egal
	4	#	<> ou ><	Non égal
	4	<	<	Plus petit que
	4	>	>	Plus grand que
	4	<=	<= ou =<	Plus petit ou égal
	4	>=	>= ou =>	Plus grand ou égal
Opérateurs booléens	3	NOT	NOT	Complément logique
	2	AND	AND	ET logique
	1	OR	OR	OU logique
	Bas			
(**) En Integer Basic uniquement				

Tableau 3.3. — Les opérateurs.

Lorsqu'elles interviennent dans une expression réelle, les valeurs entières sont temporairement converties en valeurs réelles afin que l'expression soit évaluée. Le résultat final pourra être soit une valeur entière, soit une valeur réelle, tout dépendant du contexte dans lequel se place l'expression. Applesoft convertira automatiquement la valeur comme il convient.

Les valeurs réelles sont converties en entiers en abandonnant leur partie fractionnaire et en les alignant sur le nombre suivant le plus bas ; ils sont *tronqués*. Par exemple :

```
 1.1 devient   1
 1.9 devient   1
-1.1 devient  -2
-1.9 devient  -2
```

INSTRUCTIONS BASIC

Nous sommes maintenant prêts à décrire les instructions Basic, ce que l'on appelle aussi des ordres, ou des commandes. En toute rigueur, une *commande* est une instruction émise en mode immédiat.

Chaque instruction exécute une tâche spécifique. Ce chapitre va vous présenter les concepts de la programmation, montrant la façon dont les instructions interviennent. Nous ne décrirons pas les instructions en détail dans ce chapitre. Vous lirez leur description

complète dans le chapitre 8 et vous comprendrez ainsi comment ces instructions fonctionnent. Ce chapitre ne vous fournira qu'une vue partielle des possibilités des instructions. Une dernière note avant de commencer. Bien que ce chapitre vous présente les concepts de la programmation, il ne peut couvrir toute la programmation en profondeur. Si vous souhaitez employer davantage d'instructions en programmant, consultez la bibliographie donnée dans l'appendice K.

REMARQUE

Il est nécessaire de commencer cette discussion des instructions Basic par la description de la seule instruction Basic que l'ordinateur ignorera : la remarque. Si les trois premiers caractères d'une instruction sont les trois premières lettres de remarque, soit REM, l'ordinateur ignorera totalement l'instruction. Alors, pourquoi l'inclure dans un programme ? Tout simplement parce que les REM rendront votre programme plus facile à lire.
Si vous rédigez un programme court, de cinq ou dix instructions, vous n'aurez certainement aucun mal à vous souvenir de ce que le programme fait — à moins que vous ne l'abandonniez pour six mois puis y reveniez pour l'utiliser à nouveau. Si vous rédigez un programme de 100 ou 200 instructions, il est probable que vous aurez oublié quelque chose d'important le concernant lorsque vous voudrez vous en servir une prochaine fois. Dès que vous aurez écrit des douzaines de programmes, vous n'aurez plus aucune chance de vous rappeler ce qu'ils font en détail. La solution consiste alors à documenter ces programmes en leur incorporant des remarques décrivant ce qu'ils sont censés faire.
Les bons programmeurs utilisent de nombreuses remarques émaillant tous les leurs programmes. Dans les exemples de ce chapitre, nous allons inclure des remarques décrivant ce qui se passe, simplement afin que vous preniez l'habitude d'en faire autant.
Les instructions de remarques se voient affecter un numéro de ligne. Ce numéro peut être utilisé comme n'importe quel numéro d'instruction.

INSTRUCTIONS D'AFFECTATION

Les instructions d'affectation servent à attribuer des valeurs aux variables. Vous les rencontrerez fréquemment, dans chaque type de programme en Basic. Voici un exemple d'instruction d'affectation :

```
90 REM INITIALISATION DE LA VARIABLE X
100 LET X=3
```

A la ligne 100, la variable X se voit attribuer la valeur 3, ce qu'on pourrait d'ailleurs ré-écrire ainsi :

```
100 X=3
```

Le mot LET (correspondant à : SOIT, en français) est optionnel et peut être omis.
Voici une instruction d'affectation avec une chaîne :

```
215 A$= "COURS AUSSI"
```

La chaîne A$ se voit attribuer les deux mots COURS AUSSI.

Voici quelques autres instructions d'affectation attribuant des valeurs à des variables en tableau R$( ), que nous avons présentés en décrivant les tableaux :

```
200 REM R$( ) EST LA LISTE DES CLIENTS DU MOTEL
210 R$(1) = "JEAN"
220 R$(2) = "PIERRE"
230 R$(3) = "DENISE"
```

Rappelez-vous qu'on peut loger plusieurs instructions sur la même ligne : de ce fait, les trois affectations R$ peuvent être regroupées :

```
200 REM R$( ) EST LA LISTE DES CLIENTS DU MOTEL
210 R$(1) = "JEAN" : R$(2) = "PIERRE" : R$(3) = "DENISE"
```

Souvenez-vous qu'un double point doit séparer des instructions adjacentes apparaissant sur une même ligne.
Les instructions d'affectations peuvent inclure les opérateurs arithmétiques ou logiques décrits précédemment dans ce chapitre. En voici des exemples :

```
90 REM ORDRE D'AFFECTATION AVEC OPERATEURS ARITHMETIQUES
100 V = 33 + 7/9
```

Ici, on vient d'affecter la valeur 4.17647059 à la variable réelle V ; cette instruction est équivalente aux trois suivantes :

```
90 REM X ET Y DOIVENT ETRE INITIALISEES
   SEPAREMENT POUR USAGE ULTERIEUR
100 X=7
110 Y=9
120 V=33 + X/Y
```

Ce que l'on pourrait encore écrire :

```
100 X=7:Y=9:V=33+X/Y
```

Voici des affectations avec opérateurs booléens :

```
90 REM CES EXEMPLES ONT ETE PRESENTES
   ANTERIEUREMENT DANS CE CHAPITRE
100 A=NOT ((3+4)>=6)
110 B=("AA"="AB") OR ((8*2)=(4 ^ 2))
```

L'exemple suivant montre comment une variable chaîne pourrait se voir assigner une valeur par concaténation en Applesoft :

```
90 REM R$(6) SE VOIT AFFECTER M. DURAND
100 MR$ = "M."
110 MS$ = "MME"
120 N$ = "DURAND"
200 R$(6) = MR$ + N$
```

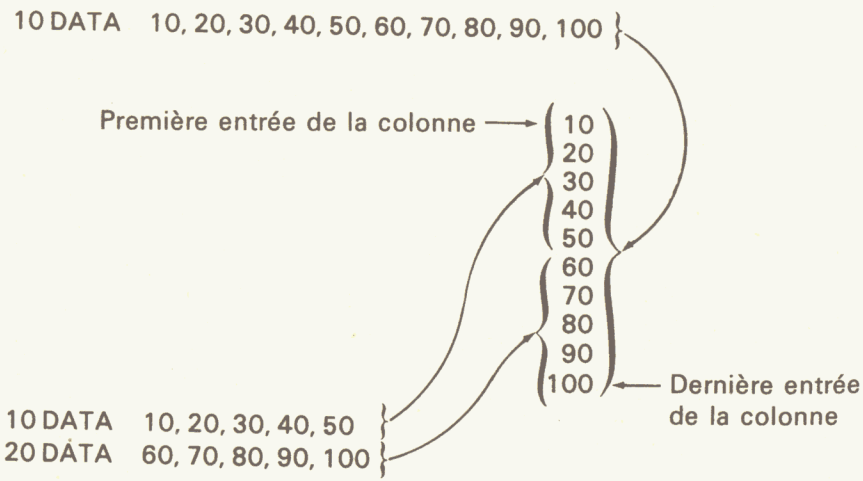
Instructions DATA et READ

Lorsque de nombreuses variables doivent se voir affecter des valeurs dans un programme en Applesoft, vous pouvez employer les instructions DATA et READ, plutôt que la méthode d'affectation ci-dessus. READ signifie « lire » et DATA « donnée ». Par exemple :

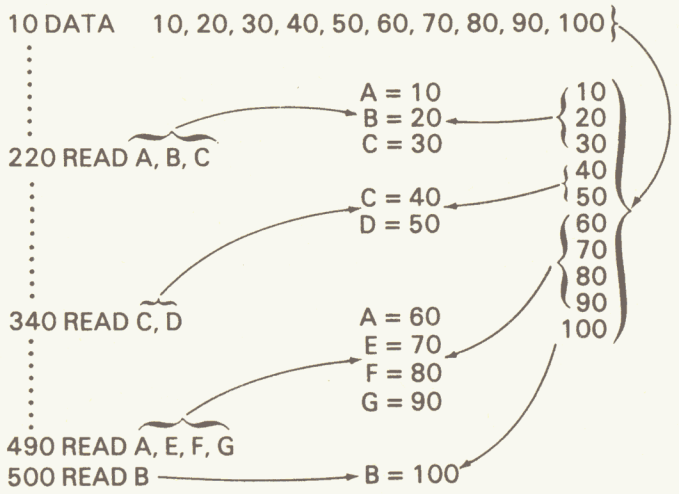
```
 5 REM INITIALISATION DE TOUTES LES VARIABLES
10 DATA 10, 20,-4, 300
20 READ A,B,C,D
```

La ligne 10 spécifie quatre valeurs numériques. Celles-ci se voient affecter aux quatre variables de la ligne 20. Après exécution de ce programme, A = 10, B = 20, C = −4 et D = 300.

Si vous disposez d'une ou de plusieurs instructions DATA dans votre programme, vous pourrez les visualiser en bâtissant une colonne de valeurs. Par exemple, un ordre DATA contenant une liste de dix valeurs mènera à une colonne à dix entrées. Deux ordres DATA spécifiant chacun cinq valeurs aboutiront au même résultat exactement. Ce que l'on peut illustrer ainsi :



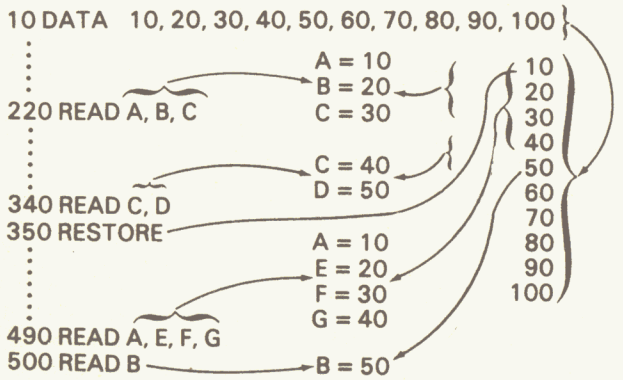
La première instruction READ du programme démarre avec la première entrée de la colonne, puis prélève séquentiellement les valeurs en les affectant à ses variables. La seconde instruction READ du programme prélèvera les valeurs à la suite dans la colonne, à partir du point où le READ précédent l'avait abandonnée. On peut illustrer ainsi ce processus :



La colonne DATA peut contenir à la fois des valeurs numériques et des chaînes. Lorsque vous attribuerez des valeurs à des variables en exploitant l'ordre READ, chaque variable devra être du même type (chaîne ou numérique) que la valeur qui lui est affectée.

L'instruction RESTORE

A tout moment, on peut ramener le pointeur au début de la colonne DATA en exécutant un ordre RESTORE (pour « restaurer », remettre) en Applesoft. Voici ce qui se passe :



Mise à zéro de variables

L'Integer Basic et l'Applesoft vous permettent de remettre à zéro toutes les variables numériques, tous les éléments d'un tableau et toutes les variables chaînes, même en tableau, d'un seul coup.

C'est l'instruction de remise à zéro, ou « clear » en anglais, qui le permet. En Integer Basic, elle se note CLR et vous pouvez l'employer en mode immédiat :

```
>X=37

>PRINT X
37

>CLR

>PRINT X
0
```

L'ordre CLEAR en fait autant en Applesoft. Il remet aussi au départ le pointeur de Data dans sa colonne, comme le ferait RESTORE. En voici un exemple :

```
]10 REM INITIALISATION DE VARIABLES

]20 X=37

]30 A$="CHEMIN DE FER"

]40 PRINT A$

]50 CLEAR

]60 PRINT X

]RUN
CHEMIN DE FER
0
```

DÉCLARATION DES DIMENSIONS DE CHAÎNES ET TABLEAUX

Si vous envisagez l'utilisation de variables chaînes ou de tableaux dans votre programme, vous aurez à « déclarer » leurs dimensions maximales dans un ou plusieurs ordres DIM, tout au début du programme. Une instruction de dimension peut définir les dimensions de n'importe quel nombre de variables chaînes et de tableaux, pour autant que tout cela tienne sur une unique ligne.

En Integer Basic, on exécute cela en donnant le nom du tableau ou de la variable chaîne puis en indiquant sa longueur maximale (dans des parenthèses). Seul, des tableaux numériques à une seule dimension sont autorisés ; les tableaux de chaînes ou les tableaux à plusieurs dimensions sont interdits. Voici un exemple de déclaration de deux chaînes de 5 et 25 caractères respectivement, et d'un tableau de 13 éléments (de 0 à 12) :

```
10 DIM S1$(5),S2$(25),NB(12)
```

Le nombre suivant le nom de la variable chaîne est la longueur maximale que la chaîne pourra prendre dans le programme. Le nombre suivant un nom de tableau numérique est égal à la valeur de l'index, la plus grande que vous pourrez utiliser pour ce tableau.

La première fois que l'Applesoft rencontre un tableau, il vérifie si vous l'avez dimensionné. Sinon, il le dimensionne automatiquement en autorisant l'index à aller de 0 à 10 pour chaque dimension du tableau. Aussi n'avez-vous pas besoin de dimensionner un tableau s'il ne dépasse pas 11 dans chacune de ses dimensions. L'exemple suivant dimensionne le tableau R$ et le tableau d'entiers R% de 21 éléments :

```
]15 DIM R$(10),R%(20)
```

Le double dimensionnement de la liste des clients de notre motel s'écrirait :

```
]15 DIM H$(8,10)
```

Le, ou les nombres qui suivent un nom de tableau dans une instruction DIM, correspondent à la valeur d'index la plus élevée que pourront prendre les index. Rappelez-vous cependant que l'indexation commence en zéro : ainsi, R$(10) correspond à 11 valeurs de R$() et non à 10, puisqu'on pourra passer par les valeurs 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 et 10. De même, H$(8,10) spécifie un tableau à deux dimensions avec 99 entrées, puisque la première dimension peut prendre les valeurs 0, 1, 2, 3, etc. et que la seconde va de 0 à 10.

Redimensionnement des tableaux

Une fois que vous avez dimensionné un tableau, de variables, vous pouvez le redimensionner sans relancer tout le programme. Des références ultérieures ne peuvent pas recourir à un index supérieur au nombre que vous avez déclaré ; chaque index doit se voir attribuer une valeur entre 0 et le nombre précédent de dimensionnement.

INSTRUCTIONS DE BRANCHEMENT

La plus simple des instructions de branchement est GOTO (« aller en »). Elle sert à spécifier l'instruction qui sera ensuite exécutée. Considérez la séquence suivante :

```
20   A = 4
30   GOTO 100
40
50
60
70
80
90
100
110
etc.
```

L'instruction de la ligne 20 est une affectation : elle assigne la valeur 4 à la variable A. L'instruction suivante est un GOTO (qui se prononce, dans la langue de Shakespeare, « gotou ») : elle spécifie au programme de poursuivre l'exécution à la ligne 100. Ainsi, la séquence du programme est 20, 30, puis 100.

Naturellement, d'autres instructions de branchement ramèneront à la ligne 40, sans quoi celle-ci ne sera jamais exécutée si l'on s'en tient à la logique du programme ci-dessus.

Vous pouvez vous brancher à n'importe quel numéro de ligne, même si celle-ci n'est autre qu'une remarque. L'ordinateur ignorant la remarque, l'effet sera le même que si l'on était passé à la ligne suivant cette remarque. Par exemple, considérez le branchement suivant :

```
20  A = 4
30  GOTO 70
40
50
60
70  REM ICI, REMARQUE QUELCONQUE
80
90
etc.
```

A l'exécution, le programme va se brancher de la ligne 30 à la ligne 70 ; celle-ci ne comportant qu'une remarque, l'ordinateur va passer à la ligne 80, exécutant l'ordre prescrit sur celle-ci. Ainsi, bien que vous puissiez vous brancher sur une remarque, vous pouvez aussi la sauter et vous brancher à l'instruction qui la suit :

```
20  A = 4
30  GOTO 80
40
50
60
70  REM ICI, REMARQUE QUELCONQUE
80
90
etc.
```

Mais si vous essayez de vous brancher sur une ligne inexistante, vous recevrez un message d'erreur.

Instruction GOTO calculée

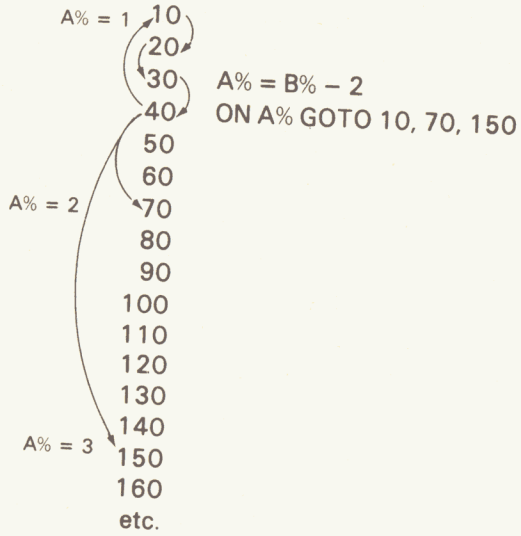
Une version du GOTO permet au programme de se brancher sur une ligne parmi deux lignes possibles, ou davantage, le branchement exécuté dépendant alors de la valeur courante de l'expression numérique.
Considérez la séquence suivante en Integer Basic :


```
          10
          20
          30   A = B - 3
          40   GOTO 30*A + 50
A = 0     50
          60
          70
A = 1     80
          90
          100
A = 2     110
          120
          etc.
```

L'ordre de la ligne 40 est un *GOTO calculé.* Lorsque cette instruction est exécutée, la logique du programme va provoquer un branchement à l'instruction dont le numéro est donné par l'évaluation de l'expression sur la même ligne. Dans cet exemple, on ira en 50 si A = 0, en 80 si A = 1, alors que A = 2 branchera en 110. Si la ligne calculée n'existe pas dans le programme, la machine vous adressera le message d'erreur ***BAD BRANCH ERR (« bad » = mauvais). Remarquez que la variable A se voit affecter une valeur à la ligne 30 ; celle-ci dépend de B. L'illustration ne montre pas comment la variable B est calculée ; cependant, et tant que B a une valeur de 3, 4 ou 5, l'instruction de la ligne 40 provoquera un branchement.
Pour tester un GOTO calculé en Integer Basic, introduisez le programme suivant :

```
>9 REM INITIALISATION DE B
>10  B = 4
>20  PRINT B
>30  A = B - 3
>40  GOTO 30 * A + 50
>49  REM  B = 3
>50  END
>79  REM  B = 4
>80  PRINT B
>90  B = 5
>100  GOTO 20
>109  REM  B = 5
>110  PRINT B
>120  B = 3
>130  GOTO 20
```

Puis, exécutez ce programme en frappant RUN.
Pouvez-vous imaginer quelle séquence de chiffres va être affichée ? Essayez de ré-écrire ce programme de façon que chaque nombre soit affiché une fois selon la séquence 345345345...
Applesoft dispose d'une version sensiblement différente du GOTO calculé :



L'instruction de la ligne 40 constitue le GOTO calculé. Lorsqu'elle est exécutée, la logique du programme commandera un branchement en 10 si la variable 1% = 1, en 70 si A% = 2, tandis que A% = 3 provoquera un branchement en 150. Si A% prend une valeur autre que 1, 2 ou 3, le programme se poursuit en 50.
Cette expression, dans un GOTO calculé en Applesoft, est évaluée et sa valeur détermine à quelle ligne le programme va se brancher. Si la valeur est 1, le premier numéro de ligne est utilisé ; si la valeur est 2, ce sera le second ; etc. Si la valeur est 0, ou excède le nombre de numéros de lignes de la liste, le programme se poursuit à la l'instruction suivant immédiatement le GOTO calculé.
Le programme suivant montre comment ce GOTO calculé fonctionne :

```
10 B% = 4
20  PRINT B%
30 A% = B% - 2
40  ON A% GOTO 10,70,150
70  PRINT B%
80 B% = 5
90  GOTO 30
150  PRINT B%
160 B% = 3
170  GOTO 20
```

BOUCLES

Le GOTO et le GOTO calculé vous servent à commander l'exécution de séquences d'instructions, selon les besoins de la logique de votre programme. Supposons que vous vouliez maintenant ré-exécuter une instruction (ou un groupe d'instructions) plusieurs fois. Par exemple, supposons qu'une variable tableau A (I) dispose de 100 éléments et que chacun d'eux doive se voir affecter une valeur allant de 0 à 99. Ecrire 100 instructions

d'affectation serait fastidieux. Il est beaucoup plus facile de ré-exécuter 100 fois la même instruction en boucle.

Instructions FOR et NEXT

On peut créer une boucle à l'aide des instructions FOR (« pour ») et NEXT (« au suivant ») comme suit :

```
10 DIM A(99)
20 FOR I=0 TO 99 STEP 1
30 A(I)=I
40 NEXT I
```

Les instructions entre FOR et NEXT vont être exécutées répétitivement. Il suffit alors d'un unique ordre placé entre FOR et NEXT ; c'est lui qui sera ré-exécuté. Une telle sorte de programme est appelée une *boucle FOR-NEXT.*
Pour que vous suiviez comment elle fonctionne, voici un programme qui affiche la valeur attribuée au tableau A() dans la boucle :

```
10 DIM A(99)
20 FOR I=0 TO 99 STEP 1
30 A(I)=I
35 PRINT A(I)
40 NEXT I
50 END
```

Lorsque vous frapperez RUN, le programme affichera 100 nombres, à partir de 0 et jusqu'à 99.
Les instructions entre FOR et NEXT sont ré-exécutées le nombre de fois spécifié par la variable index située immédiatement après FOR ; ici, l'index s'appelle I, et va de 0 à 99 par pas unitaire (STEP signifie « pas »). La variable I apparaît aussi à la ligne 30. Ainsi, la première fois que se fait l'affectation, I est égale à zéro ; l'affectation sera exécutée ainsi :

```
30 A(0)=0
```

La variable I s'accroît par pas, lequel pas est spécifié à la ligne 20 : STEP 1 (pas de 1) ; ainsi, I vaut 1 à la seconde affectation, lors de l'exécution de la ligne 30. L'affectation est effectivement devenue :

```
30 A(1)=1
```

I continue à être incrémentée du pas spécifié, jusqu'à ce que la valeur maximale 99 soit atteinte (ou dépassée).
Le pas peut être différent de 1 ; il peut être de n'importe quelle valeur entière. Modifiez le pas de la ligne 20 et portez-le à 5, puis ré-exécutez ce programme. Maintenant, l'instruction d'affectation n'est plus exécutée que 20 fois, puisqu'en incrémentant I 19 fois de 5 mène à 95 ; le 20e incrément donnerait 100, ce qui dépasse la valeur maximale 99. En conservant le pas de 5, on peut permettre à l'affectation d'être exécutée 100 fois en portant la valeur maximale de I à 500. Sauriez-vous introduire cette modification ? (N'oubliez pas de modifier aussi l'ordre DIM.)
Le pas n'est pas forcément positif. Mais si le pas est négatif, la valeur initiale de I devra être supérieure à sa valeur finale. Par exemple, si le pas est de – 1 et si nous voulons initialiser 100 éléments de A() avec des valeurs allant de 0 à 99, nous devrons ré-écrire la ligne 20 ainsi :

```
10 DIM A(99)
20 FOR I=99 TO 0 STEP -1
30 A(I)=I
35 PRINT A(I)
40 NEXT I
50 END
```

Exécutez ce programme pour tester un pas négatif.
Si le pas est de 1 (ce qui est fréquemment le cas), il n'est pas nécessaire de le spécifier. En l'absence d'une spécification de pas, Basic suppose qu'il est unitaire.
Vous pourrez aussi spécifier les valeurs intiale et finale de l'index, et le pas, à l'aide d'expressions. Evitez-le cependant, car vous compliquez inutilement le programme. Si vous deviez calculer l'une de ces valeurs, il serait plus efficace de le faire dans une instruction distincte, avant la boucle.
Vous pouvez employer des valeurs réelles pour les valeurs intiale et finale de l'index, et pour le pas, en Applesoft. Vous n'avez pas besoin de spécifier la variable dans l'instruction NEXT et en Applesoft. Mais si vous le faites, votre programme sera plus facile à lire.

Boucles emboîtées

La structure FOR-NEXT est appelée une *boucle de programme* car l'exécution tourne par FOR et NEXT. Cette boucle est très courante ; presque tous les programmes Basic que vous rédigerez en comprendront une ou plusieurs. Les boucles sont tellement utilisées qu'elles sont fréquemment introduites l'une dans l'autre, *emboîtées*. On peut placer autant d'instructions qu'on le veut entre FOR et NEX. Fréquemment, on trouvera des dizaines ou des centaines d'instructions. Et dans ces dizaines ou centaines d'instructions pourront s'introduire d'autres boucles. Le programme suivant illustre un seul niveau d'emboîtage :

```
10 DIM A(99)
20 FOR I=0 TO 99 STEP 1
30 A(I)=I
40 REM AFFICHAGE DES VALEURS DE A(I)
50 FOR J=0 TO I
60 PRINT A(J)
70 NEXT J
80 NEXT I
90 END
```

Des structures complexes de boucles apparaissent fréquemment, même dans des programmes relativement courts. En voici un exemple, montrant des instructions FOR et NEXT, mais sans instructions intermédiaires (figure A ci-contre).
La boucle la plus externe a pour index I ; elle contient trois autres boucles dont les index sont X, Y et Z. La boule X contient, à son tour, deux autres boucles avec, pour index, A et B. La boucle Y contient la boucle indexée par P. Enfin, la boucle Z, elle, n'en contient pas d'autre.
Les structures de boucles sont faciles à visualiser et à employer. Il reste cependant une erreur courante que vous devrez éviter : ne terminez pas une boucle externe avant d'avoir terminé une boucle interne. Par exemple, la structure suivante est interdite (fig. B).
Chaque programme doit disposer du même nombre de FOR et de NEXT, car chaque boucle démarre avec un FOR et se termine sur un NEXT.
Supposez, par exemple, qu'il y ait une instruction FOR mais deux NEXT. Le premier


```
50 FOR I=1 TO 10
60 FOR X=25 TO 347 STEP 3
   :
100 FOR A=9 TO 0 STEP -1
   :
140 NEXT A
200 FOR B=25 TO 100 STEP 5
   :
280 NEXT B
300 NEXT X
   :
500 FOR Y=1 TO 20 STEP 2
   :
600 FOR P=10 TO 20
   :
650 NEXT P
700 NEXT Y
   :
1000 FOR Z=1 TO 10
   :
1090 NEXT Z
   :
1200 NEXT I
   :
```

Fig. A.

```
50 FOR I=1 TO 10
60 FOR X=25 TO 347 STEP 3
   :
100 NEXT I
   :
200 NEXT X
```

Fig. B.

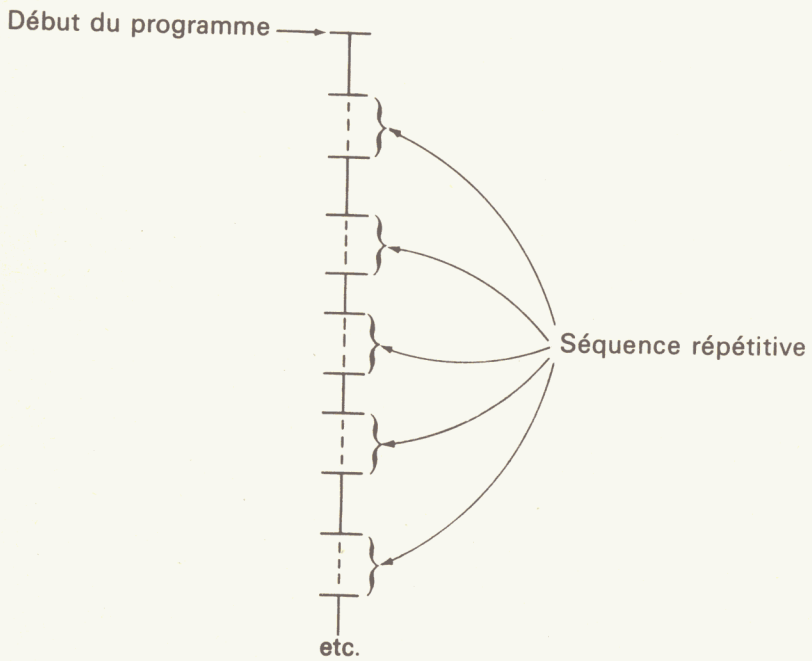
NEXT terminera l'instruction FOR de façon que la boucle s'exécute correctement. Mais le second NEXT, ne disposant pas d'un FOR, introduira une erreur.
Si vous ne précisez pas la variable (index) dans une instruction NEXT, et en Applesoft, la logique du programme terminera cependant la boucle correctement puisqu'il n'y a toujours qu'une seule possibilité de fin de boucle chaque fois qu'un NEXT est rencontré. Si

vous ne le croyez pas, examinez à nouveau l'exemple illustré précédemment ; essayez d'imaginer d'autres exemples complexes.

INSTRUCTIONS DE SOUS-PROGRAMMES

Dès que vous commencerez à rédiger des programmes dépassant plusieurs instructions, vous vous apercevrez que certaines courtes séquences reviennent continuellement. Par exemple, supposez que vous disposiez d'une variable tableau (telle que A()) qui est ré-initialisée fréquemment en divers points du programme. Allez-vous simplement répéter les trois instructions constitutives de la boucle FOR-NEXT décrite ci-dessus ? Parce qu'il ne s'agit que de trois instructions, vous pourriez le faire.

Mais supposez maintenant que cette boucle comprenne dix ou onze instructions de traitement des données dans le tableau, avant qu'il ne soit initialisé. Si vous deviez user de cette boucle plusieurs fois dans le même programme, sa ré-écriture à chaque fois pourrait vous sembler fastidieuse ; vous perdriez du temps et, de plus, vous gaspilleriez de la mémoire dans votre ordinateur. Ce concept est illustré par la figure suivante :



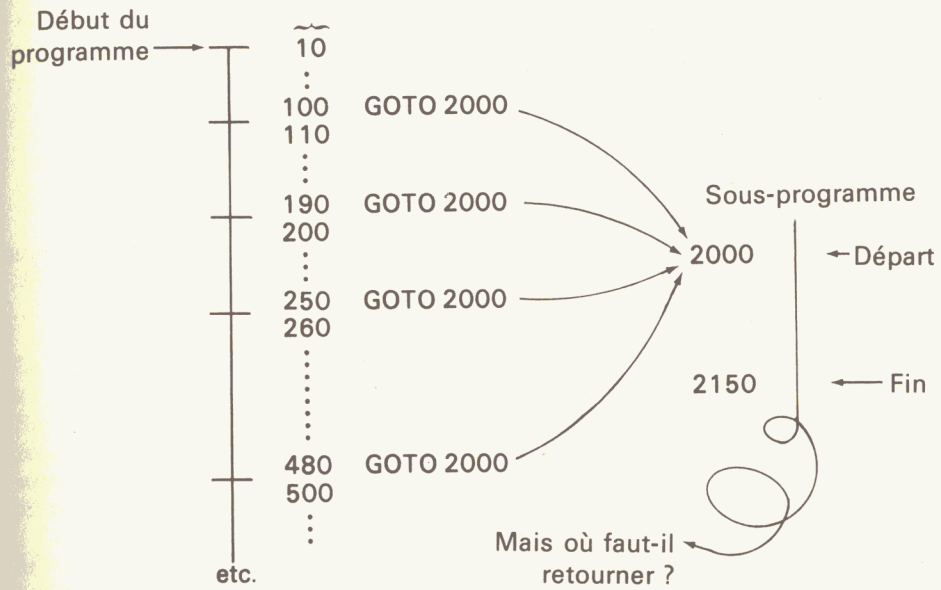
Que diriez-vous alors d'isoler la séquence répétitive et de se brancher sur elle à la demande ?

C'est précisément ce que nous allons faire ; le groupe d'instructions répétitives est alors appelé un *sous-programme.*

Mais un problème surgit. Se brancher au sous-programme à partir de votre programme est facile ; le sous-programme dispose d'un numéro de ligne pour sa première instruction. Mais à l'issue du sous-programme, comment revenir à votre programme ?

Vous pourriez exécuter un GOTO chaque fois que vous avez à vous brancher sur le sous-programme :

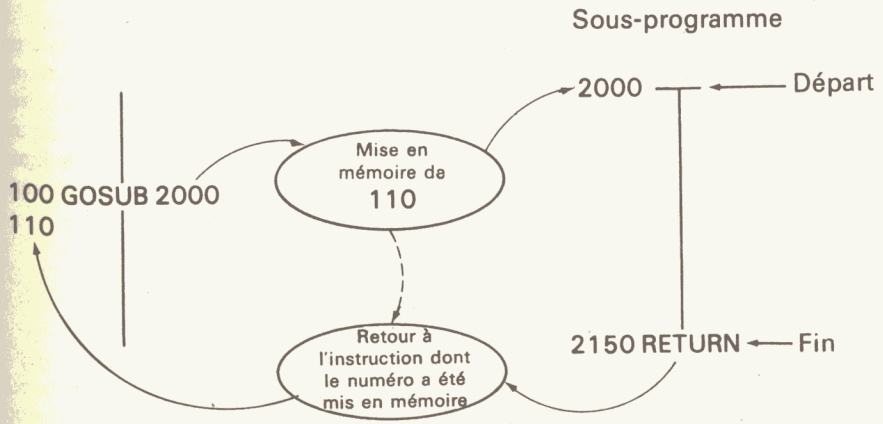
Numéros arbitraires de lignes



Mais à la fin du sous-programme, où retournerez-vous ? Si deux GOTO, ou davantage, vous branchent sur le sous-programme, il y aura deux positions possibles de retour, ou davantage, après exécution du sous-programme. La solution réside dans une instruction spécifique pour sous-programme. Au lieu d'utiliser un GOTO, on va se servir d'un GOSUB.

Instruction GOSUB

L'instruction GOSUB (abrégé de « Go to Subroutine », allez à un sous-programme) fonctionne comme un GOTO mais en plus, elle se souvient de l'instruction à laquelle il faudra faire retour. On dit, en jargon informatique, que le GOSUB *appelle* un sous-programme, ce qu'on peut illustrer ainsi :



Le sous-programme se termine par une instruction RETURN (« retour »). Elle provoque un branchement à l'instruction qui suit celle du GOSUB. Si GOSUB était la dernière instruction de la ligne, le programme se brancherait à la ligne suivante.
La boucle des trois instructions initialisant le tableau A(), si elle était convertie en sous-programme, donnerait :

```
10 REM PROGRAMME PRINCIPAL
20 REM VOUS POUVEZ DIMENSIONNER UNE VARIABLE DE
   SOUS-PROGRAMME
30 REM DANS LE PROGRAMME PRINCIPAL
40 REM C'EST D'AILLEURS UNE BONNE PRATIQUE QUE DE DIMENSIONNER
50 REM TOUTES LES VARIABLES AU DEPART DU PROGRAMME PRINCIPAL
60 DIM A(99)
70 GOSUB 2000
80 REM AFFICHER QUELQUE CHOSE POUR PROUVER QUE LE RETOUR
   S'EST FAIT
90 PRINT "RETOUR FAIT!"
100 END
2000 REM SOUS-PROGRAMME
2010 FOR I=0 TO 99
2020 A(I)=I
2030 PRINT A(I)
2040 NEXT I
2050 RETURN
```

Instruction POP

Parfois, vous ne voudrez pas que le sous-programme vous ramène à l'instruction qui suit le GOSUB. Vous pourriez être tenté d'introduire un GOTO pour le retour, mais c'est dangereux car le Basic va continuer à se souvenir de l'adresse à laquelle le sous-programme doit retourner. Dans ce cas, usez de l'instruction POP, sinon, vous risquerez des erreurs dues aux instructions RETURN inutilisées. Tout ce que fait POP, c'est de faire oublier au Basic la plus récente position de retour. Vous pourrez ensuite utiliser un GOTO pour aller quelque part ailleurs dans le programme.
Ne sautez que rarement un RETURN. Un usage excessif de POP pour permettre des branchements avec GOTO à la sortie de sous-programmes mène à des programmes embrouillés, confus.

Sous-programmes emboîtés

Les sous-programmes peuvent être emboîtés. Cela signifie qu'un sous-programme peut en appeler un autre, qui peut à son tour en appeler un troisième, et ainsi de suite. Vous n'avez rien d'autre à faire — il n'est pas besoin d'instructions spéciales — pour emboîter des sous-programmes. Il vous suffit de vous brancher à un sous-programme avec un GOSUB et de terminer ce dernier avec un RETURN. Le Basic se souviendra de la ligne exacte à laquelle il doit faire retour après chaque sous-programme emboîté.
En voici un exemple :

```
10 REM PROGRAMME PRINCIPAL
20 REM DIMENSIONNEZ LES VARIABLES DES SOUS-PROGRAMMES
30 REM DANS LE PROGRAMME PRINCIPAL
40 REM C'EST UNE BONNE HABITUDE QUE DE LE FAIRE
50 REM AU DEBUT DU PROGRAMME PRINCIPAL ET POUR TOUTES LES
   VARIABLES
60 DIM A(99)
```



```
70 GOSUB 2000
80 REM AFFICHER QUELQUE CHOSE POUR PROUVER QUE LE RETOUR
   S'EST FAIT
90 PRINT "RETOUR FAIT!"
100 END
2000 REM PREMIER NIVEAU DE SOUS-PROGRAMME
2010 FOR I=0 TO 99
2020 A(I)=I
2030 GOSUB 3000
2040 NEXT I
2050 RETURN
3000 REM SOUS-PROGRAMME EMBOITE
3010 PRINT A(I)
3020 RETURN
```

Ce programme transfère l'instruction PRINT A (I) du sous-programme de la ligne 2000 au sous-programme emboîté de la ligne 3000. Aucun autre changement n'intervient.
Bien que ce puisse être parfaitement acceptable, et même souhaitable, un sous-programme peut en appeler un autre mais ne peut pas s'appeler lui-même. De même, un sous-programme ne peut pas en appeler un autre qui, à son tour, rappelle le premier. Un tel processus, appelé *récursion,* n'est pas autorisé en Basic sur Apple II.

Instruction GOSUB calculé

Les instructions GOTO et GOSUB sont fort semblables. La seule différence réside dans le fait que GOSUB se souvient du numéro de ligne suivant. On ne surprendra donc personne en indiquant qu'il existe un GOSUB calculé, comparable au GOTO calculé. Le GOSUB calculé sert à vous brancher à un sous-programme parmi plusieurs, en fonction de la valeur d'une expression numérique. Le GOSUB calculé se souvient où il doit retourner. Quel que soit le sous-programme appelé, le RETURN provoquera un branchement ramenant à la ligne dont le numéro avait été mémorisé.
Vous pouvez emboîter des sous-programmes avec le GOSUB calculé, tout comme avec le GOSUB standard.
Examinez l'instruction suivante, en Integer Basic :

```
>100 GOSUB A*500+2000
>110 REM
```

L'expression, sur la ligne 100, est un GOSUB calculé. Lorsque cette instruction sera exécutée, la logique du programme le branchera au sous-programme dont le numéro de ligne est donné par le calcul de l'expression. Dans cet exemple, si A = 0, le branchement se fera en 2000 ; en 2500 si A = 1, etc. Si la ligne calculée n'existe pas dans le programme, vous recevrez un message d'erreur ***BAD BRANCH ERR.
La version Applesoft du GOSUB calculé est un peu différente, mais fonctionne tout comme le GOTO calculé standard ;

```
]90

]100 ON A GOSUB 1000,500,5000,2300

]110 REM
```

Lorsque l'instruction de la ligne 100 est exécutée et si A = 1, le sous-programme sera celui commençant à la ligne 1000. Si A = 2, ce sera celui de la ligne 500 ; pour A = 3, on ira en

5000 et si A = 4, en 2300. Toute autre valeur de A fera que le programme se poursuivra à la ligne 110 (sans appel de sous-programme).

EXÉCUTION CONDITIONNELLE

Les GOTO et GOSUB calculés sont des instructions conditionnelles. Cela signifie que le déroulement ultérieur du programme dépend de la valeur d'une ou de plusieurs variables pouvant être modifiées au cours de l'exécution du programme. Le cheminement exact du programme dépend ainsi des conditions imposées aux variables.

Instruction IF-THEN

L'instruction IF-THEN (« si-alors ») est une autre instruction conditionnelle. Sa forme générale est :

IF *expression* THEN *instruction*

Si l'expression est *vraie,* alors l'instruction est exécutée. Les expressions relationnelles et booléennes sont très courantes avec IF-THEN, mais on peut employer aussi des expressions arithmétiques. Un programme Basic peut ainsi élaborer des décisions réelles. Voici trois exemples simples d'instructions IF-THEN :

```
10 IF A=B+5 THEN PRINT MSG$
40 IF CC$="M" THEN IN=0
50 IF Q<14 AND M<M1 THEN GOTO 66
```

A la ligne 10, on déclenche un PRINT si la valeur de la variable A est égale à la valeur de la variable B plus 5 ; sinon, le PRINT n'est pas exécuté.
EN 40, on établit à zéro la variable numérique IN si la variable chaîne CC$ est la lettre M.
En 50, on déclenche un branchement en 66 si la variable Q est inférieure à 14, et si M est inférieure à M1. Autrement, l'exécution du programme se poursuit à la ligne suivante.
Si vous ne comprenez pas l'évaluation des expressions suivantes IF, reportez-vous à la discussion sur les expressions, présentée dans ce chapitre.
Une instruction IF-THEN peut être suivie par d'autres instructions sur la même ligne. L'Integer Basic et l'Applesoft traitent différemment une telle situation.
En Integer Basic, seule l'instruction qui suit immédiatement THEN est exécutée conditionnellement. Les autres instructions sur la même ligne de programme sont toujours exécutées, que l'expression de IF-THEN soit vraie ou fausse. Illustrons ceci de la façon suivante :

```
10 IF V>100 THEN PRINT "VIVE LES VERTS": GOSUB 2000
20 T=T+V: PRINT T
```

Dans cet exemple, le programme imprime le message VIVE LES VERTS si la valeur de la variable dépasse 100. Le programme appellera alors le sous-programme commençant en 2000, quelle qu'ait été la valeur de V.
Applesoft, lui, n'exécutera l'instruction se trouvant sur la même ligne que IF-THEN que si l'expression de IF-THEN est vraie. Si elle est fausse, l'exécution se poursuit à la ligne suivante. Dans l'exemple précédent, si V est supérieur à 100, le programme imprimera VIVE LES VERTS et appellera la sous-programme de la ligne 2000. Mais si V est inférieur ou égal à 100, le programme n'imprimera pas le message mais n'appellera pas, non plus, le sous-programme. Il passera directement à la ligne suivante, ligne 20.

Une forme spéciale de l'instruction IF-THEN existe en Applesoft. Lorsque l'ordre conditionnel est un GOTO, vous pouvez l'omettre si vous le désirez. Ainsi, les deux instructions suivantes sont équivalentes :

```
110 IF MM$=DD$ THEN GOTO 100
```

fait la même chose que :

```
110 IF MM$=DD$ GOTO 100
```

INSTRUCTIONS D'ENTRÉE ET DE SORTIE

Des instructions variées commandent, en Basic, le transfert d'informations de, et vers l'ordinateur. On les appelle collectivement *instructions d'entrées-sorties.* Les plus simples servent à entrer les informations du clavier et à sortir des informations sur l'écran. Nous allons en discuter ici. Mais il en existe aussi de plus complexes qui organisent les échanges entre l'ordinateur et les périphériques, telles qu'unités à cassettes, unités à disquettes, et imprimantes. Elles seront décrites dans les chapitres 4 et 5. Le chapitre 6 couvre les instructions de sortie pour le graphique.
Nous avons déjà rencontré l'instruction PRINT, qui sort des données vers l'écran. Aussi va-t-on commencer par cette instruction, avant d'examiner les autres.

Instruction PRINT

Pourquoi utiliser l'ordre PRINT (littéralement, *imprimer*) plutôt que DISPLAY (« afficher ») ou toute autre abréviation ? C'est que vers les années 60, lorsque le langage Basic a été créé, les affichages étaient très chers ; aussi, les ordinateurs de moyenne ou de bas de gamme n'en disposaient souvent pas. Le terminal informatique standard disposait d'un clavier et d'une imprimante ; c'est pourquoi le mot PRINT a été retenu, et il s'étend à tout ce que provoque un affichage.
L'instruction PRINT affichera du texte ou des nombres. Par exemple, la ligne suivante affichera le mot TEXTE :

```
10 PRINT "TEXTE"
```

Pour afficher un nombre, vous le placerez, lui ou un nom de variable, après PRINT :

```
>A=10

>PRINT 5,A
5          10
```

L'instruction ci-dessus affiche le nombre 5, puis le nombre 10 sur la même ligne.
Vous pouvez afficher un mélange de texte et de nombres en listant les informations à afficher après PRINT. Elles seront séparées par des virgules. Le PRINT suivant affiche les mots UN, DEUX, TROIS, QUATRE et CINQ, suivis par leur symbole numérique :

```
10 PRINT "UN", 1, "DEUX", 2, "TROIS", 3, "QUATRE", 4,
       "CINQ", 5
20 END
```

Si vous séparez les variables par des virgules, comme ci-dessus, Apple II attribuera automatiquement un nombre fixe d'espaces à chaque variable affichée. Vérifiez-le en mode immédiat. Si vous voulez remplir ces espaces, séparez ces variables avec des points-virgules :

```
10 PRINT "UN"; 1; "DEUX"; 2; "TROIS"; 3; "QUATRE"; 4;
        "CINQ"; 5
20 END
```

A nouveau, introduisez cet ordre en mode commande et exécutez-le.
Une instruction PRINT ramènera automatiquement le curseur au bord gauche et à la ligne du dessous lorsque son action cesse. En jargon informatique, on appelle cette action un *retour chariot*. Vous pouvez supprimer ce retour chariot en plaçant une virgule ou un point virgule après la dernière information dans la liste suivant PRINT. Une virgule déplacera le curseur à la position où la valeur suivante (s'il y en a une) sera affichée. Pour illustrer ceci, frappez les trois instructions suivantes et exécutez-les en faisant RUN ;

```
10 PRINT "UN", 1, "DEUX", 2
20 PRINT "TROIS", 3, "QUATRE", 4
30 END
```

Maintenant, ajoutez une virgule à la fin de la ligne 10 et ré-exécutez ce programme avec RUN. Vous constaterez que les deux lignes à afficher le sont maintenant sur une seule. Remplacez la virgule à la fin de la ligne 10 par un point virgule et lancez à nouveau le programme. L'affichage se fait sur une seule ligne mais il n'y a plus d'espace entre le chiffre 2 et le mot TROIS. En remplaçant sélectivement les autres virgules par des points-virgules, vous supprimerez sélectivement les espaces.
Nous avons affiché des valeurs numériques en les introduisant directement dans l'instruction PRINT. Vous pourriez, à la place, afficher le contenu de variables. Le programme suivant fait la même chose que notre exemple précédent PRINT, mais se sert d'un tableau A() pour créer les digits. Introduisez-le en machine et lancez-le :

```
 5 DIM A(5)
10 FOR I=1 TO 5
20 A(I)=I
30 NEXT I
40 PRINT "UN"; A(1); "DEUX"; A(2); "TROIS"; A(3); "QUATRE";
   A(4); "CINQ"; A(5)
50 END
```

En Applesoft, on pourrait placer les mots à afficher dans des variables chaînes et introduire le PRINT dans une boucle FOR-NEXT :

```
10 DATA "UN", "DEUX", "TROIS", "QUATRE", "CINQ"
20 FOR I = 1 TO 5
40 READ N$
50 PRINT N$;I;
60 NEXT I
70 END
```

Instruction INPUT

Lorsqu'une instruction INPUT (« entrée ») est exécutée, l'ordinateur attend une frappe au clavier ; tant qu'il ne l'aura pas obtenue, il ne se passera rien.
Dans sa forme la plus simple, cette instruction commence par INPUT, suivie du nom de la variable. Les données frappées au clavier se voient attribuées à la variable indiquée. Son nom détermine le type de données que vous pourrez introduire. Un nom de variable

numérique ne se satisfera que d'une entrée numérique. Pour en faire la démonstration, frappez ce court programme et lancez-le (en essayant de frapper des données alphabétiques aussi, pour voir ce qui se passe) :

```
10 INPUT A
20 PRINT A
25 REM FIN DU PROGRAMME SI A=0
30 IF A = 0 THEN END
40 GOTO 10
```

Lorsqu'il rencontre une instruction INPUT, l'ordinateur affiche un point d'interrogation et attend votre entrée. Le programme ci-dessus affiche chaque touche que vous pressez. En jargon informatique, on dit qu'il se fait l'*écho* du clavier. De plus, le nombre est affiché à nouveau en raison de l'ordre PRINT de la ligne 20. Le premier affichage survient lorsque l'INPUT de la ligne 10 est exécuté et que vous frappez sur le clavier. Le second résulte de l'ordre PRINT de la ligne 20.
Une instruction INPUT peut acquérir plus d'une valeur à la fois. Pour ce faire, listez toutes les variables que vous voulez à la suite du mot INPUT. Séparez les variables par des virgules. Lorsqu'un tel ordre INPUT sera exécuté, vous devrez répondre en séparant les variables par des virgules. Assurez-vous que chaque valeur est bien du même type que la variable à laquelle elle est affectée.
Lorsque vous répondez à un INPUT, n'employez pas de virgule décimale ou autre.
Le programme suivant acquiert deux valeurs numériques et les affiche :

```
20 INPUT A,B
30 PRINT A,B
35 REM FIN DU PROGRAMME SI UN 0 EST FRAPPE
40 IF A=0 OR B=0 THEN END
50 GOTO 20
```

Lancez ce programme et essayez de frapper un nombre suivi par une virgule, puis un second nombre, et pressez RETURN. Maintenant, essayez quelque chose d'un peu différent. Entrez un nombre et pressez RETURN. Comme vous le constaterez, Apple II vous enjoindra d'entrer une seconde valeur. Faites-le, et pressez RETURN. Ainsi, lorsqu'une instruction INPUT vous demande plusieurs valeurs numériques, vous pourrez soit les introduire en une seule ligne, soit les présenter sur des lignes séparées.
L'ordre INPUT fonctionne un peu différemment avec des variables chaînes en Integer Basic. D'abord, il n'affiche pas de point d'interrogation. Essayez avec cet exemple :

```
10 DIM A$(19)
20 INPUT A$
30 PRINT A$
35 REM FIN DU PROGRAMME SI ENTREE NULLE
40 IF A$="" THEN END
50 GOTO 20
```

Après avoir lancé ce programme, essayez d'entrer une chaîne de plus de 20 caractères. Vous obtiendrez un ***STR OVFL ERR, message d'erreur, et le programme stoppera. La longueur de la chaîne introduite ne peut excéder la longueur maximale de la variable chaîne employée dans l'instruction INPUT.

L'Integer Basic vous oblige à introduire chaque chaîne sur une ligne distincte. Si un ordre INPUT spécifie une liste de variables, parmi lesquelles des variables chaînes, celles-ci devront être introduites sur des lignes séparées. Cela, parce que l'Integer Basic autorise l'usage de virgules dans une valeur chaîne. Vous pouvez vous en assurer en lançant le programme ci-dessus et en frappant DURAND, LOUIS. L'exemple suivant montre ce qui arrive lorsqu'une variable chaîne fait partie d'une instruction Basic en Integer Basic. Essayez ce programme ; essayez d'entrer les quatre valeurs sur la même ligne, en les séparant par des virgules. Que se passera-t-il ? Entrez maintenant ces valeurs sur des lignes séparées. Trouvez ce qui survient si vous frappez une valeur numérique ou une virgule dans une valeur chaîne :

```
10 DIM A$(10),B$(10)
20 INPUT A$,A,B$,B
30 PRINT A$,A,B$,B
35 REM FIN DU PROGRAMME SI ENTREE NULLE
40 IF A$="" THEN END
50 GOTO 20
```

Ainsi que nous en avons déjà discuté, une variable réelle peut avoir une valeur entière en Applesoft. De ce fait, vous pourrez introduire une valeur entière pour une variable réelle. Par contre, une valeur réelle entrée pour une entière sera convertie en valeur entière selon les règles énoncées précédemment dans ce chapitre.

Messages d'appel de INPUT

L'instruction INPUT est très troublante ; sa syntaxe est trop exigeante pour un opérateur humain normal. Imaginez qu'un employé de bureau, sans aucune connaissance informatique, se heurte à ces innombrables messages d'erreur que sa maladresse va déclencher : en désespoir de cause, il finira par abandonner. Vous serez alors amené à passer bien du temps pour rédiger des programmes de « test d'idioties » vérifiant les entrées. Il s'agit de programmes surveillant tous les types d'erreurs qu'on peut commettre en introduisant des données. Un tel programme traitera ces erreurs de façon que n'importe quel opérateur comprenne ce qu'il a à faire. Le chapitre 4 décrit en détail ces techniques de programmation d'entrées.

Une astuce qu'on peut immédiatement noter, cependant, est la faculté qu'à l'instruction PRINT d'afficher un court message explicitant ce qu'elle attend. Ce message apparaît dans l'ordre PRINT comme une chaîne, entre guillemets. Il sera affiché juste au-dessus de la demande de réponse. Ce sera certainement très utile lorsque vous aurez à introduire plusieur valeurs, affectées à des variables dont vous ne pourriez vous souvenir de l'ordre, du type, ou de la signification.

En Integer Basic, vous placerez ce *message d'appel* juste après le mot INPUT. Il sera suivi par une virgule, puis par la liste des variables. Si la liste contient plus d'une variable, le message sera affiché à nouveau sur la première ligne de l'entrée. Si la première variable de la liste est numérique, un point d'interrogation sera aussitôt affiché. S'il s'agit d'une chaîne, il n'y aura pas de point de d'interrogation. En voici un exemple :

```
10 DIM A$(10)
20 INPUT "DONNEZ VOTRE NOM ET VOTRE AGE ", A$,A
30 PRINT A$; "EST AGE DE"; A
```



```
35 REM SI ENTREE NULLE, FIN DU PROGRAMME
40 IF A$="" THEN END
50 GOTO 20
```

En Applesoft, vous placerez le message d'appel juste après INPUT. Il sera suivi par un point-virgule, puis par la liste des variables. L'existence d'un message d'appel supprime le point d'interrogation standard. Ce message n'est affiché qu'une fois, même si plusieurs lignes sont requises pour entrer toutes les valeurs demandées par la liste de variables. Voici un exemple :

```
20 INPUT "DONNEZ VOS NOM ET AGE:"; A$,A
30 PRINT A$; " EST AGE DE "; A
35 REM SI ENTREE NULLE, FIN DU PROGRAMME
40 IF A$="" THEN END
50 GOTO 20
```

L'instruction GET

Disponible seulement en Applesoft, l'instruction GET entre un seul caractère du clavier ; elle ne l'affiche pas sur l'écran. Vous n'avez pas à presser RETURN. L'entrée est traitée comme une valeur chaîne ou une valeur numérique, selon le type de variable qui suit GET. Introduisez ce programme et lancez-le :

```
10 GET A$
20 PRINT A$
25 REM SI ENTREE EST UN E, FIN DU PROGRAMME
30 IF A$="E" THEN END
40 GOTO 10
```

On peut demander à GET d'attendre un caractère précis de la façon suivante :

```
10  GET A$
20  IF A$ <  > "X" THEN  GOTO 10
30  PRINT A$
40  END
```

Ce programme attend que la lettre X soit entrée. Rien d'autre ne le satisfera.

Si l'ordre GET spécifie un entier ou une variable réelle, l'entrée doit être un digit numérique. Sinon, le message ?SYNTAX ERROR apparaît et le programme stoppe. C'est pour cela, et pour d'autres problèmes pouvant surgir lorsqu'on emploie GET avec une variable numérique, que l'instruction GET s'appliquera généralement à des chaînes de caractères.

On utilise le plus souvent GET dans des programmes pour créer un dialogue avec l'opérateur. Par exemple, un programme attendra une entrée témoignant que l'opérateur est présent. Voici ce que donnerait la logique d'un tel programme :

```
10 PRINT "OPERATEUR, ES-TU LA?"
15 PRINT "SI OUI, FRAPPE Y POUR YES"
20 GET A$
30 IF A$ < > "Y" THEN GOTO 20
40 PRINT "OK, POURSUIVONS"
50 END
```

Remarquez que cette séquence n'affiche pas le caractère frappé au clavier. Essayez de ré-écrire ce programme de façon qu'il affiche le caractère frappé en réponse à GET.

HALTE ET REPRISE DE L'EXÉCUTION D'UN PROGRAMME

Si vous voulez arrêter un programme qui tourne, pressez simultanément CTRL et C. Si, à ce moment, le programme attendait une entrée du clavier via un ordre INPUT, vous devrez presser RETURN après CTRL-C.
En Integer Basic, vous obtiendrez alors le message STOPPED AT suivi du numéro de la ligne où s'est produite la halte. Vous reprendrez l'exécution en frappant la commande CON.
En Applesoft, le message affiché sera BREAK IN, suivi par le numéro de la ligne où s'est produit l'arrêt, quand vous avez pressé CTRL-C. Vous pourrez reprendre l'exécution en frappant CONT.

La touche RESET

Bien entendu, on peut interrompre un programme à tout moment en pressant la touche RESET (ou, sur quelques versions de l'Apple II, CTRL-RESET).
Avec l'Apple II Plus et avec l'Apple II disposant de la carte Système Langage, RESET a le même effet que CTRL-C.
Sur des versions de l'Apple II sans le moniteur Autostart, presser RESET fera apparaître le caractère d'appel du moniteur (*). Si vous êtiez en Integer Basic ou en Applesoft firmware, faites CTRL-C pour revenir au langage utilisé. Si vous utilisiez un Applesoft en cassette, frappez OG pour retourner à l'Applesoft. S'il s'agissait de l'Applesoft disquette, frappez 3DOG.
Si, après un RESET accidentel, vous faites une tentative incorrecte pour revenir à votre programme, vous le perdrez totalement.

L'instruction END

Le programme cessera l'exécution dès qu'il rencontrera l'instruction END, ainsi qu'on l'a montré dans ce chapitre.
Vous ne pourrez poursuivre l'exécution d'un programme en Integer Basic après que celui-ci ait exécuté un END.

L'instruction STOP

Applesoft dispose d'une commande supplémentaire qui sert à arrêter l'exécution lorsque le programme la rencontre : la commande STOP. Lorsqu'Applesoft exécute un STOP, il affiche un message BREAK IN (« pause, arrêt, rupture à... ») suivi du numéro de la ligne où s'est produit l'arrêt.
Vous pourrez reprendre l'exécution du programme, en Applesoft et soit après un END, soit après un STOP, en frappant CONT *(continuer)*.

L'instruction WAIT

Applesoft dispose aussi d'une instruction enjoignant au programme d'effectuer une pause. Avec l'instruction WAIT, l'exécution du programme cesse jusqu'à ce qu'une position mémoire (que vous spécifiez) atteigne une valeur que vous avez spécifiée. Vous pouvez, par exemple, déclencher une pause jusqu'à ce que quelqu'un presse le bouton de la commande de jeu numéro 1. Voici comment :


```
10 REM ATTENTE D'UNE ACTION SUR LE BOUTON JEU NO.1
20 PRINT "PRESSEZ LE BOUTON DE JEU NO.1"
30 WAIT - 16286,128
40 PRINT "BANG!"
50 END
```

Reportez-vous au chapitre 8 pour plus de détails sur l'ordre WAIT.

FONCTIONS

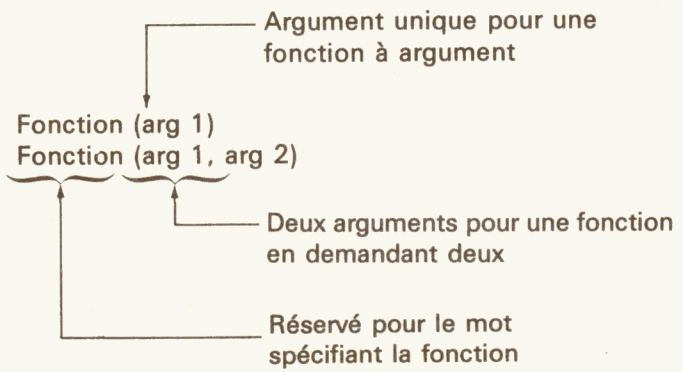
Basic comprend un autre type d'éléments, les fonctions ; elles ressemblent sous certains aspects aux variables mais par ailleurs, agissent aussi comme des instructions.
Pour comprendre ce qu'est une fonction, le plus simple consiste à prendre un exemple. Dans l'instruction d'affectation :

```
110 A=SQR (B)
```

la variable A se voit attribuer comme valeur la racine carrée de B. SQR spécifie la fonction racine carrée. Voici la fonction chaîne :

```
20 L=LEN(D$)
```

Dans cet exemple, la variable L se voit affecter la longueur de la variable chaîne D$.
Les fonctions peuvent se substituer aux variables ou aux constantes n'importe où dans une instruction Basic, sauf à la gauche d'un signe égal. En d'autres mots, vous pouvez écrire A = SQR(B).
La discussion qui suit va vous montrer comment utiliser les fonctions. Un résumé incomplet des fonctions en Integer Basic et en Applesoft est présenté ici, mais leur description complète est donnée dans le chapitre 8. De nombreuses fonctions ne sont pas disponibles en Integer Basic ; nous les signalerons.
Vous spécifiez une fonction par le nom réservé approprié (tel que SQR pour racine carrée), suivi par un ou plusieurs arguments entre parenthèses. Dans le cas de A = SQR(B), SQR ne requiert qu'un argument, ici la valeur dont il faut extraire la racine carrée. Pour L = LEN (D$), LEN spécifie la fonction ; l'argument D$, placé entre parenthèses, est la chaîne dont il faut prendre la longueur.
De façon générale, chaque fonction aura l'un de ces deux formats :



Peu de fonctions demandent trois arguments. Chaque argument d'une fonction peut être une constante, une variable ou une expression.
Chaque fonction, dans une instruction Basic, est ramenée à une valeur numérique simple, ou une valeur chaîne avant que le reste de l'instruction soit évalué. D'abord, l'argument de la fonction est évalué en respectant les règles que nous avons présentées antérieurement. Dès qu'elle a été réduite à une valeur numérique ou chaîne, la fonction lui est appliquée, menant à une autre valeur numérique ou chaîne. Une expression ne sera pas évaluée tant que toutes les fonctions qu'elle contient n'ont pas été évaluées. Par exemple et dans l'instruction suivante :

```
]10 B=24.7*(SQR(C)+5)-SIN(0.2+D)
```

Les fonctions SQR et SIN sont évaluées en premier. Supposons que SQR (C) = 6.72 et que SIN (0.2 + D) = 0.625. L'expression de la ligne 10 sera d'abord ramenée à :

$$24.7*(6.72+5)-0.625$$

qui est alors plus simple à évaluer.

FONCTIONS NUMÉRIQUES

Voici une liste des fonctions numériques que vous pourrez employer aussi bien avec l'Integer Basic qu'avec l'Applesoft.

SGN	Retourne le signe d'un argument : +1 s'il est positif, −1 s'il est négatif, 0 si l'argument est zéro.
ABS	Retourne la valeur absolue d'un argument. Un argument positif ne change pas ; un argument négatif est converti en son équivalent positif.
RND	Crée un nombre aléatoire. Reportez-vous au chapitre 8 pour plus de détails.

Voici une liste de fonctions utilisables en Applesoft uniquement.

INT	Convertit un argument en virgule flottante en son équivalent entier.
SQR	Calcule la racine carrée d'un argument.
EXP	Elève la constante *e* à la puissance de l'argument (e^{arg}).
LOG	Retourne le logarithme naturel de l'argument.
SIN	Retourne le sinus de l'argument, en radians.
COS	Retourne le cosinus de l'argument, en radians.
TAN	Retourne la tangente de l'argument, en radians.
ATN	Retourne l'arc tangente de l'argument, en radians.

Utilisation des fonctions numériques

Vous devriez très vite utiliser les fonctions numériques, sans vous soucier de celles que vous ne comprenez pas encore. Par exemple, si vous ne connaissez pas la trigonométrie, vous n'emploierez probablement pas les fonctions SIN, COS, et TAN dans vos programmes.

Voici un exemple d'emploi des fonctions numériques :

```
10 A=-234
20 B= SGN (A)
30 PRINT B
40 END
```

Quand vous exécuterez ce programme, le résultat affiché sera −1 car −234 est négatif. A titre d'exercice, modifiez la ligne 10 pour introduire un INPUT. Modifiez la ligne 40 et placez un GOTO 10. Désormais, vous pouvez entrer de nombreuses valeurs pour A et vérifier le fonctionnement de la fonction SGN.
Voici un programme plus complexe en Applesoft avec des fonctions numériques :

```
10 INPUT A,B
20 IF  LOG (A) < 0 THEN A = 1 / A
30 PRINT  SQR (A) *  EXP (B)
39 REM FRAPPEZ CTRL-C POUR TERMINER
40 GOTO 10
```

Si vous ne connaissez pas les logarithmes, et à titre d'exercice, modifiez la ligne 20 et remplacez LOG par une autre fonction numérique.

FONCTIONS CHAÎNES

Les fonctions portant sur des chaînes servent à manipuler les données des chaînes de multiples façons. Vous pouvez éviter l'emploi des fonctions numériques que vous ne connaissez pas, mais vous devrez faire l'effort d'apprendre ces fonctions de chaînes.
En voici une liste que vous pourrez appliquer avec l'Integer Basic et l'Applesoft :

ASC	Convertit un caractère chaîne en sa valeur numérique (ASCII) équivalente.
LEN	Retourne le nombre de caractères contenu dans une chaîne.

Les fonctions de chaînes suivantes ne sont disponibles qu'en Applesoft :

STR$	Convertit une valeur numérique en une chaîne de caractères.
VAL	Convertit une chaîne de caractères en valeur numérique équivalente (si une telle conversion est possible).
CHR$	Convertit un code numérique (ASCII) en son caractère équivalent.
LEFT$	Extrait la partie gauche d'une chaîne. Les arguments de cette fonction identifient la chaîne et la longueur à extraire.
RIGHT$	Extrait la partie droite d'une chaîne. Les arguments de cette fonction identifient la chaîne et la longueur à extraire.
MID$	Extrait le milieu d'une chaîne. Les arguments de cette fonction identifient la chaîne et la longueur à extraire.

Ainsi, les fonctions de chaînes servent à déterminer la longueur d'une chaîne, à en extraire certaines portions, à convertir des valeurs numériques, des codes (ASCII) et des caractères. En voici quelques exemples :

STR$(14)	Convertit 14 en " 14 "
LEN("ABC")	Retourne la longueur de la chaîne : 3, puisque celle-ci contient ici, trois caractères.
LEN(A$+B$)	Retourne la longueur combinée des deux chaînes.
LEFT$(ST$,1)	Retourne le caractère le plus à gauche de la chaîne ST$.

Sous-chaînes en Integer Basic

Bien que l'Integer Basic ne dispose pas des fonctions permettant d'extraire des portions de chaînes, il existe une méthode menant au même résultat. Vous spécifiez la position de départ et le nombre de caractères d'une sous-chaîne, comme dans l'exemple suivant :

```
10 DIM A$(20),B$(5)
20 B$=A$(1,4)
```

Ici, B$ est établi pour être égal aux quatre premiers caractères de A$.
Tout se passe comme si B$ se voyait affecter la valeur d'un des éléments d'un tableau A$(), mais rappelez-vous que l'Integer Basic n'accepte pas les tableaux de chaînes, et encore moins des tableaux à deux dimensions de chaînes. Cette notation se réfère donc, ici, à une sous-chaîne. La première valeur entre parenthèses est la position de départ de la sous-chaîne, et la seconde, le nombre de ses caractères.

Concaténation de chaînes en Integer Basic

La fonction LEN sert à concaténer des chaînes en Integer Basic. En voici un exemple :

```
10  DIM A$(10),B$(10),C$(10)
20 A$="OLEO"
30 B$="CALO"
40 C$="DUC"
50  A$( LEN(A$)+1)=B$
60  PRINT A$
70  B$( LEN(B$)+1)=C$
80  PRINT B$
90  END

>RUN
OLEODUC
CALODUC
```

FONCTIONS SYSTÈME

Afin d'être complet, voici encore quelques fonctions relatives au système. Elles exécutent des fonctions que vous n'utiliserez probablement pas tant que vous n'aurez pas acquis quelque expérience :

PEEK	Va lire le contenu d'une position mémoire.
FRE	Retourne l'espace mémoire encore disponible, c'est-à-dire le nombre d'octets encore libre dans la mémoire à lecture-écriture (RAM). N'existe pas en Integer Basic.
USR	Transfert le contrôle à un programme en langage assembleur. Non disponible avec l'Integer Basic.

FONCTIONS DÉFINIES PAR L'UTILISATEUR

En plus de ces fonctions, qui participent au Basic standard, vous pouvez définir des fonctions arithmétiques qui vous sont propres en Applesoft, pour autant qu'elles ne soient pas trop complexes. Les fonctions de chaînes « utilisateur » ne sont pas admises.
Une instruction DEF FN servira à les définir. Vous l'emploierez pour définir une fonction de votre cru. Voici un court programme définissant une fonction :

```
10 DEF FN P(X) = 100 * X
20 INPUT A
30 PRINT A, FN P(A)
35 REM FRAPPEZ CTRL-C POUR SORTIR DE CE PROGRAMME
40 GOTO 20
```

L'identification de fonction suit le mot réservé FN. Les règles, pour nommer une fonction, sont les mêmes que pour les variables. Ici, nous avons usé de P et de ce fait, la fonction devient FNP. Si l'identification avait été AB, et non plus P, le nom de la fonction aurait été FNAB.
L'expression arithmétique, à droite du signe égal, définit la fonction. Lorsque vous appellerez cette fonction, l'expression sera évaluée (à partir des valeurs courantes des variables qui y interviennent). Le résultat numérique sera traité de la même façon que toute autre valeur numérique, dans le contexte de l'instruction FN.
Avec l'instruction DEF FN, une seule variable doit suivre l'identificateur de fonction, et doit être incluse dans des parenthèses. Il s'agit d'une variable *locale ;* elle n'a pas d'effet hors de l'instruction DEF FN. Vous pourrez utiliser le même nom de variable n'importe où dans votre programme sans affecter la fonction, et sans que la variable de la fonction n'affecte cette autre variable portant le même nom dans votre programme.
A l'usage, l'instruction FN doit être suivie par l'identificateur de fonction, puis par une constante numérique, une variable ou une expression mise entre parenthèses. Lorsqu'Applesoft rencontre l'instruction FN, il assigne la valeur de la constante, variable ou expression à la variable locale de l'instruction DEF FN. (La valeur d'une autre variable externe portant le même nom ne change pas.) Si la variable locale apparaissait dans l'expression définissant la fonction, Applesoft utilisera la valeur la plus récente qui lui aura été attribuée lors de l'évaluation de l'expression.

EMBOÎTAGE DES FONCTIONS

L'argument d'une fonction peut être une expression ; cette expression peut contenir des fonctions. En d'autres termes, les fonctions peuvent être emboîtées. En voici un exemple :

```
10 INPUT A
20 PRINT SGN ( ABS (A) )
25 REM FRAPPEZ CRTL-C POUR SORTIR DE CE PROGRAMME
30 GOTO 10
40 END
```

Expérimentez ce programme en créant des instructions en mode immédiat qui intègrent des fonctions numériques et des chaînes complexes.

CHAPITRE 4

PROGRAMMATION AVANCÉE EN BASIC

Ce chapitre développe le précédent en montrant comment programmer l'Apple II en Basic. Il couvre de nombreuses instructions nouvelles Basic et explore d'autres facettes de certaines qui nous sont déjà familières. Le chapitre 3 vous en disait assez pour démarrer avec l'Apple II ; celui-ci va vous montrer comment en faire un instrument utile.

ACCÈS DIRECT ET COMMANDES

Un certain nombre d'instructions vous permettent d'accéder directement à l'Apple II. Vous pourrez accomplir beaucoup de choses uniquement via ces instructions, et par exemple, procéder à des commandes de jeux, animer le haut-parleur, et employer pleinement vos périphériques.

MÉMOIRE ET ADRESSAGE

L'Apple II peut disposer de 65536 positions mémoires adressables séparément ; chacune de celles-ci peut stocker un nombre compris entre 0 et 255 (cette valeur bizarre est tout simplement l'équivalent de 2^8). Tous les programmes et données sont convertis dans des séquences de nombres ainsi rangés.

Vous devez spécifier une position mémoire pour chacune des instructions suivantes. Vous pouvez spécifier l'adresse avec un nombre, une variable ou une expression. Dans tous les cas, leur évaluation devra mener à une position mémoire valide. On dispose de deux adresses valides pour chaque position mémoire. La première est positive et est un entier compris entre 0 et 65535. La seconde est négative et peut être obtenue en soustrayant 65536 de l'adresse positive. Par exemple, −32767 et 32768 adressent la même position. Une autre position sera adressée à la fois par −1 ou 65535.

Si vous vous souvenez qu'en Integer Basic, le plus grand nombre admis est 32767, vous comprendrez l'utilité de ces nombres négatifs pour adresser les positions supérieures de la mémoire.

Si vous spécifiez une position mémoire avec une valeur réelle, en Applesoft, elle sera convertie en entier puis servira d'adresse.

PEEK et POKE

La fonction PEEK vous sert à lire la valeur rangée dans une quelconque position mémoire de votre Apple II. Considérez l'instruction suivante :

```
10 A = PEEK(200)
```

Cette instruction affecte le contenu de la position mémoire 200 à la variable A.
L'instruction POKE range une valeur dans une position mémoire. Par exemple :

```
20 POKE 8000,A
```

rangera la valeur de A à la position mémoire 8000. La valeur inscrite en mémoire peut être un nombre, une variable, ou une expression, avec une valeur comprise entre 0 et 255. Vous pouvez utiliser PEEK avec des mémoires à lecture-écriture (RAM) ou des mémoires à lecture seulement (ROM). Mais vous ne pouvez employer POKE qu'avec des mémoires à lecture-écriture. C'est évident : une ROM ne peut qu'être lue.

Instruction CALL

Vous pouvez transférer les commandes de votre Basic à un programme en langage assembleur, ou à un sous-programme, avec l'instruction CALL (« appel »). Examinez l'instruction suivante :

```
100 CALL A1
```

Elle transfère le contrôle à la position mémoire spécifiée par la variable A1.
Le programme, ou sous-programme en langage assembleur, peut être l'un de ceux qui résident en permanence dans les mémoires ROM de votre Apple II, ou être d'une origine autre. Le moniteur dispose d'un sous-programme intrinsèque qui efface l'écran, par exemple. L'appendice D donne la liste complète des sous-programmes intrinsèques que vous pourrez utiliser. Vous pourrez aussi vous référer au chapitre 7 pour une étude plus complète du moniteur et du langage assembleur.

Instructions HIMEM: et LOMEM:

La mémoire de votre Apple II est employée de diverses façons. Une partie sert à stocker votre programme en Basic, une autre est consacrée à vos variables, une troisième est réservée aux programmes de gestion des unités à disques (si vous en disposez), etc. Une partie de la mémoire servira au graphique, comme on le verra au chapitre 6.
Les instructions HIMEM: et LOMEM: vous servent à réserver des zones en mémoire pour vos programmes en langage assembleur et pour vos graphiques à haute résolution. Ces mots proviennent, pour HIMEM, de « High Memory », *mémoire haute,* et pour LOMEM, de « Low Memory », *mémoire basse.*
Voici un exemple :

```
50  HIMEM: 38400
60  LOMEM: 12291
```

C'est votre Apple II qui établira ensuite où il rangera votre programme en Basic, compte tenu de ces limites et des positions mémoires disponibles en haut et en bas de la mémoire. Vous pourrez avoir à remettre à zéro ces limites si vous travaillez en langage assembleur ou en graphique à haute résolution, ce que nous examinerons dans les chapitres 6 et 7. Pour plus d'informations sur les mémoires, voyez l'appendice G.

UTILISATION DE PÉRIPHÉRIQUES

Lorsque vous mettez votre Apple II sous tension, il attend les ordres que vous frapperez au clavier, en entrée, et les affichera sur l'écran, en sortie. Le clavier est le périphérique standard d'entrée et l'écran, le périphérique standard de sortie. Mais il existe d'autres périphériques, parmi lesquels :

- L'unité à cassettes, pour sauvegarder ou charger des programmes (et quelques données).
- L'unité à disquettes, pour sauvegarder ou charger programmes et données.
- Une imprimante, pour obtenir des copies papier de programmes ou de données.
- Une tablette graphique, pour introduire des schémas à main levée.
- Des dispositifs de communication avec d'autres ordinateurs.

L'unité à cassettes se connecte à l'Apple II via deux jacks spéciaux.
Tous les autres périphériques d'entrée ou de sortie s'enfichent dans l'Apple II via des cartes appelées *contrôleurs, cartes d'interface, cartes* ou *interfaces* tout simplement. Vous spécifiez le périphérique que vous allez utiliser par le numéro du connecteur femelle dans lequel vous enficherez son contrôleur. Il existe huit connecteurs femelles, numérotés de 0 à 7. L'écran et le clavier sont toujours reliés au connecteur 0.

Les instructions PR# et IN#

Pour sélectionner un autre connecteur femelle, utilisez l'instruction PR# pour une sortie. Pour une autre source d'entrée, employer IN#. La commande suivante, en mode immédiat, sélectionne l'écran en sortie :

```
PR#0
```

Tant que le système d'exploitation disque (DOS) n'est pas présent, vous utiliserez PR# et IN# de la même façon en mode programmé et en mode immédiat. Avec le DOS, l'usage de PR# et IN# est un peu plus compliqué. Vous devrez passer par une instruction PRINT affichant un caractère CTRL-D, suivie immédiatement par la commande PR# ou IN#. L'instruction suivante dirige le PRINT sur la sortie, connecteur femelle 1 ; via l'interface connectée ici, la sortie se fera sur une imprimante ou tout autre périphérique :

```
100 D$="": REM CTRL-D
110 PRINT D$; "PR#1": REM SELECTION CONNECTEUR
    NUMERO 1 POUR IMPRIMANTE
```

Si le DOS était absent, vous auriez eu à faire :

```
110 PR#1: REM SELECTION CONNECTEUR 1
    POUR IMPRIMANTE
```

Remarquez que PR# et IN# disposent chacun d'un paramètre, qui doit être une valeur numérique comprise entre 0 et 7. Toute autre valeur provoquera des effets imprévisibles. S'il n'y a pas de carte interface dans le connecteur visé par PR# ou IN#, Apple II se verrouille et il vous faudra recourir à RESET.

SORTIES DE PROGRAMMES ET ENTRÉES DE DONNÉES

Le programmeur le plus inexpérimenté découvre vite que les sections d'entrées et de sorties de son programme sont remplies de pièges.
Chaque programme doit pratiquement utiliser les données qui seront entrées via le clavier. Quelques instructions INPUT suffiront-elles ? Le plus souvent, la réponse est non. Que se passera-t-il si l'opérateur frappe la mauvaise touche ? Ou pire, que se passerait-il si l'opérateur découvrait qu'il a introduit la mauvaise donnée, et cela, après avoir entré les deux ou trois données suivantes ? Un programme exploitable doit tenir compte du caractère humain de l'opérateur, qui pourra produire toutes les erreurs humaines concevables. De ce fait, les résultats ne peuvent pas être simplement affichés, ou imprimés, après une brassée d'instructions INPUT. Un humain lira aussi ces résultats, et si la sortie n'a pas été parfaitement conçue, elle pourra être très difficile à lire ; en conséquence, des erreurs de lecture pourront survenir, et on pourra même totalement passer à côté de certaines informations.
Par bonheur, le Basic de l'Apple II dispose de nombreuses possibilités facilitant la programmation d'entrées et de sorties correctes. Nous allons en décrire quelques-unes avant de traiter de la bonne façon de programmer des entrées et des sorties.

UN PEU PLUS SUR L'INSTRUCTION PRINT

Normalement, une instruction PRINT se termine par un retour à la ligne suivante de l'écran (équivalent à un *retour chariot*). On peut ainsi commencer le PRINT suivant tout au début de la ligne. Ainsi, le programme suivant affiche une colonne de 20 caractères W en première position de 20 rangées :

```
>NEW

>200 C$="W"
>210 FOR I=1 TO 20
>220 PRINT C$
>230 NEXT I
```

Utilisation du point-virgule

Un point-virgule(;) placé après une variable dans une instruction PRINT fait que l'affichage suivant se fera immédiatement à la suite. Le point-virgule peut ainsi séparer des paramètres dans une liste suivant PRINT, ou apparaîtra après la dernière (ou la seule) variable de l'instruction PRINT, supprimant le retour chariot. Ainsi, le programme suivant affichera 800 caractères W sur 20 rangées de 40 colonnes (figure A ci-contre).
L'index de la boucle FOR-NEXT, la variable I, sert de compteur pour indiquer le nombre de W à afficher, et dans ce cas, 800. A la première exécution de PRINT, une nouvelle ligne commence et le caractère W est affiché. Le point-virgule interdit tout retour à la ligne, aussi le curseur reste-t-il dans la position suivant la première lettre W. Le second W est alors affiché et le curseur va dans la position du caractère suivant. Cette séquence se poursuit jusqu'à la fin de la première ligne, à la suite de quoi le curseur va se positionner au début de la ligne suivant. La séquence continue sur les 20 lignes (pour un affichage sur 40 colonnes) (figure B ci-contre).

```
>240 PRINT "BOF!"
>250 END
>RUN
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
BOF!
```

Fig. A.

```
>NEW

>200 C$="W"
>210 FOR I=1 TO 800
>220 PRINT C$;
>230 NEXT I
>240 PRINT "BOF!"
>250 END

>RUN
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
BOF!
```

Fig. B.

Pourquoi BOF ! est-il affiché sur une nouvelle ligne ? En fait, BOF ! semble seulement être sur une nouvelle ligne mais en réalité, le dernier W est affiché en dernière position de la ligne précédente. Modifiez la boucle FOR-NEXT pour passer de 800 à 780 itérations et BOF ! viendra à la fin de la série de caractères, dans la dernière ligne :

```
>200 C$="W"
>210 FOR I=1 TO 780
>220 PRINT C$;
>230 NEXT I
>240 PRINT "BOF!"
>250 END

>RUN
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWBOF!
```

Le point-virgule commande la concaténation des chaînes de données en affichant les données les unes à la suite des autres, sans espacement intermédiaire. Les données numériques sont également affichées à la suite, sans espacement entre les valeurs contiguës.
Pour illustrer ce point, modifiez la variable chaîne et mettez à la place un seul digit numérique :

```
>200 C=5
>210 FOR I=1 TO 800
>220 PRINT C;
>230 NEXT I
>240 PRINT "BOF!"
>250 END
```

Le chiffre 5 est ainsi affiché :


```
>RUN
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
BOF!
>
```

Maintenant, changez C et mettez −1 :

```
>200 C=-1
>210 FOR I=1 TO 800
>220 PRINT C;
>230 NEXT I
>240 PRINT "BOF!"
>250 END

>RUN
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
```

```
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
BOF!
```

Que s'est-il produit ici ? Lorsque C est -1, le programme affiche autant de lignes que lorsque C était à 5. Il occupe une position supplémentaire sur l'écran pour afficher le signe moins.

Des nombres à plusieurs chiffres (y compris négatifs) vont faire se dérouler l'affichage, sauf si l'index de la boucle FOR-NEXT est bien ajusté. Si C passe à 2001, un champ de quatre digits est nécessaire ; aussi va-t-on ramener l'index de 800 à $800/4 = 200$:

```
>200 C=2001
>210 FOR I=1 TO 200
>220 PRINT C;
>230 NEXT I
>240 PRINT "BOF!"
>250 END
>RUN
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
```



```
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
BOF!
```

Les nombres seront coupés à la fin des lignes si nécessaire. Cela, parce que le point-virgule provoque un affichage à la suite et que rien, sauf la fin de la ligne, ne peut déclencher un retour à la suivante. Essayez, par exemple, $C = 201$ pour le vérifier.

Utilisation des virgules

Les virgules placées après une variable ou à la fin d'une instruction PRINT traitent l'affichage comme s'il était tabulé, divisé en colonnes. Les règles sont ici différentes en Integer Basic et en Applesoft.

L'Integer Basic établit cinq tabulations sur l'écran. Elles commencent en positions 1, 9, 17, 25 et 33 et sont séparées par 8 espaces. En voici un exemple en Integer Basic :

```
>NEW
> 200 C=123
> 210 FOR I=1 TO 100
> 220 PRINT C,
> 230 NEXT I
>240 PRINT "BOF!"
>250 END
>RUN
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
BOF!
```

Un espace (un blanc) doit exister au repère de début de chaque tabulation (sauf pour la première) faute de quoi cette tabulation serait inactive. Cela signifie que rien ne doit être affiché au repère de la position de départ de la tabulation commençant en colonne 8 pour qu'elle soit active, et ainsi de suite. C'est ce qu'illustre l'exemple suivant avec des chaînes :

```
>10 REM IL FAUT DIMENSIONNER LA VARIABLE CHAINE
>20 DIM C$(8)
>200 C$="12345678"
>210 FOR I=1 TO 20
>220 PRINT C$,
>230 NEXT I
>240 PRINT " BOF! "
>250 END
RUN
12345678        12345678        12345678
        12345678        12345678
12345678        12345678        12345678
        12345678        12345678
12345678        12345678        12345678
        12345678        12345678
12345678        12345678        12345678
        12345678        12345678
BOF!
```

Si vous attribuez à C$ la chaîne « 1234567 », les cinq tabulations seront utilisées.
Des conditions spéciales vont gouverner la tabulation avec des virgules en Applesoft, comme le montre la figure 4.1.
Pour observer l'action des virgules en Applesoft avec le même exemple de programme, remplacez le point-virgule de l'instruction PRINT par une virgule. L'affichage va se faire en trois colonnes. Avec trois nombres par ligne, l'index de FOR-NEXT sera de $3*20=60$.
Le programme complet est le suivant :

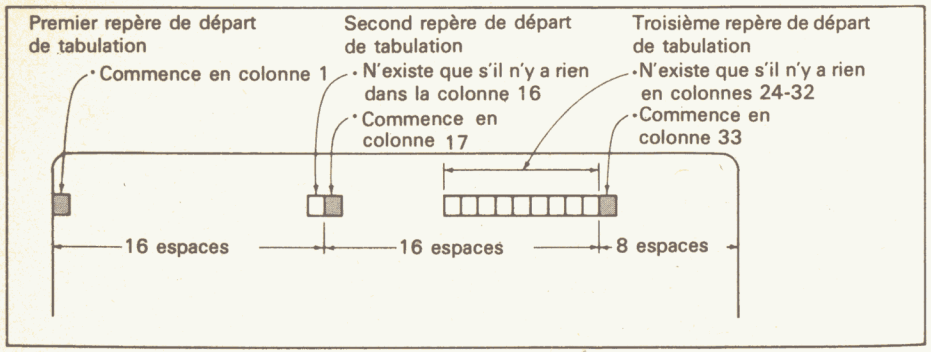


Fig. 4.1. — Les repères de début de tabulation en Applesoft, pour des instructions PRINT avec des virgules.

```
]200 C = 2001
]210 FOR I = 1 TO 60
]220 PRINT C,
]230 NEXT I
```



```
]240 PRINT "BOF!"
]250 END
RUN
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
BOF!
```

Les virgules fonctionnent également avec des chaînes. Comme exemple, entrez les ordres suivants affichant 20 lignes tabulées :

```
]NEW
]100 A$="DEUX"
]110 B$="BEAUX"
]120 C$="YEUX"
]210 FOR I=1 TO 20
]220 PRINT A$,B$,C$
]230 NEXT I
]240 PRINT "BOF!"
]250 END
]RUN
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
```

```
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
DEUX            BEAUX           YEUX
BOF!
```

Poursuivons en remplaçant B$ par « TRES BEAUX » au lieu de « BEAUX ». Si vous vous reportez à la figure 4.1, et si vous lancez ce nouveau programme, vous ne serez pas surpris si la troisième tabulation n'est pas employée. La chaîne « TRES BEAUX », commençant en 17, déborde dans les colonnes 24 à 32. Or, un affichage dans ces colonnes désactive la troisième tabulation.

FONCTIONS PRINT FORMATÉES

Nous utilisons le mot formaté pour décrire le processus d'organisation des informations sur l'écran de façon qu'elles soient plus faciles à comprendre et plus agréablement présentées. Ainsi qu'on l'a vu, les virgules et points-virgules y contribuent. Cependant, le formatage peut être plus complexe. Par exemple, supposez qu'on veuille afficher un titre au milieu de la ligne supérieure de l'écran.
Voici un programme en Integer Basic qui recourt à des virgules pour centrer le texte sur l'écran. Il emploie aussi l'instruction CALL, que nous avons découverte récemment, pour positionner le curseur à l'angle supérieur gauche de l'écran (voir appendice D).

```
>NEW

>5 REM DIMENSIONNONS LES CHAINES
>10 DIM HD$(30)
>500 REM EFFACONS L'ECRAN
>510 REM ET PLACONS LE CURSEUR
>520 REM A L'ANGLE SUPERIEUR GAUCHE
>530 CALL - 936
>540 HD$="TOP SECRET"
>550 REM TABULONS POUR CENTRER
>555 REM PUIS AFFICHONS LE TITRE
>560 PRINT "", "", HD$
>570 END
>RUN
                    TOP SECRET
```

Le titre est approximativement centré sur l'écran. S'il était plus long, par exemple « INSTRUCTIONS D'EMPLOI », il ne le serait plus (vérifiez-le). Ce dernier titre serait, lui aussi, à peu près centré si l'on supprimait l'une des virgules et sa chaîne vide associée, de façon que ce texte commence à la seconde tabulation et non plus à la troisième.

On pourrait se servir de la même méthode en Applesoft. Mais ici, on dispose d'instructions PRINT formatées plus faciles à utiliser : les fonctions SPC, TAB et POS.

Fonction SPC

La fonction SPC établit des espaces. On inclut SPC dans les termes d'un PRINT, et SPC est suivie par un certain nombre de positions de caractères (entre parenthèses) qui est le nombre d'espaces à sauter pour afficher la suite. Par exemple, et pour centrer l'en-tête TOP SECRET, on ferait :

```
]100 REM EFFACONS L'ECRAN

]110 HOME

]120 REM ESPACES ET AFFICHAGE

]130 PRINT SPC(15);"TOP SECRET"

]140 END

]RUN
                    TOP SECRET
```

Remarquez le point-virgule après SPC. Une virgule aurait provoqué une tabulation du texte suivant, après SPC.
Chaque fois que vous introduisez un SPC dans une instruction TAB, vous commanderez simplement un affichage au-delà du nombre de positions spécifiées après SPC ; rien d'autre ne change.

Fonction TAB

La fonction TAB agit à la façon d'une tabulation de machine à écrire.
Supposez que vous vouliez afficher ou imprimer des informations en colonne. Une forme telle que celle de l'appendice L est commode. Dans la figure 4.2, les colonnes commencent aux positions de caractères 1, 15 et 31. Maintenant, au lieu d'insérer des espaces d'une colonne à l'autre, on va simplement commander un TAB après chaque entrée dans l'instruction PRINT.
Examinez l'une des lignes affichées dans la figure 4.2. En comptant les positions des caractères, on pourrait afficher la ligne sans tabulation, de la façon suivante :

```
]10 ?"JONES,P.J          431-25-6277        1420.00"
```

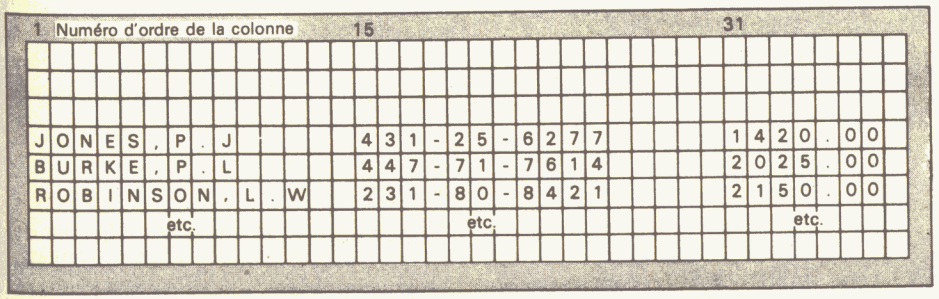


Fig. 4.2. — Pour déterminer la position des caractères sur l'écran.

Mais au lieu d'introduire des espaces, on pourrait utiliser l'instruction SPC et faire :

```
]10 ?"JONES,P.J";SPC(7);"431-25-6277";SP
C(5);"1420.00"
```

Mais une tabulation est encore plus simple parce qu'on tabule sur un numéro connu de colonne plutôt que de compter des espaces :

```
]10 ?"JONES,P.J";TAB(16);"431-25-6277";T
AB(32);"1420.00"
```

Pour déterminer la position du curseur (horizontalement)

POS est la dernière des fonctions formatage en Applesoft. Elle indique sur quelle colonne se trouve le curseur. Cette position est exprimée par un nombre, égal au nombre caractérisant la colonne sur laquelle le curseur clignote. Vous devrez toujours inclure un argument factice, 0, après POS, et écrire : POS(0).
L'instruction suivante montre l'action de POS.

```
]? "LA POSITION DU CURSEUR EST" ; POS (0)
```

Exécutez-la en mode immédiat. L'écran montrera :

```
]? "LA POSITION DU CURSEUR EST";POS (0)
LA POSITION DU CURSEUR EST 19
```

En effet, après avoir affiché LA POSITION DU CURSEUR EST, le curseur se trouve à la 27e position. Si vous introduisez des espaces après EST et avant de fermer les guillemets, le nombre 27 changera et sera remplacé par un nombre plus élevé.
En Integer Basic, vous pouvez simuler la fonction POS(0) avec PEEK(36) pour déterminer sur quelle colonne le curseur se trouve. Voici l'instruction correspondante :

```
>PRINT "LA POSITION DU CURSEUR EST ";PEEK (36)
LA POSITION DU CURSEUR EST 19
```

Remarquez que l'écran est divisé en colonnes numérotées de 0 à 39 pour POS, PEEK(36), et PRINT SPC. Les colonnes sont numérotées de 1 à 40 pour PRINT TAB et pour deux autres instructions que nous allons bientôt rencontrer, HTAB et TAB (Integer Basic).

Pour déterminer la position du curseur (verticalement)

L'ordre PEEK(37) vous donnera la rangée sur laquelle se trouve le curseur. Les rangées sont numérotées de 0 à 23 pour PEEK(37), mais de 1 à 24 pour VTAB (traitée dans la section suivante).

COMMANDE DU CURSEUR ET EFFETS VIDÉO SPÉCIAUX

Un certain nombre d'instructions Basic accroissent la souplesse d'emploi de l'écran de l'Apple II. Elles comprennent les FLASH, INVERSE, SPEED, NORMAL, HOME, VTAB et HTAB/TAB. La plupart ne sont disponibles qu'avec l'Applesoft. On trouvera aussi des instructions spécifiques au graphique, et on les examinera dans le chapitre 6.

Positionnement du curseur

Nous avons déjà indiqué plusieurs méthodes pour contrôler la position du curseur : virgules et points-virgules avec l'instruction PRINT. En Applesoft, les fonctions SPC, TAB et POS serviront aussi dans ce but.

Effacement de l'affichage

On peut effacer l'écran et repositionner le curseur à la position HOME (en haut et à gauche) avec l'instruction CALL-936 ou, en Applesoft, avec l'instruction HOME. Essayez ces instructions, l'une en Integer Basic et l'autre en Applesoft :

```
>CALL -936

]HOME
```

Positionnement horizontal et vertical

Deux instructions vous permettent de positionner le curseur sur l'écran ; VTAB déplace le curseur verticalement et HTAB (ou TAB en Integer Basic), horizontalement. Vous devez spécifier le numéro de ligne pour VTAB et celui de colonne pour HTAB. La première ligne de l'écran est la numéro 1, et la dernière, la 24. La colonne la plus à gauche est la numéro 1, la plus à droite est la 40.
Le programme suivant emploie ces deux instructions (en Applesoft) pour positionner le curseur et afficher un astérisque à cette position. Si vous utilisez l'Integer Basic, mettez TAB au lieu de HTAB aux lignes 90 et 120.

```
90  HTAB 1: VTAB (1)
100  INPUT "RANGEE?";R
110  INPUT "COLONNE?";C
120  VTAB R: HTAB C
130  PRINT "*";
140  GOTO 90
```

Instructions INVERSE et NORMAL

On peut obtenir une inversion vidéo sur l'écran (les noirs et les blancs sont inversés) à l'aide de l'instruction INVERSE. Lorsqu'elle est exécutée, tout ce qui est affiché par PRINT est inversé (en vidéo). Mais le caractère que vous frapperez au clavier sera affiché, en écho, en mode normal.
Apple II reviendra en mode vidéo normal lorsqu'il exécutera une instruction NORMAL.
Pour examiner le fonctionnement de ces commandes, essayez ce programme (les caractères ombrés apparaîtront en vidéo inversée) :

```
]INVERSE

]?"NOIR ET BLANC"
NOIR ET BLANC

]NORMAL

]?"BLANC ET NOIR"
BLANC ET NOIR
```

Les ordres INVERSE et NORMAL n'existent pas en Integer Basic.

L'instruction FLASH

Non seulement peut-on obtenir une inversion vidéo mais encore, on peut faire clignoter, flasher les caractères dans les deux modes, en vidéo normale ou inversée. Pour cela, utilisez l'instruction FLASH. A nouveau, NORMAL fera revenir l'Apple II à la vidéo normale. Les zones ombrées vont clignoter :

```
]FLASH
]?"ATTENTION"
ATTENTION
 .........
]NORMAL
 .........
]?"ATTENTION"
ATTENTION
```

L'instruction FLASH n'existe pas en Integer Basic.

L'instruction SPEED

Le rythme d'affichage des caractères sur l'écran est variable. Vous pouvez le ralentir, réduire sa vitesse normale, avec l'instruction SPEED (« vitesse »).
Le programme suivant montre comment SPEED agit :

```
]100 INPUT "VITESSE" = ";SP
]110  SPEED = SP
]120  FOR CT = 1 TO 3
]130 PRINT "HIP"
]140  NEXT
]150 PRINT "HOURRA"
]160  SPEED = 255
]170  END
```

La valeur de l'expression (la valeur de SP, à la ligne 110) ajuste la vitesse ; 0 est la plus basse et 255 la plus forte. L'ordre SPEED affecte également la vitesse de transmission des caractères envoyés à des périphériques autres que l'écran.
SPEED n'est pas disponible en Integer Basic.

FENÊTRES DE TEXTE

Normalement, Apple II travaille sur 24 lignes et 40 colonnes en mode texte de l'écran. Avec l'instruction POKE, on peut modifier cette *fenêtre de texte*. Quatre positions mémoires commandent les dimensions, la forme et la position, comme le montre le tableau 4.1.
Respectez soigneusement les gammes indiquées dans le tableau. Sinon, vous obtiendrez des résultats imprévisibles.

Le programme ci-dessous établit une fenêtre de deux lignes au milieu de l'écran pour introduire une valeur numérique. Pour apprécier l'utilité de cette technique, essayez d'entrer des valeurs non numériques et observez ce qui se passe avec les messages d'erreur. Notez aussi que l'établissement d'une fenêtre n'efface pas le reste de l'écran ni ne déplace le curseur dans la fenêtre. Vous devriez fournir des instructions Basic séparées pour cela.

Position mémoire	Commande	Gamme autorisée
32	Marge gauche	0 à 39
33	Largeur	1 à 40 moins la marge gauche
34	Ligne supérieure	0 à ligne inférieure
35	Ligne inférieure	24 moins ligne supérieure

Tableau 4.1

```
 .........
1000 REM ETABLISSEMENT D'UNE FENETRE DE TEXTE
1010 T = 10:W = 20:LM = 11:B = 13
1020 REM EFFACEMENT DE L'ECRAN
1030 CALL - 936
1040 REM ETABLISSEMENT FENETRE POUR UNE ENTREE
1050 GOSUB 3200
1060 REM ENCADRER LA FENETRE D'ASTERISQUES
1070 GOSUB 3000
1080 REM PLACER LE CURSEUR DANS LA FENETRE; FAIRE L'ENTREE
1090 VTAB T + 2
1100 INPUT M1
1110 REM RETABLIR LE PLEIN ECRAN
1120 GOSUB 3300
1130 REM PLACER LE CURSEUR SUR LIGNE DU BAS
1140 VTAB 23
1150 END
2990 REM ENCADRER LA FENETRE D'ASTERISQUES
3000 VTAB T + 1
3010 GOSUB 3100
3020 VTAB B + 1
3030 GOSUB 3100
3040 RETURN
3090 REM AFFICHAGE DES ASTERISQUES
3100 FOR I = 1 TO W
3110 PRINT "*";
3120 NEXT I
3130 RETURN
3190 REM ETABLISSEMENT FENETRE POUR ENTREE
3200 POKE 32,T
3210 POKE 33,W
3220 POKE 34,LM
3230 POKE 35,B
3240 RETURN
3290 REM  RETABLISSEMENT DU PLEIN ECRAN
```

```
3300 POKE 32,0
3310 POKE 33,40
3320 POKE 34,0
3330 POKE 35,24
3340 RETURN
```

LA FONCTION CHR$: PROGRAMMATION DE CARACTÈRES EN ASCII

Dans le chapitre précédent, nous avons discuté de la façon de créer des caractères invisibles, tels que CTRL-G qui déclenche un « bip » sonore. Mais il existe d'autres caractères (à la fois visibles et invisibles) que vous ne pouvez frapper directement au clavier ; ils comprennent les symboles « [» et « \ ». Or, vous pourrez les créer en Applesoft avec la fonction CHR$.

Pour bien comprendre cette fonction, vous devez comprendre comment les caractères sont stockés dans la mémoire de l'Apple II. En réalité, c'est très simple. La mémoire d'un ordinateur peut stocker des nombres, mais non des caractères. Aussi les caractères sont-ils convertis en codes numériques. L'Apple II recourt au même code que tous les autres micro-ordinateurs, le code ASCII (American Standard Code for Information Interchange). Par exemple, le code ASCII de la lettre A est 65, celui de B est 66, on a 67 pour C, etc. Vous trouverez un tableau complet des codes ASCII dans l'appendice I. Lorsque Apple II traite des chaînes, il interprète des valeurs numériques comme des codes ASCII pour des caractères.

En Applesoft, si vous ne pouvez presser une touche pour inclure un caractère dans une chaîne, vous pourrez toujours sélectionner ce caractère à l'aide de son code ASCII.

La fonction CHR$ traduit un code numérique ASCII dans son caractère équivalent. Par exemple, pour créer le symbole dollar, recherchez son code ASCII dans l'appendice I. Pour l'utiliser, faites :

```
]PRINT CHR$(36)
$
```

Expérimentez cette méthode en mode immédiat, avec des codes compris entre 0 et 255. Vous pouvez user de la fonction CHR$ avec des chaînes courantes intervenant dans un ordre PRINT :

```
]?CHR$(34);"BOF!";CHR$(34)
"BOF!"
```

La fonction CHR$ vous permet d'inclure dans une chaîne des caractères non disponibles autrement, par exemple un retour chariot, ou des guillemets.

PROGRAMMATION D'UNE ENTREE

L'objectif de tout programme devrait consister à minimiser les erreurs d'entrée de données, et faciliter à l'opérateur leur localisation et leur correction. Il existe des méthodes participant à ce but.

Tout d'abord, introduisez un bloc fonctionnel entier de données et *ensuite,* traitez-le. La méthode consistant à traiter chaque constituant de la donnée séparément, au fur et à mesure qu'il arrive, est peu satisfaisante.

Une liste d'envoi *(mailing list),* par exemple, requiert des noms et des adresses qui devront être introduites en tant que données. Vous devriez traiter chaque ensemble, nom et adresse, comme une unique entité fonctionnelle. En d'autres termes, votre programme devrait demander le nom et l'adresse, permettre à l'opérateur d'introduire toutes ces

informations, et d'en modifier tout ou une partie. Lorsque l'opérateur est satisfait du résultat, le programme pourrait se poursuivre et le traitement commencer. Le programme pourrait alors demander le nom et l'adresse suivants.

Dans ce cas, ce serait une mauvaise méthode que de demander le nom, de le traiter immédiatement, puis l'adresse, en la traitant séparément comme une entité fonctionnelle distincte.

Une bonne idée consiste à organiser l'entrée de données de façon qu'elles restent sur l'écran un moment encore après leur introduction. L'opérateur a alors l'opportunité de remarquer une erreur et de la corriger. Si l'entrée disparaît très vite, il ne lui restera aucune chance de remarquer une erreur éventuelle. Naturellement, l'opérateur devra pouvoir effectuer les corrections.

Dans certains cas, l'introduction de données en blocs fonctionnels, avec de grandes possibilités de révision et correction, n'est pas la meilleure des méthodes, ce qui peut paraître surprenant. Supposons, par exemple, qu'un opérateur frappe au clavier des centaines de noms et d'adresses par jour. L'expérience montre que le plus fort volume de données précises est introduit lorsque l'opérateur ignore les erreurs qu'il peut commettre. Le programme d'entrée ne doit pas permettre la correction des erreurs, même si l'opérateur les remarque aussitôt. L'opérateur doit les ignorer et poursuivre l'entrée des informations aussi vite que possible. Dans ce cas, les informations pourront être introduites deux fois, de préférence par deux opérateurs différents. Un programme séparé compare les deux entrées. Les risques pour que les deux opérateurs aient commis la même erreur sont si faibles que vous pourrez être certain que toutes les erreurs se manifesteront par des entrées différentes. Un programme ad hoc servira à ré-entrer la donnée incorrecte.

Entrée interactive de données

Un programme comportant une entrée interactive de données guide l'utilisateur en affichant des instructions et en présentant des demandes de réponses. Voici un exemple simple qui le démontre. Nous allons modifier un programme antérieur de façon qu'il procède en mode interactif, afin de faciliter l'entrée.

Voici ce programme :

```
200 C$ = "W"
210  FOR I = 1 TO 800
220  PRINT C$;
230  NEXT I
240  PRINT "BOF!"
250  END
```

Ce programme affiche 800 caractères W, suivis par le mot BOF ! Il fonctionne aussi bien en Applesoft qu'en Integer Basic.

Supposons qu'on veuille afficher un autre caractère quelconque, et non W.

Dans ce cas, on va éliminer l'instruction d'affectation ; rappelez-vous que pour supprimer une instruction, il suffit de frapper son numéro de ligne, puis de commander RETURN.

```
210  FOR I = 1 TO 800
220  PRINT C$;
230  NEXT I
240  PRINT "BOF!"
250  END
```

La ligne 200 n'est plus dans le programme.

Introduisez la commande C$ = « X » en mode immédiat, et lancez le programme :

```
]C$="X"

]RUN
BOF!
```

Le mot BOF! est affiché, mais pas les X. A l'évidence, la valeur de C$ n'a pas été transmise au programme.

L'ordre RUN met à zéro toutes les variables numériques et annule toutes les chaînes avant l'exécution d'un programme. Aussi, C$ a été annulé et rien n'a été affiché (mal ou rien sont ici synonymes : le programme n'affiche rien et ne déplace pas le curseur).

Existe-t-il une méthode permettant de transférer une valeur différente de C$ chaque fois qu'on lance le programme ? Au lieu de faire RUN, qui initialise les variables, faites GOTO 210 (où 210 est le numéro de la première ligne du programme). Cela ne modifiera plus les valeurs des variables :

```
]C$="X"

]GOTO 210
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
BOF!
```

Le lancement du programme demande deux étapes, et la procédure est détestable. Vous devez, en effet, frapper une ligne (pour l'affectation), et si vous faites ensuite RUN, vous devrez tout recommencer. Mais le programme pourrait acquérir le caractère affiché alors qu'il est en train de tourner avec une instruction INPUT.

Frappez la ligne suivante :

```
]200 INPUT C$
```

Listez le programme pour vérifier que vous avez correctement introduit cette ligne :


```
]LIST

200  INPUT C$
210  FOR I = 1 TO 800
220  PRINT C$;
230  NEXT I
240 PRINT "BOF!"
250 END
```

Lancez ce programme. Le curseur passe une ligne. (En Applesoft, un point d'interrogation apparaît à côté du curseur.) Introduisez un *unique* caractère, et pressez RETURN. Le caractère sera affiché 800 fois. Relancez le programme. L'affichage se fera avec un nouveau caractère. Notez que si vous introduisiez plus d'un caractère dans ce programme, en Integer Basic, vous obtiendriez un message d'erreur ***STR OVFL ERR. Cela, parce que C$ n'a pas été dimensionné.

Il s'agit là d'une amélioration réelle du programme original. Cependant, c'est assez déconcertant de voir le curseur clignoter sur l'écran alors qu'il attend que vous pressiez une touche. Ajoutez une ligne explicative au début du programme, affichant une demande pour l'action attendue. Faites :

```
]190 PRINT "FRAPPEZ UN CARACTERE"
```

Listez le programme et vérifiez qu'il ne comporte pas d'erreurs.

Maintenant, c'est le programme qui donne des instructions à l'opérateur. Faites-le tourner plusieurs fois avec divers caractères et appréciez sa facilité d'utilisation.

Il reste une modification importante à accomplir. Si vous souhaitez que le programme se répète automatiquement, introduisez une instruction GOTO pour, de la fin, revenir à son début. Dès lors, vous n'aurez plus à frapper RUN pour le ré-exécuter. Ajoutez la ligne suivante :

```
]250 GOTO 190
```

L'usage du programme en est encore simplifié. Faites RUN et lancez-le. Bien sûr, pour terminer, vous aurez à frapper CTRL-C. Cette obligation peut être éliminée en programmant une touche de fin spéciale ; ce pourra être la touche RETURN. Voici comment procéder :

```
190 PRINT "FRAPPEZ UN CARACTERE"
200  INPUT C$
205  IF C$ = "" THEN  END
210  FOR I = 1 TO 800
220  PRINT C$;
230  NEXT I
240 PRINT "BOF!"
250 GOTO 190
```

La ligne 205 teste C$ afin de vérifier si sa valeur est nulle. Cela ne surviendra que si vous ne frappez pas de caractère en réponse à l'INPUT de la ligne 200 et si vous faites directement RETURN.

Enfin, vous pourriez relire votre programme et ajouter des remarques. Incluez un commentaire sur la façon dont l'index de la boucle FOR-NEXT a été calculé ; vous pourriez l'écrire sur la même ligne, après un double point séparant les ordres :

```
210 FOR I=1 TO 800:REM 800/40=20 lignes
```

Ajoutez, en pense-bête, qu'une entrée nulle termine le programme :

```
203 REM FIN DE PROGRAMME SUR ENTREE NULLE
```

Pour terminer, ajoutez aussi quelques lignes au début du programme décrivant ce qu'il fait et lui donnant un titre, comme le montre la figure 4.3.

Messages d'interrogation

Tout programme demandant des entrées de données devrait émettre un message d'interrogation destiné à l'opérateur. Les questions apparaissent généralement sur une seule ligne et proposent une réponse simple, du genre « oui » ou « non ». Par exemple, le message suivant pourrait être affiché :

VOULEZ-VOUS INTRODUIRE DES MODIFICATIONS ?

L'opérateur devra répondre par oui ou par non. Souvent, les lettres O et N suffiront. Un autre cas consiste à laisser l'opérateur choisir entre diverses options :

QUELLE ENTREE VOULEZ-VOUS MODIFIER ?

On permettra alors à l'opérateur de frapper un code identifiant l'entrée.
Les programmes contrôlant ce type de dialogue devraient être rédigés séparément, en sous-programmes, lesquels ne dépendraient absolument pas du programme qui les appelle, et en resteraient totalement indépendants. Ce qui implique trois choses :

1. Vous ne pourrez supposer que l'espace où le message d'appel, d'interrogation, va apparaître, est disponible. S'il n'est pas blanc, ce message recouvrira ce qui pré-existait. Mais les caractères subsistants ne seront pas effacés ; le tout pourra se mélanger, ce qui tendrait à créer une confusion et pourrait mener à des erreurs. Le sous-programme suivant effacera le nombre d'espaces déterminé par la valeur de la variable ER :

```
5000 REM POUR EFFACER DU TEXTE
5010 FOR I=1 TO ER : PRINT "";:NEXT I: RETURN
```

2. Un sous-programme doit recevoir des informations du programme appelant. Par exemple, si un sous-programme demande à l'opérateur d'introduire un nombre, c'est le programme l'exploitant qui devra spécifier le minimum et le maximum acceptables pour ce nombre.
3. Le sous-programme doit retourner la réponse de l'opérateur au programme appelant. Ce pourra être un caractère (par exemple, O ou N), un mot (par exemple, OUI ou NON), ou un nombre.

La logique du sous-programme ne peut savoir où elle doit afficher le message sur l'écran. Il est alors courtois de demander au programme appelant qu'il positionne correctement le curseur avant d'appeler le sous-programme.


```
 10 REM ****** CARPETTE ******
 20 REM AFFICHAGE D'UNE LIGNE CONTINUE
 30 REM D'UN CARACTERE ENTRE
 40 REM AU CLAVIER
 50 REM ********************
190 PRINT " FRAPPEZ UN CARACTERE "
200 INPUT C$
203 REM FIN DE PROGRAMME SUR ENTREEE NULLE
205 IF C$="" THEN END
210 FOR I=1 TO 800 : REM 800/40=20 LIGNES
220 PRINT C$ ;
230 NEXT I
240 PRINT " BOF! "
250 GOTO 190
```

Fig. 4.3. — Le programme « Carpette » : on déroule une carpette de lettres.

Examinez, maintenant, le sous-programme nécessaire pour poser une question demandant une réponse par O pour « oui » et N pour « non ». Nous allons utiliser une instruction PRINT pour poser la question, suivie par un INPUT attendant une réponse sur un caractère. Nous allons effacer une partie de l'écran avec le sous-programme ci-dessus. Voici ce que cela donne :

```
100 REM DEPLACEMENT DU CURSEUR
140 VTAB 1
150 REM APPEL SOUS-PROGRAMME POUR REPONSE
160 GOSUB 3020
170 END
3000 REM ++ACQUISITION DE LA REPONSE O/N++
3010 REM
3020 C = POS (0): REM ATTENTION A LA COLONNE DU CURSEUR
3030 R = PEEK (37): REM ATTENTION A LA RANGEE DU CURSEUR
3040 REM EFFACER POUR LE DIALOGUE
3050 ER = 35: REM EFFACER 35 POSITIONS
3060 GOSUB 5010
3070 HTAB C + 1: REM REPOSITION DU CURSEUR
3080 PRINT "VOULEZ-VOUS FAIRE UNE MODIFICATION?";
3090 INPUT R$
3100 IF R$ = "O" OR R$ = "N" THEN RETURN
3110 REM REPONSE INCORRECTE
3120 VTAB R + 1: REM REPOSITION (VERT) DU CURSEUR
3130 GOTO 3050: REM ESSAYEZ A NOUVEAU
5000 REM ++CREATION D'ESPACES++
5010 FOR I = 1 TO ER: PRINT " "; : NEXT I: RETURN
```

Maintenant, considérez le dialogue qui va permettre à l'opérateur d'introduire un nombre. Nous allons concevoir un sous-programme n'acceptant une réponse que si elle n'est pas inférieure à la variable LO et pas supérieure à la variable HI. Ce sous-programme retournera le nombre entier dans la variable NM. Voici ce programme :

```
100 REM POUR ETABLIR GAMME ET POSITION CURSEUR
140 LO = 1:HI = 10
150 VTAB 1
160 REM APPEL SOUS-PROGRAMME POUR REPONSE
170 GOSUB 3500
180 END
3500 REM ++ACQUISITION DU NOMBRE++
3510 REM ++ LO<=REPONSE<=HI++
3520 REM ++ LA REPONSE EST NM++
3530 GOSUB 5110: REM OU EST LE CURSEUR MAINTENANT?
3540 REM EFFACER POUR PLACER LA QUESTION
3550 ER = 35: REM EFFACER 35 POSITIONS
3560 GOSUB 5010
3570 HTAB C + 1: REM REPOSITIONNONS LE CURSEUR
3580 PRINT "QUELLE ZONE VOULEZ-VOUS MODIFIER? (1-10)";
3590 INPUT NM
3600 IF NM >.= LO AND NM < = HI THEN RETURN
3610 REM MAUVAISE REPONSE
3620 VTAB R + 1: REM REPOSITION CURSEUR (VERT)
3630 GOTO 3550: REM ESSAYEZ A NOUVEAU
5000 REM ++SUPPRIMER LES ESPACES++
5010 FOR I = 1 TO ER: PRINT "";: NEXT I: RETURN
5100 REM ++ POSITION COURANTE DU CURSEUR ++
5110 C = PEEK (36): REM COLONNE
5120 R = PEEK (37): REM RANGEE
5130 RETURN
```

Pourriez-vous modifier ce sous-programme pour qu'il accepte une entrée sur deux digits ? Essayez de le rédiger par vous-même. Si vous ne réussissez pas, avancez dans ce chapitre ; vous y trouverez un sous-programme commandant l'entrée d'une date.
Vous pourriez exécuter une autre modification simple portant sur les dialogues décrits. Le message d'appel affiché pour les deux programmes pourrait être fourni par le programme appelant, via une variable chaîne. Le sous-programme deviendrait plus général. Pourriez-vous ré-écrire ce programme ainsi modifié ?

Détection d'erreur et contrôle

Si vous voulez rédiger un programme bien étudié, il vous faudra anticiper les erreurs que l'utilisateur du programme pourra commettre. Votre programme détectera les erreurs d'entrées et obligera l'utilisateur à ré-enter les valeurs qui provoqueraient une halte anormale.
Il est vrai que l'Apple II détecte déjà certaines erreurs pour vous. Il n'acceptera pas une entrée alphabétique s'il attend une valeur numérique avec une instruction du type INPUT A. Si vous tentiez d'introduire des lettres en réponse à une telle instruction, Apple II vous retournerait un message d'erreur et vous demanderait de recommencer votre entrée.
Les tests d'erreurs incorporés ont cependant des capacités limitées. Il est possible d'introduire avec un bon type de donnée (par exemple, numérique, ou chaîne) une donnée inacceptable, c'est-à-dire une donnée qui pourra provoquer une erreur ultérieure. En voici un exemple :


```
100  INPUT X
200  PRINT 100 / X
300  END
```

Si vous répondez par zéro, à INPUT X, le programme va être piégé car il essaiera de diviser 100 par zéro, à la ligne PRINT. Cette erreur est facile à éviter. Les lignes suivantes vont tester l'entrée et vérifier qu'il ne s'agit pas d'un zéro, dans lequel cas une autre entrée sera demandée :

```
110  IF X <  > 0 THEN 200
120 PRINT "INTERDIT!... RECOMMENCEZ"
130 GOTO 100
```

Ce principe peut être développé ; vous constatez qu'il est alors facile de tester une entrée afin de vérifier qu'elle est correcte. En fonction des circonstances, il pourra être intéressant de tester des gammes, en Applesoft, avec les instructions ON-GOTO ou ON-GOSUB (et en Integer Basic, avec GOTO et GOSUB calculés), plutôt qu'avec les instructions IF-THEN. La notion suivante présente un exemple de programme avec test plus extensif des erreurs.

Les instructions ONERR GOTO et RESUME

Applesoft dispose d'une instruction spéciale pour les erreurs détectées, intervenant avant l'affichage de tout message d'erreur ou avant tout blocage du programme. En voici un exemple :

```
50  ONERR  GOTO 8000
```

Lorsque cette instruction sera exécutée, Applesoft branchera le programme à la ligne 8000 s'il a décelé une erreur. Il placera aussi un code numérique décrivant l'erreur à la posiiton mémoire 222, que vous pourrez inspecter avec l'instruction PEEK 222. Le tableau C.1 de l'appendice C liste les erreurs détectables par ONERR GOTO.
La procédure usuelle de traitement des erreurs avec ONERR GOTO consiste à écrire un programme de traitement de l'erreur auquel ONERR GOTO va se brancher lorsqu'une erreur survient. A la fin de ce programme, l'instruction RESUME commande un retour au début de l'instruction où l'erreur est survenue. Ou encore, GOTO transférera la suite à n'importe quelle ligne du programme. Il faut rédiger le programme de traitement d'erreur de façon qu'il puisse entreprendre les actions nécessaires en fonction de l'erreur et de l'état du programme, ce qui est généralement déterminé par l'inspection des valeurs des variables-clés.
Pour annuler l'effet de ONERR GOTO et restituer à Apple II son mode normal de traitement automatique d'erreurs, utilisez l'instruction POKE 216,0.
Le programme suivant fait la démonstration de ONERR GOTO. Dans ce programme, toute erreur ne provenant pas d'une entrée maladroite au clavier est traitée comme une erreur fatale, avec le message approprié. Les erreurs d'entrées sont annoncées et suivies d'une demande d'entrée à refaire.

```
50 ONERR GOTO 8000
200 PRINT "ENTREZ UNE CHAINE"
210 INPUT X$
220 PRINT "ENTREZ UNE VALEUR NUMERIQUE"
230 INPUT X
```

```
240 PRINT "ENTREZ UN ENTIER"
250 INPUT X%
260 GOTO 200
500 REM FIN DU PROGRAMME
510 PRINT "LES DERNIERES ENTREES ETAIENT ";X$;", ";X;"ET";X%
515 POKE 216,0: REM RETOUR A TRAITEMENT AUTOMATIQUE
520 END
8000 REM ++ PROGRAMME DE TRAITEMENT D'ERREURS ++
8010 E = PEEK (222): REM ACQUISITION TYPE DE L'ERREUR
8020 IF E = 255 THEN GOTO 500: REM FIN PROGRAMME PAR CTRL-C
8030 IF E = 53 OR E = 176 OR E = 254 THEN 8100
8035 INVERSE
8040 REM ERREUR DETECTEE
8050 PRINT "AH! ERREUR NO. " ;E; " TROUVEE "
8055 PRINT "ECRIVEZ CE NOMBRE"
8060 PRINT "ET LA DESCRIPTION DU PROCESSUS"
8070 PRINT "APPELEZ UN PROGRAMMEUR A L'AIDE"
8080 PRINT "ABANDONNEZ L'ORDINATEUR"
8090 NORMAL : STOP
8100 REM ERREUR D'ENTREE DETECTEE
8110 PRINT "";: REM CARACTERES DE CTRL-G ENTRE GUILLEMETS
8130 PRINT "ERREUR... RECOMMENCEZ"
8140 RESUME
```

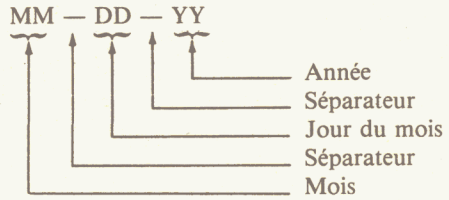
Pour entrer la date

Dans cette section, nous allons développer un programme employant plusieurs des techniques présentées ci-dessus. Ce programme recourt à certaines fonctions Basic disponibles seulement en Applesoft. Il peut être rédigé en Integer Basic ; vous pourriez effectuer la conversion.

La plupart des programmes demandent, à un moment quelconque, l'introduction de données relativement simples : peut-être plus qu'un simple oui ou non, mais beaucoup moins qu'un plein écran. Considérons le cas d'une date.

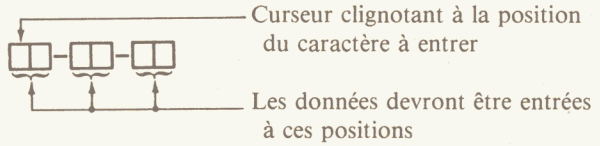
Vous devrez prêter davantage d'attention à ce genre de programme qu'il n'y paraît de prime abord. Selon toutes probabilités, la date sera juste une entrée parmi d'autres. En concevant soigneusement l'entrée de la date, chaque fois que nécessaire, vous vous éviterez bien des ennuis si l'opérateur s'embrouille dans les séquences d'entrées.

Nous supposerons que la date doit ainsi être entrée (à l'américaine, c'est-à-dire comme elle apparaît sur la plupart des montres électroniques).



Le mois, le jour du mois et l'année sont introduits chacun sur deux digits, sans RETURN. Le programme fournit les tirets séparant les entrées. Selon vos préférences, vous pourrez placer une barre oblique, ou tout autre caractère qu'il vous plaira.

Programmez cette entrée de façon qu'elle soit agréable à l'opérateur. Celui-ci devrait immédiatement saisir où la date doit être entrée, comment elle doit être présentée, et où on en est dans le traitement des données. Une bonne façon de faire, pour montrer où entrer la date, consiste à inverser le champ d'entrée. Par exemple, le programme demandant la date pourra créer une zone en vidéo inversée sur l'écran :



Pour créer un tel affichage, voici le programme à appliquer :

```
10 HOME : VTAB 3: HTAB 20: REM POSITION POUR L'ENTREE
20 IW = 2: GOSUB 1100: REM CHAMP D'ENTREE SUR 2 CARACTERES
30 PRINT "-";
40 GOSUB 1100: REM CHAMP D'ENTREE SUR 2 CARACTERES
50 PRINT "-";
60 GOSUB 1100: REM CHAMP D'ENTREE SUR 2 CARACTERES
70 VTAB 3: HTAB 20: REM POSITION POUR REDEMARRER AU DEBUT
80 END
1090 REM ++ AFFICHAGE "IW" EN INVERSE
1100 INVERSE
1110 FOR I = 1 TO IW: PRINT " ";: NEXT I
1120 NORMAL
1130 RETURN
```

Le programme ci-dessus inclut des instructions positionnant l'entrée de la date au début de la colonne 21, sur la rangée 3. Il efface aussi l'écran afin qu'aucun texte ne détourne l'attention. Après que l'entrée de la date ait été affichée, le curseur revient à la position du premier caractère, du premier champ, bien que ce ne soit pas évident en raison de l'instruction END.

Essayez d'employer un INPUT à la ligne 80 pour obtenir la première partie de la date : le mois. On pourra procéder ainsi :

```
80  INPUT M$
90  END
```

Introduisez ces deux lignes, 80 et 90, et exécutez-les. Comme vous pourrez le constater, l'instruction INPUT ne conviendra pas. En dehors du fait que le point d'interrogation d'appel déplace le champ inversé, l'action sur RETURN à la fin de l'entrée effacera le reste de la ligne d'affichage. Voici donc l'occasion d'utiliser l'instruction GET. Ajoutez les lignes suivantes :

```
80 GOSUB 1200:MM$ = C$: REM PREMIER CHIFFRE DU MOIS
90 GOSUB 1200:MM$ = MM$ + C$: REM DEUXIEME CHIFFRE DU MOIS
1190 REM ++ UN CARACTERE ACCEPTE ++
1200 GET C$
1210 PRINT C$;: REM FRAPPE AFFICHEE EN ECHO
1220 RETURN
```

Ces instructions acceptent une entrée sur deux digits. L'entrée est affichée dans le premier

champ inversé de la date. Ces deux digits viennent d'une entrée sans RETURN ni aucune autre frappe de terminaison. Le programme termine automatiquement l'entrée de la date après que les deux chiffres aient été frappés.
Mais il faut introduire trois paires de digits : mois, jour, année. Plutôt que de répéter les instructions des lignes 80 et 90, on va les introduire dans un sous-programme qu'on appellera trois fois :

```
80 GOSUB 1300:MM$ = CC$: REM OBTENTION DU MOIS
90 PRINT "-";: REM ESPACE POUR CHAMP SUIVANT
180 GOSUB 1300:DD$ = CC$: REM OBTENTION DU JOUR
190 PRINT "-";: REM ESPACE POUR CHAMP SUIVANT
280 GOSUB 1300:YY$ = CC$: REM OBTENTION ANNEE
290 END
1290 REM ++ POUR UNE ENTREE SUR 2 CARACTERES ++
1300 GOSUB 1200: CC$ = C$: REM APPEL 1ER CARACTERE
1310 GOSUB 1200: CC$ = CC$ + C$: REM APPEL 2EME CARACTERE
1320 RETURN
```

Les variables MM$, DD$ et YY$ représentent les mois, jour et année respectivement. Chacune se compose d'une chaîne de deux caractères.
Deux façons permettent à l'opérateur de corriger les données en entrant la date :

1. Le programme peut automatiquement tester la validité des mois, jour et année.
2. L'opérateur disposera d'un moyen de recommencer l'entrée.

Le programme peut tester si le mois est compris entre 1 et 12. Il ne s'occupera pas des années bissextiles, mais vérifiera les jours en tenant compte du mois spécifié. Toutes les années, de 00 à 99, seront autorisées. Toute entrée invalide déclenchera une nouvelle séquence complète d'entrée de la date.
Si l'opérateur presse la touche RETURN, la séquence complète redémarrera aussi. Par conséquent, le programme final est le suivant :

```
10 HOME : VTAB 3: HTAB 20: REM POSITION POUR ENTREE
15 RC$ = CHR$ (13): REM CARACTERE DE REDEMARRAGE
17 REM
18 REM AFFICHAGE 3 CHAMPS ENTREES 2 CARACTERES
19 REM
20 IW = 2: GOSUB 1100
30 PRINT "-";
40 GOSUB 1100
50 PRINT "-";
60 GOSUB 1100
65 PRINT "";: REM CTRL-G (SONNETTE)
67 REM
68 REM APPEL 3 CHAMPS D'ENTREES
69 REM
70 VTAB 3: HTAB 20: REM POSITION POUR REDEMARRER AU DEBUT
79 REM
80 REM OBTENTION DU MOIS
81 REM
90 GOSUB 1300: IF C$ = RC$ THEN GOTO 10: REM
TEST POUR RENOUVELLEMENT ENTREE
100 MM = VAL (CC$): REM NOMBRE DU MOIS
```



```
110 IF MM < 1 OR MM > 12 THEN 10: REM TEST POUR MOIS
120 DT$ = CC$: PRINT "-";: REM CHAMP SUIVANT
125 REM COMBIEN LE MOIS A-T-IL DE JOURS?
130 DM = 31: REM SUPPOSONS 31 JOURS
135 REM SAUF FEVRIER
140 IF MM = 2 THEN DM = 29
150 REM OU, AVRIL, JUIN, SEPT, NOV
160 IF MM = 4 OR MM = 6 OR MM = 9 OR MM = 11 THEN DM = 30
169 REM
170 REM OBTENTION JOUR
171 REM
180 GOSUB 1300: IF C$ = RC$ THEN GOTO 10: REM TEST POUR
RENOUVELLEMENT ENTREE
190 DD = VAL (CC$): IF DD < 1 OR DD > DM THEN 10: REM
RECOMMENCER SI ENTREE INVALIDE
200 DT$ = DT$ + "-" + CC$: PRINT "-";: REM CHAMP SUIVANT
269 REM
270 REM OBTENTION ANNEE
271 REM
280 GOSUB 1300: IF C$ = RC$ THEN GOTO 10: REM TEST POUR
RENOUVELLEMENT ENTREE
290 YY = VAL (CC$): IF YY < 0 OR YY > 99 THEN 10: REM
RECOMMENCER SI ENTREE INVALIDE
300 DT$ = DT$ + "-" + CC$
389 REM
390 REM AFFICHAGE ENTREE
391 REM
400 VTAB (10): HTAB (18): PRINT "DATE INTRODUITE:"
410 VTAB (11): HTAB (20): PRINT DT$
420 END
1089 REM
1090 REM ++ AFFICHAGE 'IW' EN INVERSE ++
1091 REM
1100 INVERSE
1110 FOR I = 1 TO IW: PRINT " "; NEXT I
1120 NORMAL
1130 RETURN
1189 REM
1190 REM ++ ENTREE 1 CARACTERE ACCEPTEE ++
1191 REM
1200 GET C$: IF C$= " " THEN 1200
1210 IF C$ = RC$ THEN RETURN: REM TEST POUR REDEMARRAGE
1220 IF C$ < "0" OR C$ > "9" THEN GOTO 1200
1230 PRINT C$;: REM ECHO DE LA FRAPPE
1240 RETURN
1289 REM
1290 REM ++ APPEL ENTREE 2 CARACTERES ++
1291 REM
1300 REM 1ER CARACTERE; TEST POUR REDEMARRAGE
1310 GOSUB 1200: IF C$ = RC$ THEN RETURN
1315 CC$ = C$
1320 REM 2EME CARACTERE; TEST POUR REDEMARRAGE
```

```
1330 GOSUB 1200: IF C$ = RC$ THEN RETURN
1335 CC$ = CC$ + C$
1340 RETURN
```

Remarquez que la date est constituée par la chaîne DT$ de huit caractères, avec le mois, le jour et l'année.
Ces trois tests sont effectués sur l'entrée :

1. Le caractère est-il un retour chariot ?
2. Sinon, est-ce un digit valide ?
3. Les deux caractères entrés correspondent-ils à un mois, un jour ou une année valides ?

Nous avons sélectionné comme caractère commandant le recommencement de l'entrée, le retour chariot. En remplaçant CHR$(13) de la ligne 15, vous pouvez adopter tout autre caractère de redémarrage. Lorsque l'opérateur presse ce caractère, la séquence complète d'entrée recommence. Nous devons tester s'il s'agit du caractère de redémarrage avec le sous-programme d'entrée un caractère (ligne 1200), puis à nouveau avec le sous programme d'entrée deux caractères (ligne 1300), si nous voulons être en mesure de redémarrer après l'entrée du premier ou du second caractère. Le programme principal recherche aussi le caractère de redépart de façon à brancher à la ligne 10 et à recommencer la séquence entière. Vous pourriez faire ce branchement directement depuis le sous-programme d'entrée un caractère vers la ligne 10, éliminant ainsi les autres tests. Mais c'est une mauvaise habitude que de sortir d'un sous-programme avec une instruction GOTO au lieu d'un RETURN. Chaque sous-programme devrait être traité comme un module logique, spécifiant le, ou les points d'entrée, et le retour standard. Si l'on se branche sur une autre voie pour sortir d'un sous-programme, on s'embrouillera immanquablement et des erreurs surviendront. (Rappelez-vous que si vous sortez d'un sous-programme sans user d'un RETURN, vous devrez annuler la position de retour avec une instruction POP.)
La logique du programme testant les caractères pour savoir s'ils sont autres que des chiffres tient dans le sous-programme d'entrée un caractère. Nous avons choisi d'ignorer les caractères non numériques. L'instruction de la ligne 1220 recherche ces caractères non numériques.

La logique de test pour des mois, jours et années valides doit exister dans le programme appelant (et non le sous-programme) car chacune des paires de caractères possède des limites différentes.
La ligne 110 vérifie si le mois est valide.
Les lignes 130, 140 et 160 vérifient les limites pour les jours, la ligne 190 testant alors si l'entrée du jour est valide.
Le test pour l'année est simple : il se fait à la ligne 290.
Cela prend davantage de temps d'écrire un bon programme d'entrée qui affiche l'information de manière agréable, teste les entrées, et permet à l'opérateur de recommencer s'il s'est trompé. Cela en vaut-il la peine ? Résolument oui. Vous n'écrivez ce programme qu'une seule fois ; l'opérateur, lui, pourra avoir à l'exécuter des centaines ou des milliers de fois. De ce fait, vous ne passerez davantage de temps qu'une fois pour éviter des centaines ou des milliers de pertes de temps à l'opérateur.

ENTRÉES SOUS FORME DE QUESTIONNAIRE

La section suivante décrit des techniques de programmation davantage adaptées à l'Applesoft. La plupart des effets spéciaux utilisés ici sont difficiles sinon impossibles à obtenir en Integer Basic. Si vous programmez exclusivement en Integer Basic, lisez quand même cette section car certaines des techniques proposées sont adaptables à l'Integer Basic.
La meilleure façon de traiter une liste d'entrées consiste à la présenter sous forme d'un formulaire, d'un questionnaire à remplir ensuite. Considérez un nom et une adresse. Affichez d'abord le questionnaire suivant :

```
DONNEZ LE NOM ET L'ADRESSE
1 NOM :
2 RUE :
3 VILLE:
4 PAYS :
5 CODE POSTAL :
```

Chaque entrée, remarquez-le, se voit affecter un numéro. Ces numéros apparaissent en vidéo inversée.
L'opérateur introduit les données séquentiellement, en commençant par 1 et en terminant par 5. Il peut alors modifier n'importe quelle entrée.
Le programme suivant efface l'écran et affiche ce questionnaire :

```
109 REM
110 REM AFFICHAGE DU FORMULAIRE
111 REM
120 CALL - 936: VTAB 2: REM EFFACER L'ECRAN ET
    POSITIONNEZ LE CURSEUR
125 PRINT "ENTREZ LE NOM ET L'ADRESSE"
130 REM AFFICHAGE NUMEROS DES CHAMPS
140 INVERSE
150 FOR I = 1 TO 4: HTAB 2: PRINT I: NEXT I
160 VTAB 6: HTAB 29: PRINT 5
170 NORMAL
180 REM AFFICHAGE NOMS DES CHAMPS
190 VTAB 3: HTAB 6: PRINT "NOM:"
200 HTAB 4: PRINT "RUE:"
210 HTAB 6: PRINT "VILLE:"
220 HTAB 5: PRINT "PAYS:"
230 HTAB 31: PRINT "CODE POSTAL:"
```

Au fur et à mesure des entrées, nous allons créer un champ inversé pour identifier la positin où la donnée sera affichée. CTRL-X sert à recommencer l'entrée dans le champ courant. La séquence suivante pourvoit à cette logique :

```
100 RC$ = CHR$ (24): REM CTRL-X SERT A REDEMARRER
299 REM
300 REM ENTRER LES 5 CHAMPS
301 REM
310 FOR F = 1 TO 5: GOSUB 1900: NEXT F
320 END
990 REM ++++++++++SOUS-PROGRAMME 1000++++++++++
991 REM ENTREE CHAINE DANS CHAMP DE LN CARACTERES
```

```
992 REM LE CURSEUR DOIT ETRE AU DEBUT
993 REM RETURN TERMINERA L'ENTREE
994 REM LA FLECHE GAUCHE LA FERA REDEMARRER
995 REM PAS DE TESTS DE VALIDITE
996 REM LA CHAINE ENTREE EST RETOURNEE DANS CC$
997 REM
1000 HT = POS (10) + 1: REM POSITION DE COMMENCEMENT
     DU CHAMP
1010 REM AFFICHAGE VIDEO INVERSEE
1020 INVERSE
1030 FOR I = 1 TO LN: PRINT " ";: NEXT I
1040 NORMAL : HTAB (HT): REM REPOSITION DEPART CHAMP
1050 REM ENTREE DE DONNEES
1060 CC$ = "": REM INITIALISATION SORTIE A NUL
1070 GET C$
1080 IF C$ = RC$ THEN HTAB (HT): GOTO 1020: REM
     RECOMMENCER L'ENTREE?
1090 IF C$ = CHR$ (13) THEN GOTO 1140: REM FIN ENTREE?
1100 REM SI ENTREE TERMINEE ATTENDRE RETURN OU REDEMARRAGE
1110 IF LEN (CC$) = LN GOTO 1070
1120 PRINT C$;: REM ECHO DE LA FRAPPE
1130 CC$ = CC$ + C$: GOTO 1070
1135 REM ENTREE TERMINEE, REMPLIR LE RESTE DE CC$ AVEC BLANCS
1140 J = LEN (CC$)
1150 FOR I = J TO LN:CC$=CC$ + " ": NEXT I
1160 REM RE-AFFICHAGE ENTREE
1170 HTAB (HT): PRINT CC$;: RETURN
1889 REM ++++++++++++SOUS-PROGRAMME 1900++++++++++++
1890 REM BRANCHER A PROGR ENTREE CHAMP NO.F
1891 REM
1900 ON F GOTO 2000, 2100, 2200, 2300, 2400: RETURN
1989 REM
1990 REM ENTRER NOM SUR 20 CARACTERES
1991 REM
2000 VTAB 3: HTAB 11
2010 LN=20: GOSUB 1000:NA$=CC$ : RETURN
2089 REM
2090 REM ENTREE SUR 20 CARACTERES RUE
2091 REM
2100 VTAB 4: HTAB 11
2110 LN = 20: GOSUB 1000:CI$ = CC$: RETURN
2189 REM
2190 REM ENTREE SUR 20 CARACTERES VILLE
2191 REM
2200 VTAB 5: HTAB 11
2210 LN = 20: GOSUB 1000: RETURN
2289 REM
2290 REM ENTREE SUR 18 CARACTERES PAYS
2291 REM
2300 VTAB 6: HTAB 11
2310 LN = 18: GOSUB 1000: ST$ = CC$ : RETURN
2389 REM
2390 REM ENTREE CODE POSTAL SUR 5 CARACTERES
2391 REM
```


Frappez ce programme de la ligne 100 à la ligne 2400 et lancez-le. Rappelez-vous que les instructions 109 à 230 sont déjà en machine, depuis l'exercice précédent ; il est donc inutile de les ré-entrer.

Si votre programme ne tourne pas correctement, vérifiez soigneusement le listage. Attention, particulièrement, aux points-virgules des ordres PRINT.

Lorsque vous déroulerez ce programme, chacun des cinq champs sera mis en évidence à tour de rôle. Si vous entrez des caractères, ils seront affichés en écho dans le champ voulu. Lorsque vous frapperez RETURN, le champ inversé complet sera remplacé par les nouvelles données que vous avez introduites. Essayez d'employer CTRL-X pour redémarrer l'entrée des données.

Etudiez soigneusement la logique du sous-programme d'entrée de la chaîne commençant en 1000 et se terminant en 1170. Essayez de bien la comprendre avant d'aller plus loin. Remarquez la facilité avec laquelle vous voyez ce que vous entrez, et la simplicité d'un recommencement pour corriger les données erronées.

Après que le nom et l'adresse aient été introduits, le programme devrait demander à l'opérateur s'il souhaite modifier quelque chose, et dans quel champ. Un sous-programme demandant une réponse par oui ou non a été proposé antérieurement. Nous allons en prendre une version modifiée, où le programme appelant fournit la question à l'opérateur. Voici ce programme complet, avec les nouvelles instructions :

```
9 REM ******************************************************
10 REM CE PROGRAMME AFFICHE UN FORMULAIRE DEMANDANT LE
11 REM NOM ET L'ADRESSE PUIS DEMANDE CES DONNEES
12 REM ******************************************************
13 REM
100 RC$ = CHR$ (24): REM CTRL-X EST LE CARACT DE REDEMARRAGE
109 REM
110 REM AFFICHAGE DU FORMULAIRE
111 REM
120 CALL - 936: VTAB 2: REM EFFACER L'ECRAN
ET POSITIONNER LE CURSEUR
125 PRINT "ENTREZ LE NOM ET L'ADRESSE"
130 REM AFFICHAGE NUMEROS DES CHAMPS
140 INVERSE
150 FOR I = 1 TO 4: HTAB 2: PRINT I: NEXT I
160 VTAB 6: HTAB 29: PRINT 5
170 NORMAL
180 REM AFFICHAGE NOMS DES CHAMPS
190 VTAB 3: HTAB 6: PRINT "NOM:"
200 HTAB 4: PRINT "RUE:"
210 HTAB 6: PRINT "VILLE:"
220 HTAB 5: PRINT "PAYS:"
230 HTAB 31: PRINT "CODE POSTAL:"
299 REM
300 REM ENTRER LES 5 CHAMPS
301 REM
310 FOR F = 1 TO 5: GOSUB 1900: NEXT F
319 REM
320 REM POUR PERMETTRE LES MODIFICATIONS
321 REM
330 VTAB 23: HTAB 1: REM ENTREE SUR LIGNE DU BAS
340 QU$ = "UNE MODIFICATION?"
```

```
350 GOSUB 1300: REM REPONSE Y (OUI) OU N (NON)
360 IF YN$ = "N" THEN GOTO 500
370 VTAB 23: HTAB 1: REM ENTREE SUR LIGNE DU BAS
380 QU$ = "NUMERO DU CHAMP A MODIFIER"
390 LO = 1:HI = 5
400 GOSUB 1400: REM ATTENTE REPONSE NUMERIQUE
410 F = NM: GOSUB 1900: REM CHANGER CHAMP F
420 GOTO 330
490 REM FIN DE PROGRAMME
500 VTAB 23: HTAB 1: GOSUB 1200: REM EFFACER LIGNE DU BAS
510 END
990 ++++++++++SOUS-PROGRAMME 1000++++++++++
991 REM ENTRER CHAINE DANS CHAMP DE LN CARACTERES
992 REM LE CURSEUR DOIT ETRE AU DEBUT
993 REM RETURN TERMINERA L'ENTREE
994 REM LA FLECHE GAUCHE LA FERA REDEMARRER
995 REM PAS DE TESTS DE VALIDITE
996 REM LA CHAINE ENTREE EST RETOURNEE DANS CC$
997 REM
1000 HT = POS (0) + 1 : REM POSITION DEBUT DE CHAMP
1010 REM AFFICHAGE VIDEO INVERSEE
1020 INVERSE
1030 FOR I = 1 TO LN: PRINT " ";: NEXT I
1040 NORMAL : HTAB (HT): REM REPOSITION DEPART CHAMP
1050 REM ENTREE DES DONNEES
1060 CC$ = "": REM INITIALISATION SORTIE A NUL
1070 GET C$
1080 IF C$ = RC$ THEN HTAB (HT): GOTO 1020: REM RECOMMENCER
     L'ENTREE?
1090 IF C$ = CHR$ (13) THEN GOTO 1140:REM FIN ENTREE ?
1100 REM SI ENTREE TERMINEE ATTENDRE RETURN OU REDEMARRAGE
1110 IF LEN (CC$) = LN THEN GOTO 1070
1120 PRINT C$;: REM ECHO DE LA FRAPPE
1130 CC$ = CC$ + C$: GOTO 1070
1135 REM ENTREE TERMINEE. REMPLIR LE RESTE DES CC$ AVEC BLANCS
1140 J = LEN(CC$)
1150 FOR I = J TO LN:CC$ + " ": NEXT I
1160 REM REAFFICHAGE ENTREE
1170 HTAB(HT): PRINT CC$;: RETURN
1189 REM ++++++++++SOUS-PROGRAMMES 1200++++++++++
1190 REM EFFACER RANGEE DU CURSEUR
1191 REM
1200 HTAB 1: REM AU DEBUT DE LA RANGEE
1210 FOR I = 1 TO 39: PRINT " ";: NEXT I
1220 HTAB 1: REM LAISSER CURSEUR AU DEBUT RANGEE
1230 RETURN
1289 REM ++++++++++SOUS-PROGRAMMES 1300++++++++++
1290 REM POSEZ UNE QUESTION (QU$) ET REPONDEZ PAR Y OU N
     DANS YN$
1291 REM
1300 GOSUB 1200: REM EFFACER LIGNE D'ENTREE
1310 PRINT QU$;: REM AFFICHAGE CARACTERE D'APPEL
1320 GET YN$: IF YN$ <> "N" AND YN$ <> "Y" THEN GOTO 1320
```



```
1330 PRINT YN$;: REM AFFICHER REPONSE EN ECHO
1340 RETURN
1389 ++++++++++SOUS-PROGRAMMES 1400++++++++++
1390 REM DEMANDEZ REPONSE NUMERIQUE
1391 REM RETOURNER REPONSE DANS NM
1392 REM NM DOIT ETRE <=HI ET >=LO
1393 REM
1400 GOSUB 1200: REM EFFACER LIGNE D'ENTREE
1410 PRINT QU$;: REM AFFICHAGE CARACTERE D'APPEL
1420 GET C$:NM = VAL (C$)
1425 REM TESTER SI ENTREE DANS LA GAMME
1430 IF NM < LO OR NM > HI THEN GOTO 1420
1440 PRINT C$;: REM REPONSE EN ECHO
1450 RETURN
1889 REM ++++++++++SOUS-PROGRAMME 1900++++++++++
1890 REM BRANCHER A PROGR ENTREE  CHAMP NO.F
1891 REM
1900 ON F GOTO 2000,2100,2200,2300,2400: RETURN
1989 REM
1990 REM ENTREE SUR 20 CARACTERES DU NOM
1991 REM
2000 VTAB 3: HTAB 11
2010 LN = 20: GOSUB 1000:NA$ = CC$: RETURN
2089 REM
2090 REM ENTREE SUR 20 CARACT DE LA RUE
2091 REM
2100 VTAB 4: HTAB 11
2110 LN = 20: GOSUB 1000:CI$ = CC$: RETURN
2189 REM
2190 REM ENTREE SUR 20 CARACT DE LA VILLE
2191 REM
2200 VTAB 5: HTAB 11
2210 LN = 20: GOSUB 1000: RETURN
2289 REM
2290 REM ENTREE SUR 17 CARACT DU PAYS
2291 REM
2300 VTAB 6: HTAB 11
2310 LN = 17: GOSUB 1000:ST$ = CC$: RETURN
2389 REM
2390 REM ENTREE CODE POSTAL SUR 5 CARACTERES
2391 REM
2400 VTAB 6: HTAB 35
2410 LN = 5: GOSUB 1000:ZI$ = CC$ RETURN
```

Vous devriez étudier attentivement ce programme de nom et d'adresse et comprendre quelles sont les aides aux entrées qui ont été incluses. Il s'agit de :

1. En labellant chaque champ et en juxtaposant un masque d'entrée en vidéo inversée, vous indiquez clairement à l'opérateur quelles données sont attendues et combien d'espaces sont disponibles.
2. Lorsque l'opérateur frappe un numéro de champ pour une modification, le masque d'entrée en inversé lui indique à nouveau le champ visé.

3. L'opérateur n'est pas contraint de remplir totalement de caractères le champ qui lui est imparti ; lorsqu'il presse RETURN, ce remplissage se fait automatiquement avec des blancs.
4. A tout moment, l'opérateur peut recommencer l'entrée dans un champ avec CTRL-X.
5. Lorsque le programme pose des questions, les réponses se font simplement par Y pour oui (« Yes ») et N pour non, ou par un numéro entre 1 et 5 pour le champ. Toute autre méthode n'est pas bonne. Par exemple, reconnaître un Y pour oui et n'*importe quel autre* caractère pour non peut être désastreux, car une frappe accidentelle d'un caractère sera alors interprétée comme une réponse. A l'inverse, reconnaître N pour non et tout autre caractère pour oui obligera l'opérateur, après une frappe accidentelle, à reprendre l'entrée.

Certaines caractéristiques d'entrée n'ont pas été incluses dans ce programme mais pourraient être ajoutées :

1. Tester le code postal, qui ne doit contenir que des chiffres (mais attention : pas dans tous les pays).
2. De nombreux programmeurs attentionnés poseront la question EN ETES-VOUS CERTAIN ? Si l'opérateur répond NON à la question UNE MODIFICATION ? On procure à l'opérateur une seconde chance de correction s'il a, par mégarde, frappé la mauvaise touche.
3. Fournir une touche supplémentaire avortant l'entrée courante et restituant les données antérieures. Par exemple, si l'opérateur choisit un champ erroné à modifier, le programme ci-dessus l'oblige à ré-entrer le champ. Or, le programme pourrait reconnaître une autre touche qui stopperait et annulerait l'entrée courante, conservant les anciennes valeurs.
4. Autoriser la touche ← à servir d'espace arrière. Chaque fois qu'elle sera pressée, le curseur fera un pas en arrière et le dernier caractère entré sera remplacé par un masque d'entrée en vidéo inversée sur l'écran et par un blanc dans le sous-programme de la chaîne de sortie (CC$). Naturellement, il ne peut y avoir de pas en arrière quand le curseur est sur le bord gauche du champ.
   Essayez de modifier ce programme pour inclure les caractéristiques supplémentaires décrites ici.

FORMATAGE DE LA SORTIE

Lorsque vous mettez Apple II sous tension, la sortie se fait automatiquement sur l'écran. Il existe des instructions qui organisent la sortie sur imprimante ou sur tout autre périphérique capable de recevoir une sortie.
La programmation d'une sortie sur imprimante présente quelques différences par rapport à la programmation d'une sortie sur écran. Par exemple, l'imprimante pourra accepter des lignes plus grandes que l'écran ; d'autre part, les instructions HTAB et VTAB (ou TAB, en Integer Basic) qui servent à déplacer le curseur sur l'écran ne peuvent être utilisées pour déplacer une tête d'impression sur une feuille de papier.
Mais il y a aussi bien des similitudes entre les techniques de programmation des sorties sur imprimante et sur écran. L'essentiel de la discussion qui suit s'aplique aux deux. Nous noterons ce qui ne s'applique qu'à l'écran. Si vous envisagez la rédaction de programmes

avec sortie sur imprimante, lisez également la discussion sur la programmation de l'imprimante donnée plus loin dans ce chapitre.
La programmation d'une sortie est bien plus simple que celle d'une entrée, car il n'y a plus d'interaction avec l'opérateur dont il faudrait tenir compte. Vous devez vous assurer que l'information sera facile à exploiter, et c'est tout. Voici quelques règles à respecter :

1. Evitez de charger de trop d'informations un tout petit espace.
2. Si des nombres ou des chaînes de caractères sont listés en colonne, alignez les données de façon que l'œil puisse parcourir rapidement la colonne.
3. Utilisez la vidéo inversée pour mettre en valeur les informations clés, les entêtes, ou les têtes de lignes (mais avec l'écran uniquement).

Voici quelques suggestions qui vous éviteront des erreurs lors de la programmation de sorties :

1. Faites suivre les données d'une instruction PRINT par des points-virgules, sauf si vous imposez un espacement par colonne avec des virgules. C'est là l'erreur la plus courante en programmation de sorties.
2. Avant d'aller plus avant, concevez l'organisation de votre écran ou de votre feuille. Utilisez des feuilles quadrillées, ou des matrices spécialement dessinées pour préparer la sortie. L'appendice L montre un écran qui vous permettra de calculer les rangées et les colonnes avec précision. L'autre alternative consisterait à procéder par tâtonnement, ce qui prendrait, en définitive, bien plus de temps que de tracer une matrice.
3. Prêtez attention aux tableaux contenant un nombre de données non exactement divisible par le nombre de colonnes. Par exemple, supposez que 25 items se trouvent dans le tableau N$(I), que vous allez imprimer sur trois colonnes. Vous pourriez être tenté d'écrire quelque chose comme cela :

```
100 FOR I = 1 TO 25 STEP 3
200 REM TRAITEMENT COLONNE 1
  .
  .
  .
300 REM TRAITEMENT COLONNE 2
  .
  .
  .
400 REM TRAITEMENT COLONNE 3
  .
  .
  .
500 NEXT I
```

Mais, lors de la passe finale dans la boucle FOR-NEXT, les index 26 et 27 vont être calculés bien qu'ils n'existent pas. Vous pouvez facilement tester la fin d'un tableau dans une boucle FOR-NEXT de la façon suivante :

```
100 FOR I=LO TO HI STEP ST
  .
  .
  .
350 I=I+1
360 IF I>HI THEN 510
  .
  .
  .
500 NEXT
510 REM SUITE DU PROGRAMME
```

L'écran en fenêtre de données

Lorsqu'on traite de grandes quantités d'informations, une bonne technique consiste à employer l'écran comme une fenêtre pour les données. A tout moment, l'écran ne montre qu'une partie de données disponibles. On peut, pour cela, regrouper les données en pages, chacune correspondant à l'écran. Les programmes appliquant cette technique doivent disposer de sous-programmes distincts affichants les entêtes et les données pour chaque page. Ils doivent aussi offrir à l'utilisateur le moyen de passer d'une page à l'autre. Très souvent, les tableaux serviront à stocker de grandes masses de données. Dans ce cas, vous pouvez vous servir de l'écran comme d'un viseur de caméra. Imaginez que le tableau soit rédigé sur un vaste tableau noir et que vous examiniez celui-ci à travers un viseur de caméra. Le tableau noir est tellement grand que vous ne pouvez le voir en entier d'un coup ; mais vous pouvez déplacer le viseur et parcourir le tableau. L'écran peut imiter ce mode d'action.

Nous allons illustrer l'usage de l'écran en fenêtre sur les données en créant, en Applesoft, un exemple de la technique de viseur. Pour commencer, nous allons créer un tableau d'entiers sur deux dimensions. A la valeur de chaque élément du tableau, nous allons attribuer un nombre de quatre chiffres identifiant l'index du tableau, ainsi :

$$X\%(i,j) = 0i0j$$

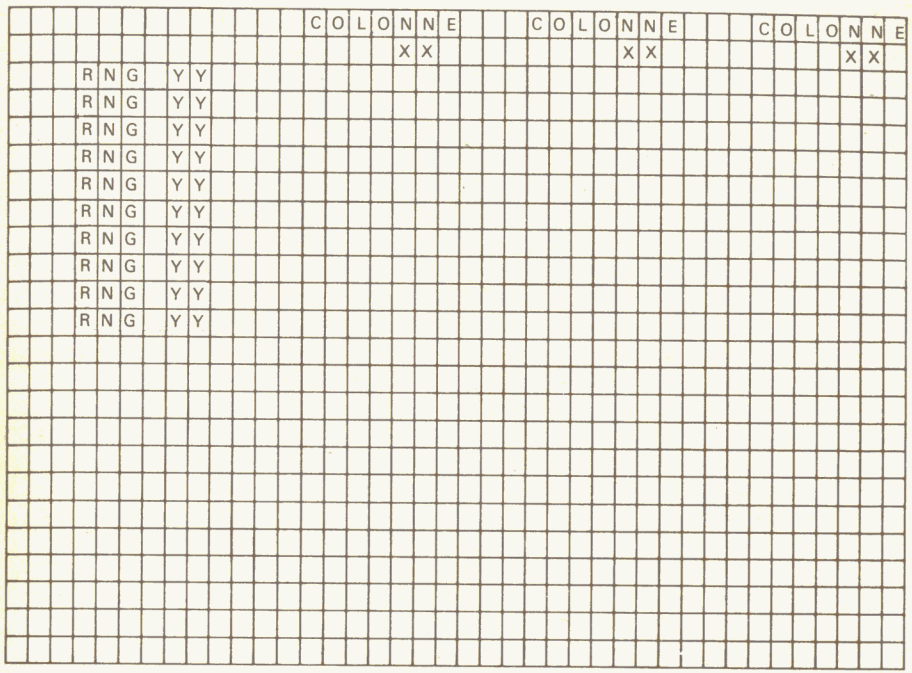
Par exemple :

$$X\%(3,2) = 0302$$
$$X\%(19,8) = 1908$$
$$X\%(11,12) = 1112$$

Nous allons initialiser ce tableau d'entiers très simplement :

```
10  DIM X%(14,50)
20  FOR I = 1 TO 14
30  FOR J = 1 TO 50
40 X%(I,J) = I * 100 + J
50  NEXT J
60  NEXT I
```

Maintenant, on va afficher une portion de ce tableau. Nous allons utiliser les deux rangées du haut de l'écran pour les entêtes, et les neuf premiers espaces de chaque ligne pour les têtes de lignes, comme suit :



Les colonnes de nombres apparaîtront là où sont les XX, et les rangées, en YY. Voici le programme servant à créer les rangées et les colonnes en vidéo inversée :

```
1000  INVERSE
1020  FOR I = 1 TO 3
1030  HTAB 4 + I * 10: PRINT "COLONN"
1040  NEXT I
1050  PRINT
1060  FOR I = 0 TO 2
1065  REM 1 ESPACE SUPPLEM POUR NO. DE 1 CHIFFRE
1070 S% = 0 IF C% + I < 10 THEN S% = 1
1080  HTAB 18 + I * 10: PRINT  SPC( S%);C% + I;
1090  NEXT I
1100  PRINT
1110  FOR I = R% TO R% + 9
1115  REM 1 ESPACE SUPPLEM POUR NO. DE 1 CHIFFRE
1120 S% = 0: IF I < 10 THEN S% = 1
1130  HTAB 4: PRINT "RNG";: HTAB 8: PRINT SPC( S%);I.
1140  NEXT I
1150  NORMAL : RETURN
```

Nous avons délibérément créé une fenêtre plus petite que l'écran complet afin de mieux illustrer ce concept de fenêtre sur les données. Rien ne vous empêche d'occuper l'écran complet ; cependant, les circonstances feront que vous voudrez parfois une petite fenêtre, de façon à faire apparaître d'autres informations conjointement sur l'écran.

Nous allons maintenant ajouter des instructions demandant à l'opérateur d'entrer des nombres représentant les plus petites colonnes et rangées du tableau. L'élément du

tableau correspond à ces plus petites valeurs apparaîtront au sommet et à gauche de l'affichage. La fenêtre se remplira avec des éléments sur des colonnes et rangées adjacentes, jusqu'au bout. Voici ce programme complet :

```
5  REM PROGRAMME DE FENETRE SUR UN TABLEAU
6  REM ***********************************************
10  HOME : PRINT "ATTENDEZ... INITIALISATION!";
20  DIM X%(14,50)
30  FOR I = 1 TO 14
40  FOR J = 1 TO 50
50 X%(I,J) = I * 100 + J
60  NEXT J
70  NEXT I
75  HOME
80  HTAB 1: VTAB 20: INPUT "ENTREZ COLONNE" (1 A 12):";C%
90  IF C% < 1 OR C% > 12 THEN GOTO 80
100  VTAB 21: INPUT "ENTREZ RANGEE (1 A 41):";R%
110  IF R% < 1 OR R% > 41 THEN GOTO 100
120  VTAB 1: HTAB 1: GOSUB 1000: REM AFFICH EN-TETES
130  REM REMPLIR LA FENETRE
135  VTAB 3
140  FOR I = R% TO R% + 9
150  HTAB 10
160  FOR J = C% TO C% + 2
165  REM AFFICH VALEURS JUSTIFIEES A DROITE
170 X$ = STR$ (X%(J,I)): PRINT SPC( 10 -  LEN (X$));X$;
180  NEXT J
190  PRINT : REM LIGNE SUIVANTE
200  NEXT I
210  VTAB 22: PRINT "ON CONTINUE? REPONDEZ PAR Y OU N";
220  GET C$: IF C$ <  > "Y" AND C$ <  > "N" THEN 220
230  IF C$ = "Y" THEN GOTO 80
240  END
990  REM
991  REM ++++++++++SOUS-PROGRAMME 1000+++++++++
992  REM AFFICHAGE DES EN-TETES
1000  INVERSE
1020  FOR I = 1 TO 3
1030  HTAB 4 + 1 * 10: PRINT "COLONN";
1040  NEXT I
1050  PRINT
1060  FOR I = 0 TO 2
1065  REM 1 ESPACE SUPPLEM POUR NO. 1 CHIFFRE
1070  S% = 0: IF C% + I < 10 THEN S% = 1
1080  HTAB 18 + I * 10: PRINT SPC( S%);C% + I;
1090  NEXT I
1100  PRINT
1110  FOR I = R% TO R% + 9
1115  REM 1 ESPACE SUPPLEM POUR NO. 1 CHIFFRE
1120  S% = 0: IF I < 10 THEN S% = 1
1130  HTAB 4: PRINT "RNG";: HTAB 8: PRINT SPC( S%);I
1140  NEXT I
1150  NORMAL : RETURN
```


Introduisez ce programme dans l'ordinateur et lancez-le. Si vous l'avez fait correctement, vous remarquerez que la première chose que fait la machine, c'est de ne rien faire pendant un moment ; elle est en train de traiter les instructions emboîtées FOR-NEXT des lignes 30 à 70. Il lui faut entre cinq et dix secondes pour remplir le tableau X% avec des nombres. Un message d'attente pour initialisation est affiché, faute de quoi l'utilisateur pourrait supposer que sa machine ne fait rien. C'est une bonne habitude à prendre que d'afficher de tels messages lors de périodes d'inactivité apparente.

Notez que des numéros de colonnes de 1 à 12 ont été attribués. Puisqu'il n'y a que trois colonnes, toute colonne jusqu'à 12 reste dans les dimensions des 14 colonnes. Les rangées vont de 1 à 41 ; les dix autres numéros de colonnes démarrant en 41 irait jusqu'en 50, ce qui est la seconde dimension du tableau.

La valeur entière du tableau X% est convertie en une chaîne par la ligne 170, avant affichage. Nous avons assuré cette conversion pour simplifier la mise en page. Il est alors facile de calculer le nombre d'espaces entre colonnes, comme le montre l'instruction PRINT de la ligne 170. Il est moins aisé d'aligner correctement des valeurs numériques lorsqu'on les affiche directement. Pour le vérifier, modifiez ainsi la ligne 170 :

```
170 PRINT SPC<7>;X%<J,I>;
```

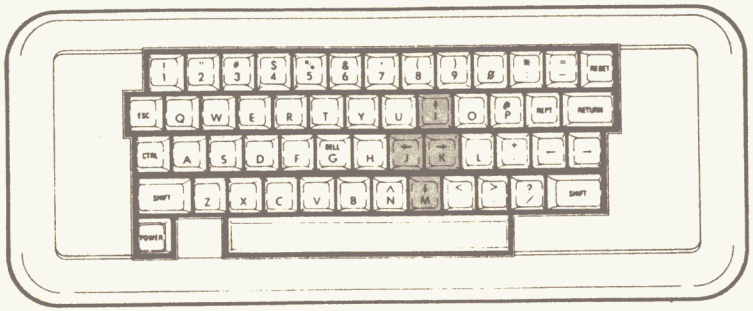
Les nombres s'aligneront tant que vous n'afficherez pas des valeurs sur quatre chiffres, dans lequel cas l'affichage sera trop large pour un écran de 40 caractères.

Notre programme prend grand soin de terminer l'affichage sur la 39e colonne de l'écran plutôt que sur la 40e. Si vous dépassiez la 40e colonne, l'affichage se poursuivrait à la ligne suivante. Ne vous embrouillez pas avec ce cauchemar que constitue le formatage de l'affichage et avec l'interaction qui pourrait résulter entre le retour chariot provoqué par un affichage sur la colonne 40 et votre propre programmation de retour chariot.

A titre d'exercice, il serait intéressant de modifier ce programme d'affichage de façon qu'il sorte de la 40e colonne. Pour cela, modifiez la tabulation horizontale de la ligne 150 et passez-la de 10 à 11 ; à la ligne 1030, passez de 4+I*10 à 5+I*10 ; à la ligne 1080, de 18+I*10 à 19+I*10, et enfin à la ligne 1130, de 4 et 8 à 5 et 9. Maintenant, relancez le programme ; vous avez maintenant trop de retours chariot ; examinez si vous pouvez en éliminer quelques-uns afin de rétablir un affichage correct. Ce n'est pas là une tâche aisée.

Remarquez que les instructions des lignes 80, 100 et 220 demandant des entrées sont suivies par des instructions interdisant des entrées invalides. Même dans cette simple démonstration, nous avons pris le temps de protéger les entrées.

Un raffinement utile, dans un programme affichant une fenêtre sur des données, consiste à procurer à l'opérateur le moyen de déplacer la fenêtre en avant et en arrière d'une rangée, à gauche et à droite d'une colonne. C'est facile à faire. Nous allons employer les touches I, J, K et M pour ces commandes directionnelles de façon très semblable à ce à quoi elles servent en mode éditeur (décrit dans le chapitre 3). La touche I déplace une rangée en haut ; M, une rangée en bas ; J, une colonne à gauche et K, une colonne à droite, comme indiqué ci-dessous.



Pour cela, nous devons remplacer les lignes 210 à 240 par les ordres suivants :

```
210 VTAB 22: PRINT "ON CONTINUE?"
215 PRINT "ENTREZ LA DIRECTION (I,J,K,M) Y, OU N ";
220 GET C$
225 REM UNE RANGEE VERS LE BAS?
230 IF R% > 1 THEN IF C$ = "M" THEN R% = R% - 1: GOTO 120
235 REM UNE RANGEE VERS LE HAUT?
240 IF R% < 41 THEN IF C$ = "I" THEN R% = R% + 1: GOTO 120
245 REM UNE COLONNE A GAUCHE?
250 IF C% > 1 THEN IF C$ = "J" THEN C% = C% - 1 : GOTO 20
255 REM UNE COLONNE A DROITE?
260 IF C% < 12 THEN IF C$ = "K" THEN C% = C% + 1: GOTO 120
270 IF C$ = "Y" THEN GOTO 80: REM NOUVELLE COL ET RNGEE
280 IF C$ = "N" THEN END
285 REM SONNETTE ET REJET D'AUTRE ENTREE
290 PRINT CHR$(7);: GOTO 220
```

Remarquez la linéarité de la logique, bien qu'on continue à tester si l'opérateur commet des erreurs. Toute autre entrée que les six caractères permis est rejetée ; de même, une commande de direction qui demanderait un dépassement des limites du tableau est rejetée.

PROGRAMMATION DES IMPRIMANTES

L'Apple II traite une imprimante comme un substitut de l'écran. Pour créer une sortie sur l'imprimante, cependant, vous devrez inclure un ordre qui détourne la sortie de l'écran vers l'imprimante. La sortie doit être ramenée vers l'écran après la fin de l'impression. C'est le rôle PR#.

Les imprimantes sont connectées à l'ordinateur via des interfaces série ou parallèles, selon l'imprimante.

Normalement, la carte d'interface série est enfichée dans le connecteur 1 de l'Apple II, et l'interface parallèle dans le connecteur 3. Mais il s'agit là d'une convention plutôt que d'une nécessité. En fait, ces interfaces série ou parallèle pourraient être insérées dans n'importe quel connecteur (autre que le 0).

Sortie d'un texte sur imprimante

Vous pouvez vous souvenir que PR# est considéré comme un ordre du DOS lorsque le DOS est présent. Cela signifie qu'il doit être donné avec un caractère en préfixe, CTRL-D

(code ASCII 4). Le programme ci-dessous imprimera deux lignes de texte sur une imprimante reliée au connecteur 1 de l'Apple II, avec DOS présent :

```
10 REM SORTIE DE 2 LIGNES SUR IMPRIMANTE
40 REM SELECTION DU PORT SERIE D'E/S
50 PRINT "PR#1"
60 PRINT "SUR ECRAN OU SUR IMPRIMANTE,"
70 PRINT "LA SORTIE SE FERA OCTET PAR OCTET."
80 REM DESELECTION DE L'IMPRIMANTE
90 PRINT "PR#0"
100 END
```

L'instruction de la ligne 30 crée le caractère de commande convertissant l'ordre PR# en Basic. PR# apparaît sur les lignes 50 et 90. La ligne 50 détourne la sortie de l'écran et la dirige sur l'imprimante, la ligne 90 ramenant la sortie sur l'écran. Les ordres PRINT des lignes 60 et 70 émettent les deux lignes qui seront imprimées. Voici ce que vous lirez sur le papier :

```
SUR ECRAN OU SUR IMPRIMANTE,
LA SORTIE SE FERA OCTET PAR OCTET.
```

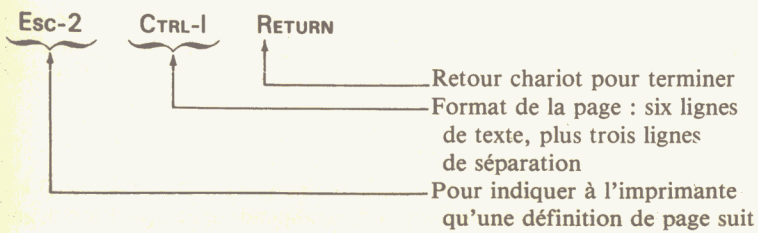
Voici le même programme, le DOS étant absent :

```
 10 REM SORTIE DE 2 LIGNES SUR IMPRIMANTE
 40 REM SELECTION DU PORT SERIE D'E/S
 50 PRINT "PR#1"
 60 PRINT "SUR ECRAN OU SUR IMPRIMANTE,"
 70 PRINT "LA SORTIE SE FERA OCTET PAR OCTET."
 80 REM DESELECTION DE L'IMPRIMANTE
 90 PRINT "PR#0"
100 END
```

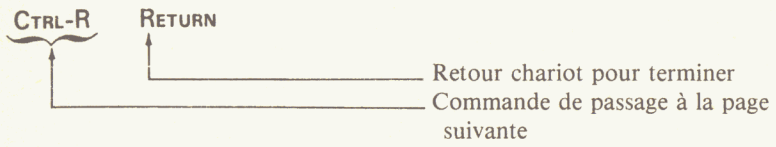
Imprimantes programmables

De nombreuses imprimantes autorisent le formatage de la sortie par le programme. En incluant les caractères de commande nécessaires, dans le flot du texte sortant vers l'imprimante, vous pouvez ajuster la longueur des lignes, le jeu de caractères, la longueur de la page, et bien d'autres caractéristiques.

De nombreuses imprimantes courantes sont associées à l'Apple II. Du point de vue programmation, cependant, Apple II les traite de façon semblable. Supposons, par exemple, qu'on emploie une imprimante type 810 de Texas Instruments, reliée au connecteur 1 de l'Apple II. Dans le manuel de l'imprimante, on peut trouver comment ajuster la page de texte, avec six lignes par page ; cela se fait ainsi :



De plus, on peut imposer à l'imprimante d'avancer en haut de la page suivante (le « form feed », en anglais), à l'aide de la séquence :



En modifiant le programme précédent, le listage suivant, en Applesoft, sort ces deux lignes de texte 15 fois, en blocs de six lignes. Ce programme tourne avec le DOS présent :

```
10 REM SORTIE DE TEXTE SUR IMPRIMANTE
11 REM AVEC 6 LIGNES PAR PAGE ET 5 PAGES
20 REM CREATION DU CARACTERE CTRL-D
30 D$ = ""
40 REM SELECTION DU PORT SERIE D'E/S
50 PRINT D$;"PR#1"
51 REM SELECTION DE 6 LIGNES PAR PAGE
52 PRINT CHR$(27);"2";CHR$(6)
53 PRINT CHR$(12);: REM POSITIONNEMENT DE LA PAGE
   POUR SORTIE
54 FOR I = 1 TO 15
60 PRINT "SUR ECRAN OU SUR IMPRIMANTE,"
70 PRINT "LA SORTIE SE FERA OCTET PAR OCTET."
75 NEXT I
80 REM DESELECTION DE L'IMPRIMANTE
90 PRINT D$;"PR#0"
100 END
```

Dans ce programme, l'ordre de la ligne 52 sélectionne six lignes par page. La fonction CHR$(27) représente le caractère d'échappement ESC. La fonction CHR$(6) définit une page de six lignes, plus trois lignes de séparation. Les point-virgules assurent la concaténation des trois caractères.
La ligne 53 positionne la page suivante ; la fonction CHR$(12) est le caractère de changement de page. Un point-virgule suit car sans lui, l'ordre PRINT provoquerait un retour chariot et l'on obtiendrait une ligne vide. Ainsi, sans ce point-virgule, la première page comprendrait une ligne vide, cinq lignes de texte, alors que les suivantes auraient six lignes de texte par page.
Les autres imprimantes programmables sont commandées en émettant des codes de commande appropriés de la même facon.
Ce programme ne fonctionne pas en Integer Basic car il utilise la fonction CHR$. Vous pouvez cependant créer toutes sortes de chaînes en pressant les touches appropriées ou une combinaison de touches ; aussi pourrez-vous remplacer CHR$ par une paire de guillemets encadrant un caractère qui n'est pas un caractère d'impression. Par exemple, la chaîne créée en frappant les trois touches « ESC » est la même que CHR$(27). Voyez l'appendice I pour la liste complète des équivalences.

Impression des listages de programmes

Lorsque vous commandez LIST au clavier, tout programme présent dans l'Apple II est listé sur l'écran. Pour *imprimer* le listage, introduisez la commande approprié PR # avant

de faire LIST. En supposant que l'imprimante soit reliée au connecteur 1, les pas suivants sont alors requis :

1. Assurez-vous que le programme à lister se trouve bien dans la mémoire de l'Apple II.
2. Le curseur étant sur une ligne vide de l'écran, sélectionnez l'imprimante en frappant PR # 1. Le curseur reviendra au premier caractère de cette ligne sans passer à la ligne suivante.
3. Frappez maintenant LIST. Cette commande n'apparaîtra pas sur l'écran, mais sur l'imprimante avec le listage du programme.
4. Lorsque tout le programme a été listé, renvoyez la sortie à l'écran en frappant PR # 0. Cette commande n'apparaîtra pas sur l'écran mais sur l'imprimante.

RANGEMENT DE DONNÉES EN CASSETTE

Nous avons appris, dans le dernier chapitre, comment sauvegarder et charger des programmes avec une cassette. En Applesoft, vous pourrez aussi enregistrer des tableaux numériques ou entiers sur bande magnétique en cassette.
L'instruction STORE enregistre les données et l'instruction RECALL les rappelle, pour les recharger en machine. Aucune de ces deux instructions ne commande les mouvements de la bande, ni n'affiche de directives à l'opérateur pour qu'il manipule les boutons ou touches de l'unité à cassettes. Le programme ci-dessous montre l'usage de STORE et RECALL. Il attribue des valeurs à un tableau numérique et les range en cassette, puis remet toutes les valeurs à zéro, et enfin recharge le tableau à partir de la cassette. Les valeurs du tableau en des points critiques sont affichées pour tenir l'opérateur au courant du déroulement du programme.

```
10 REM CE PROGRAMME MONTRE L'ACTION DE STORE ET RECALL
20 REM ************************************************
30 DIM A(10)
40 HOME
50 PRINT TAB( 4);"SAUVEGARDE"; TAB( 13);"MISE A ZERO";
   TAB( 22);"RECHARGE"
60 REM INITIALISATION DU TABLEAU
70 FOR I = 1 TO 10:A(I) = I: NEXT I
80 T = 8: GOSUB 1000: REM AFFICH VALEURS A SAUVEGARDER
90 VTAB 20: HTAB 1
100 PRINT "RE-EMBOBINEZ LA CASSETTE"
110 PRINT "METTEZ-LA EN 'ENREGISTREMENT'"
120 INPUT "ET FAITES 'GO'"; C$
130 IF C$ < > "GO" THEN GOTO 90
140 STORE A
160 CLEAR : REM REMISE A ZERO DES VALEURS
179 VTAB 2:T = 18: GOSUB 1000: REM AFFICHAGE DU TABLEAU
180 VTAB 20: HTAB 1
190 GOSUB 1100 REM EFFACEMENT DERNIERES INSTRUCTIONS
200 VTAB 20: HTAB 1
210 PRINT "RE-EMBOBINEZ LA CASSETTE, PUIS 'LECTURE'"
220 INPUT "ET FRAPPEZ 'GO' ";C$
230 IF C$ < > "GO" THEN GOTO 200
```

```
240 RECALL A
260 VTAB 2:T = 28: GOSUB 1000: REM AFFICHAGE TABLEAU
265 VTAB 20: HTAB 1
270 GOSUB 1100: REM EFFACEMENT DERNIERES INSTRUCTIONS
280 VTAB 20: HTAB 1
290 PRINT "PRESSEZ BOUTON 'STOP'"
300 END
990 REM ++++++++++SOUS-PROGRAMME 1000++++++++++
991 REM AFFICHAGE VALEURS TABLEAU A
1000 FOR I = 1 TO 10: HTAB T: PRINT A(I): NEXT I: RETURN
1090 REM ++++++++++SOUS-PROGRAMME 1100++++++++++
1091 REM EFFACEMENT 3 LIGNES SUR L'ECRAN
1100 FOR I = 1 TO 119: PRINT " ";: NEXT I: RETURN
```

Vous pouvez sauvegarder un tableau sous un nom de variable et le rappeler avec un nom différent de variable. De façon générale, les dimensions du tableau que vous rangez devront être les mêmes que celles du tableau que vous chargez. On trouvera quelques exceptions compliquées dans le chapitre 8. A moins que vous ne recherchiez des effets spéciaux, employez RECALL avec des données en cassette et pour un réseau dimensionné comme celui que vous aviez rangé avec STORE.

OPTIMISATION DES PROGRAMMES

Traditionnellement, le programme optimal est celui qui, pour une tâche donnée, tourne le plus vite et occupe le moins de mémoire. Naturellement, ce double objectif doit être tempéré car le programme doit aussi rester fiable, facile à écrire, facile à utiliser et lire, et facile à modifier. A terme, vous tirerez davantage de profit en consacrant votre temps directement à ces derniers aspects plutôt qu'en recherchant la vitesse maximale ou l'économie extrême en mémoire. En outre, si vous savez comment optimiser la vitesse et l'occupation mémoire, vous pourrez rédiger d'emblée des programmes efficaces, n'ayant plus guère besoin de mise au point ultérieure. Dans cet esprit, voici quelques méthodes servant à écrire des programmes plus rapides et consommant moins de mémoire.
Certaines techniques rendent les programmes plus rapides et leur font aussi occuper plus d'espace, alors que des techniques économisant la mémoire accroîtront le temps d'exécution. C'est à vous de décider ce qui est le plus important dans votre cas.

PROGRAMMES PLUS RAPIDES

Evitez d'utiliser des constantes (telles que 0, 100, « Y », « ENTER »). De préférence, attribuez très tôt dans le programme les valeurs des constantes à des variables. C'est particulièrement important lorsque vous employez répétitivement des constantes entières dans des expressions réelles. Cela prend plus de temps de convertir une constante en une valeur réelle qu'il n'en faut pour rechercher la valeur d'une variable. Lorsqu'une telle conversion prend place dans une boucle FOR-NEXT, un sous-programme d'usage fréquent ou une fonction définie par l'utilisateur, la différence devient bien plus significative. Cette technique offre l'avantage supplémentaire de faciliter les modifications dans votre programme. Si vous deviez changer la constante, il serait plus facile de modifier une instruction d'affectation plutôt que de changer toutes les lignes où cette constante intervient.
Utilisez les variables qui interviennent fréquemment dans un programme aussitôt que possible au cours de l'*exécution* du programme. L'attribution d'espace mémoire pour les

variables se fait dans l'ordre de leur arrivée. Le Basic trouvera ainsi la variable recherchée en tête de la liste, donc plus rapidement que si elle se trouvait en fin de liste.
Lorsque Basic rencontre une instruction de branchement à un autre numéro de ligne, il commence par rechercher ce numéro de ligne au début du programme, puis séquentiellement jusqu'à la fin, et ce jusqu'à ce qu'il trouve cette ligne. Donc, plus le numéro de ligne sera bas, en relation avec le reste du programme, plus vite Basic exécutera le branchement. De ce fait, attribuez les plus petits numéros de lignes aux sous-programmes les plus usuels.
En Applesoft, n'incorporez pas des variables d'index dans un ordre NEXT ; ainsi, Applesoft n'aura pas à vérifier si l'index spécifié est correct.

POUR COMPACTER LES PROGRAMMES

L'emploi de sous-programmes évite la duplication des séquences identiques. Il améliore aussi la lisibilité et la fiabilité des programmes tout en facilitant les modifications.
Utilisez aussi l'élément zéro dans les tableaux (par exemple, X(0), B(0)).
Il y a moins de caractères dans un nom court de variable que dans une constante disposant d'un grand nombre de digits. C'est pourquoi il vaut mieux affecter des valeurs constantes à des noms de variables ; vous utiliserez ensuite ces variables à la place des valeurs des constantes.
Placez plus d'une instruction dans une ligne de programme. Chaque recommencement de ligne demande cinq octets. Notez cependant que des lignes à instructions multiples sont difficiles à éditer et encore plus difficiles à lire et à comprendre. Se représenter comment fonctionne alors un programme la première fois n'est pas toujours une sinécure ; mais s'il faut recommencer cet exercice...
Employez judicieusement les REM ; abrégez les commentaires. Mais attention : moins le programme sera commenté, plus il sera difficile à comprendre.
Usez parcimonieusement des variables. Chaque variable requiert un certain espace mémoire, même si vous ne l'utilisez qu'une fois. Etablissez un système d'assignation des variables dans lequel vous incluerez des variables intermédiaires utilisables dans les boucles FOR-NEXT, les calculs intermédiaires, etc. Mais sans aller trop loin en ce sens : votre programme sera plus facile à comprendre si vous traitez les variables de façon significative classique. Etablissez des identités standard pour des variables individuelles (par exemple, NC$ pour nom du client) et des groupes de variables (par exemple, toutes les variables intermédiaires commenceront par X).
Recourez à INPUT et aux listes de données (si vous disposez d'une unité à disquettes ; voir le chapitre 5) au lieu des instructions d'affectation et des ordres DATA.
En Applesoft, employez des tableaux d'entiers au lieu de tableaux de nombres réels. Chaque valeur entière occupe deux octets de mémoire alors qu'une valeur réelle en demande cinq. Utilisez la fonction FRE périodiquement dans votre programme pour débarasser la zone de stockage des chaînes en mémoire.

DÉBUGAGE (MISE AU POINT)

Un nouveau programme ne fonctionnera jamais de la façon espérée. Même si la syntaxe du Basic ne comprend pas d'erreurs, il y en aura sûrement dans la logique du programme. Chaque erreur s'appelle un *bug,* en américain, soit un « insecte » ; de là vient le mot « débugage » traduit par débuguer, pour mise au point, déverminage, extraction des erreurs. Il existe plusieurs approches pour débuguer un programme.
Voilà en tout cas l'occasion traditionnelle de vous conseiller : prenez votre temps, plani-

fiez bien, faites les choses bien du premier coup. Ne vous installez pas devant votre clavier en ne sachant qu'à moitié ce que vous voulez que votre programme fasse. Si vous êtes nouveau venu en programmation, lisez, à la suite de cet ouvrage, un livre de programmation en Basic, spécialisé, dans lequel vous apprendrez de bonnes règles de conduite.
Une fois votre programme rédigé, et s'il ne produit pas les résultats que vous en attendiez, utilisez les quelques instructions Basic suivantes pour vous aider à le mettre au point.

L'instruction PRINT

Curieusement, la classique instruction PRINT constitue un outil de débugage très utile. Vous pouvez inclure des instructions PRINT supplémentaires, temporairement dans votre programme, en des points stratégiques pour afficher des messages vous indiquant que le programme a atteint tel point sans défaillance, et pour vous fournir les valeurs des variables acquises. De cette façon, vous pourrez suivre le programme en cours d'exécution et tester les résultats des calculs intermédiaires.

L'instruction TRACE

L'instruction TRACE répond à son nom : elle trace l'exécution du programme en affichant le numéro de ligne de chaque instruction exécutée. Pour examiner son fonctionnement, introduisez le programme suivant en machine, puis frappez TRACE, suivi par RUN :

```
100 PRINT "ENTREZ UN NOMBRE DE 1 A 5"
110 INPUT N
120  IF N = 1 THEN  PRINT "UNO";
130  IF N = 2 THEN  PRINT "DOS";
140  IF N = 3 THEN  PRINT "TRES";
150  IF N = 4 THEN  PRINT "CUATRO";
160  IF N = 5 THEN  PRINT "CINCO";
170  IF N > 5 THEN  GOTO 100
180  FOR I = 1 TO N
190 PRINT " *";: REM AFFICHAGE DE N ASTERISQUES
200 NEXT I
210 CALL - 936: REM EFFACER L'ECRAN
220 GOTO 100
```

Pour annuler TRACE et revenir au mode normal, frappez l'ordre NOTRACE.

L'instruction DSP

Disponible uniquement en Integer Basic, l'instruction DSP constitue un autre outil de débugage apprécié. Voici un exemple :

```
>10 DSP COMPTE
```

Dès que cette instruction est exécutée, Apple II vous préviendra chaque fois que la valeur de COMPTE change, et à quelle ligne ce changement est intervenu. Parce que RUN annule un DSP précédent, vous devrez lancer le programme avec un GOTO, ou encore introduire DSP dans une ligne de programme.
Vous pouvez aussi annuler DSP pour une variable par un ordre NODSP ; par exemple :

```
>300 NODSP NOM$
```


Cette instruction étant exécutée, Apple II cesse de vous notifier les changements de la valeur de la variable NOM$.
Essayez d'ajouter les lignes suivantes à l'exemple que nous avons proposé avec TRACE afin d'examiner l'effet de DISP. Bien qu'on puisse employer TRACE et DISP simultanément, appliquez DISP sans TRACE afin de bien mettre en évidence ses effets.

```
10 DSPN
20 DSPI
100 PRINT "ENTREZ UN NOMBRE DE 1 A 15"
110  INPUT N
120  IF N = 1 THEN  PRINT "UNO";
130  IF N = 2 THEN  PRINT "DOS";
140  IF N = 3 THEN  PRINT "TRES";
150  IF N = 4 THEN  PRINT "CUATRO";
160  IF N = 5 THEN  PRINT "CINCO";
170  IF N > 5 THEN  GOTO 100
180  FOR I = 1 TO N
190 PRINT " *";: REM AFFICHAGE DE N ASTERISQUES
200 NEXT I
210 CALL - 936: REM EFFACER L'ECRAN
215 NODSPN
220 GOTO 100
```

RESTRICTIONS AUX MODES IMMÉDIAT ET PROGRAMMÉ

Vous pouvez employer la plupart des instructions Basic en mode différé ou en mode immédiat. Certaines ne sont légales qu'en mode différé, d'autres qu'en mode immédiat. Le tableau 4.2 liste les restrictions portant sur l'usage des instructions avec l'Integer Basic et le tableau 4.3, celles relatives à l'Applesoft.

Mode différé	Mode immédiat
END	AUTO
FOR	CLR
GOSUB	CON
INPUT	DEL
NEXT	HIMEM:
RETURN	LOAD
	LOMEM:
	MAN
	NEW
	RUN
	SAVE

Tableau 4.2. — Instructions en Integer Basic à usage restreint.

Mode immédiat uniquement
DATA
DEF FN
GET
INPUT
ON ERR GOTO
RESUME

Tableau 4.3. — Instructions en Applesoft à usage restreint.

CHAPITRE 5

L'UNITÉ DISK II

L'unité à disquettes est l'un des composants essentiels d'un système informatique. Elle donne accès presqu'instantanément à n'importe quelle donnée parmi une masse importante d'informations. L'unité Disk II de l'Apple II peut stocker plus de 100 000 caractères de données sur une seule disquette. C'est environ le double de ce que stockent 48K de RAM. De plus, lorsque l'ordinateur est débranché, toutes les informations sur disquettes restent intactes.

À PROPOS DES DISQUES

Les disques mémorisent les informations sous une forme magnétique, de la même façon que les bandes en cassette. La plus grande différence réside dans le fait que le disque est rond, comme un disque classique de musique. Il est mis en rotation, comme lui. Dans l'unité à disques se trouve une *tête* capable de lire et d'enregistrer des informations. L'ordinateur peut placer la tête en n'importe quelle position du disque. Cette faculté est appelée *accès aléatoire*. Ainsi, le disque est un *support de mémorisation à accès aléatoire*. Un programme spécial, appelé *système d'exploitation disques,* ou DOS (de l'anglais : « Disk Operating System ») commande toutes les opérations avec le disque. Il existe plusieurs sortes de disques.

Disques durs

Les disques durs sont rigides, et recouverts d'une substance magnétique. Ils stockent typiquement de 5 à 10 méga-octets de données (un méga-octet vaut un million d'octets). La plupart des disques durs sont amovibles, c'est-à-dire que le disque n'est pas solidaire de l'unité à disques et peut être changé. Le prix d'un disque dur est d'environ 800 F l'un ; une unité à disques durs vaut entre 15 000 et 60 000 F. La figure 5.1 montre un système-type à disque dur.

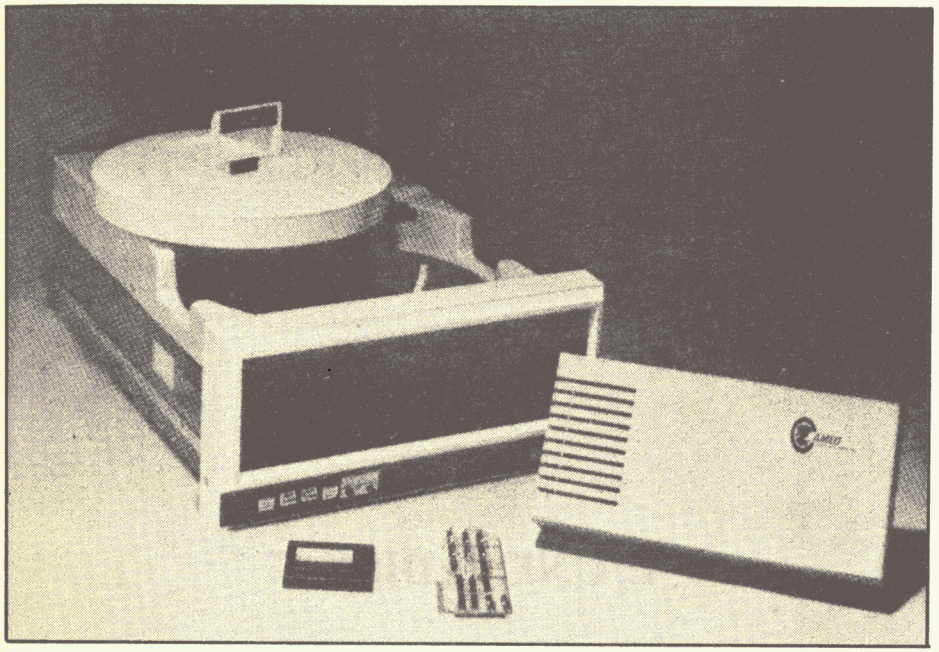


Photo courtesy of Cameo Electronics

Fig. 5.1. — Système-type à disque dur. ***(Doc. Cameo Electronics.)***

Disques Winchester

Les disques Winchester (fig. 5.2), recourent à une technologie spéciale qui multiplie par six à dix la capacité de stockage de chaque disque. Les disques Winchester sont extrêmement sensibles à la poussière et la saleté, même à la fumée de cigarette. Pour qu'ils restent très propres, ils sont enfermés dans l'unité à disques et ne peuvent être changés. Une telle unité vaut de 14 000 à 45 000 F.

Disquettes

Les disquettes sont le type de disques le plus populaire. Une disquette consiste en un disque circulaire en vinyle présenté dans une enveloppe en plastique rigide. Celle-ci protège la disquette des dommages que pourraient causer sa manipulation et son usage. La disquette peut entrer en rotation librement dans son enveloppe. Des fenêtres, dans celle-ci, permettent à la tête d'accéder à sa surface et au mécanisme d'entraînement d'accomplir son rôle. La disquette ne devra jamais être extraite de son enveloppe. La figure 5.3 montre à quoi ressemble une disquette.

Les disquettes sont aussi appelées « floppy disks » et existent en deux diamètres : 8 pouces (environ 20 cm) et 5 1/4 pouces (environ 13 cm), illustrés tous deux par la figure 5.4. L'unité Disk II utilise des 5 1/4 pouces, encore appelées *mini-disquettes,* ou « mini-floppy disks ». Chaque Disk II peut stocker plus de 100 000 octets par disquette.

COMMENT LES DONNÉES SONT STOCKÉES SUR DISQUETTES

Familiarisez-vous avec les diverses facettes du processus stockage des informations sur disquettes. Ce processus répond à plusieurs actions coordonnées.

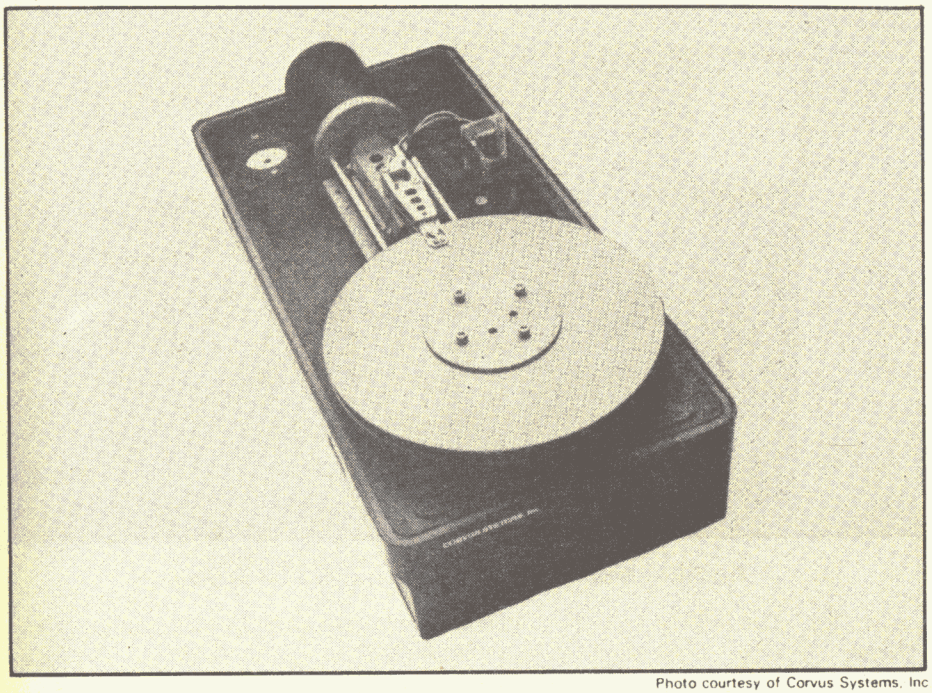


Photo courtesy of Corvus Systems, Inc

Fig. 5.2. — Unité à disques Winchester. ***(Doc. Corvus Systems.)***

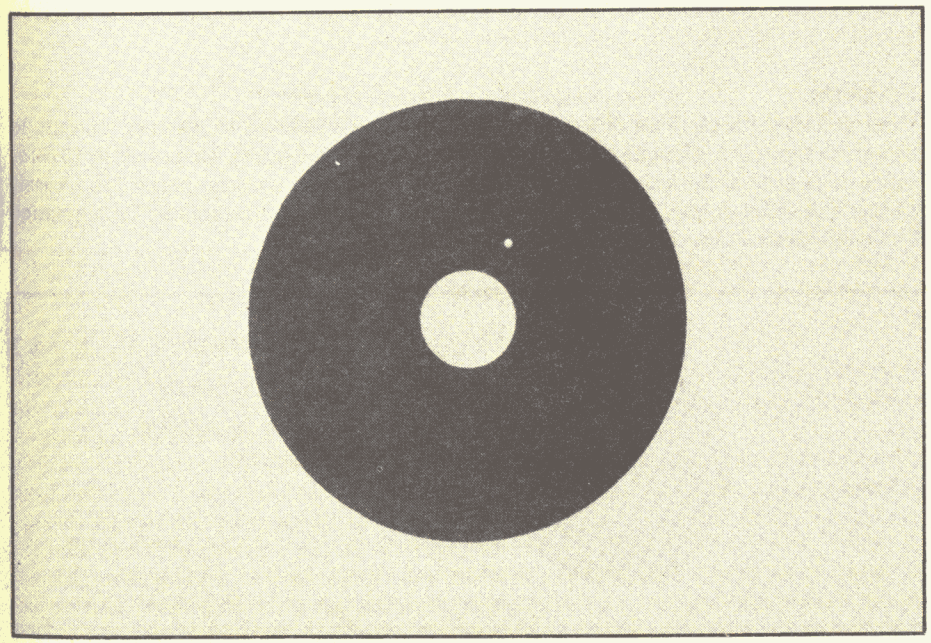


Fig. 5.3. — Une disquette extraite de son enveloppe.

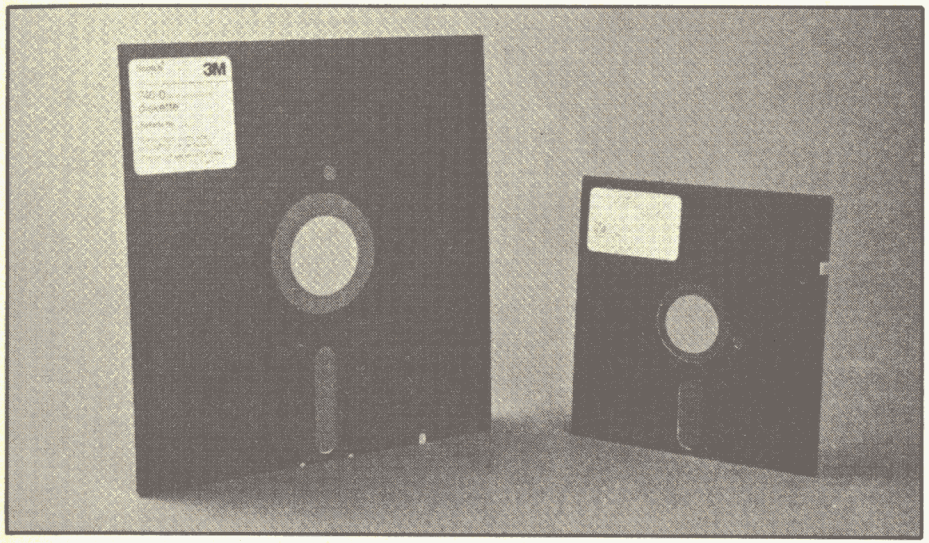


Fig. 5.4. — Deux disquettes, de 8 pouces et de 5 1/4 pouces.

Pistes

Pour accéder à un quelconque octet parmi les 100 000 stockés par une disquette, le DOS de Apple II divise la disquette en 35 pistes numérotées de 0 à 34. Ces pistes peuvent être comparées au sillon d'un disque de musique, mais on ne peut pas les voir et elles sont indépendantes, c'est-à-dire qu'il s'agit de pistes concentriques, comme le montre la figure 5.5.

Secteurs

Pour accélérer l'accès à un octet, le DOS divise chaque piste en 16 *secteurs,* comme le montre la figure 5.6. Chaque secteur contient exactement 256 octets. Disposant de la référence de la piste et du secteur, le DOS n'a plus qu'à extraire un octet parmi 256. Aussi, l'accès à un octet donné, ou à une série d'octets est-il quasiment instantané. La puissance réelle de l'unité à disquettes est ainsi exploitée.

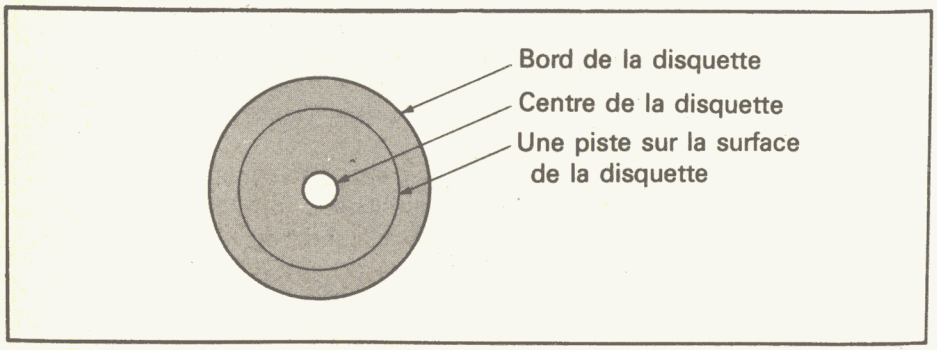


Fig. 5.5. — Les pistes de la disquette.

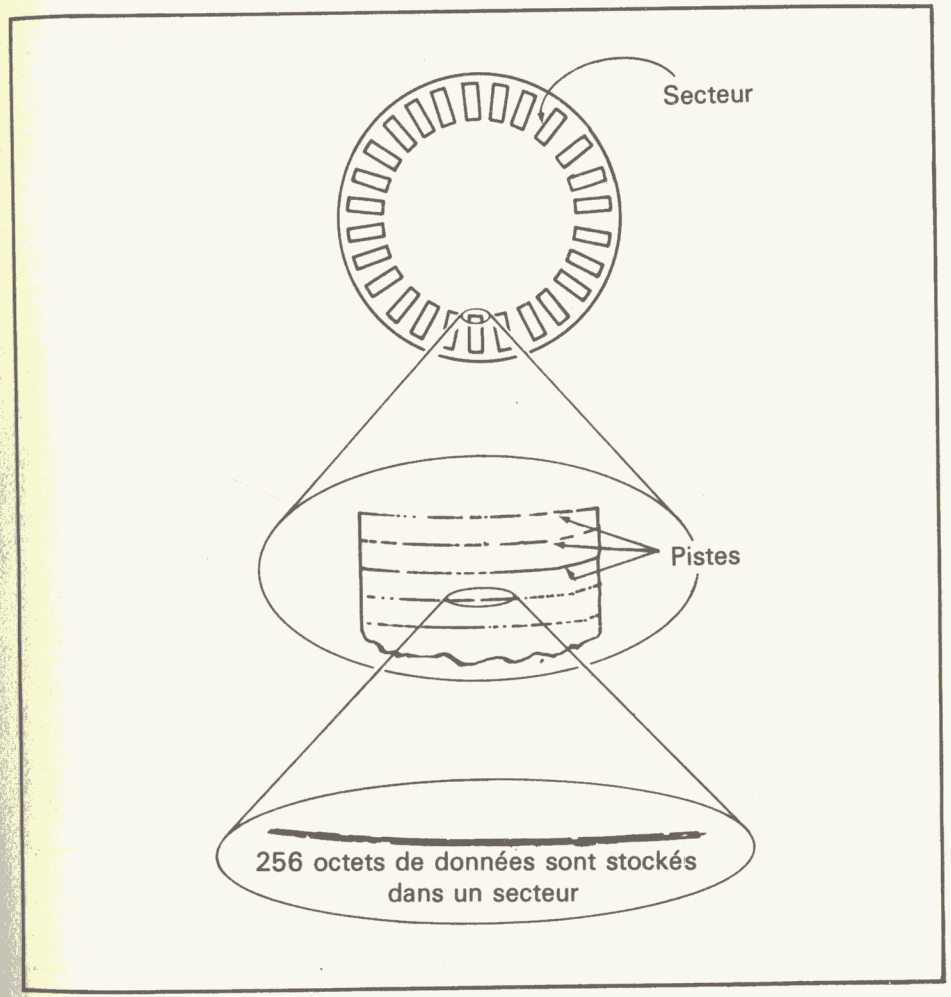


Fig. 5.6. — La surface enregistrée d'une disquette.

LOCALISATION DES PISTES ET SECTEURS

Trouver une piste sur une disquette est simple : l'unité déplace la tête sur la disquette, jusqu'à ce que la piste soit atteinte, exactement comme vous le faites pour trouver le passage voulu d'un enregistrement de musique sur un disque.

Trouver le bon secteur est un peu plus difficile. Deux méthodes sont communément employées pour localiser un secteur sur une disquette. Toutes deux recourent à une *perforation,* appelée un *index,* dans l'enveloppe de la disquette. Sur la plupart des disquettes, cet index est situé près de la grande découpe centrale de l'enveloppe. Lorsque la disquette tourne, un trou (ou des trous) présents sur la disquette elle-même passent sous cette perforation dans l'enveloppe. Un faisceau de lumière, créé dans l'unité, les traverse alors et atteint un capteur lorsque ces trous sont alignés. L'ordinateur est ainsi alerté et calcule alors, à partir de cette information, les positions des secteurs.

Les deux méthodes employées pour localiser les secteurs d'une disquette sont appelées *sectorisation matérielle* et *sectorisation logicielle.*

Sectorisation matérielle

Les disquettes à sectorisation matérielle comprennent plusieurs perforations, comme le montre la figure 5.7. Chacune indique la position d'un secteur. Un trou supplémentaire marque l'emplacement du premier secteur. L'ordinateur localise les secteurs en comptant ces trous à partir du premier.

Sectorisation logicielle

Les disquettes à sectorisation logicielle n'ont qu'un seul index (fig. 5.8). Il marque le premier secteur. La position des suivants est calculée à partir de la vitesse de rotation de la disquette. L'unité Disk II utilise des disquettes à sectorisation logicielle.

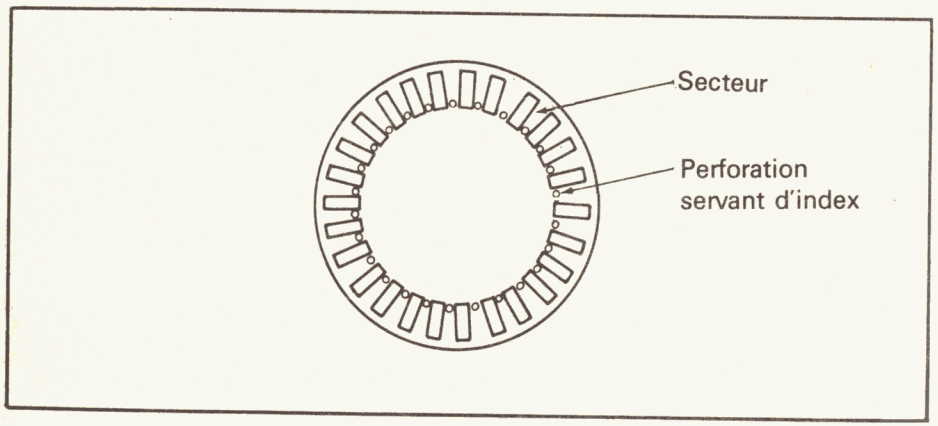


Fig. 5.7. — Disquette à sectorisation matérielle.

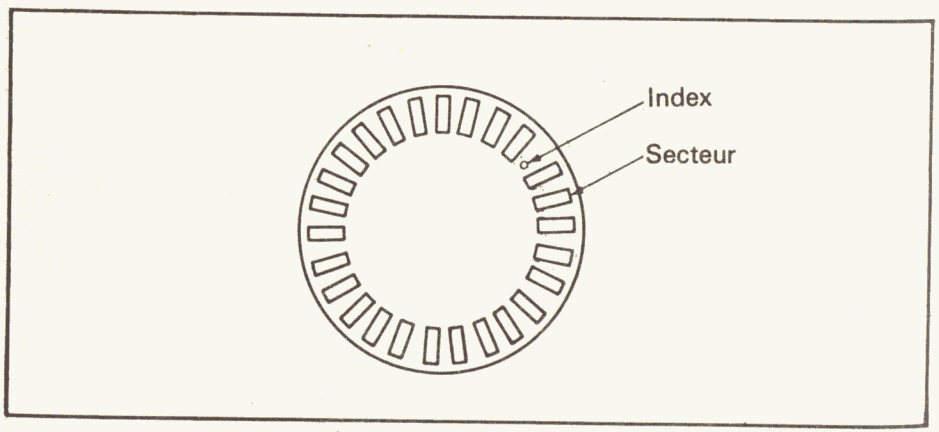


Fig. 5.8. — Disquette à sectorisation logicielle.

PROTECTION DE L'ÉCRITURE

Vous trouverez aussi une encoche sur un côté de l'enveloppe. Elle sert à autoriser ou interdire tout nouvel enregistrement sur la disquette. Sur les disquettes de 8 pouces, cette encoche est une *protection de l'écriture :* l'ordinateur ne pourra pas effectuer un enregistrement sur la disquette si cette encoche est *présente* sur l'enveloppe. Avec les disquettes de 5 1/4 pouces, cette encoche est une *autorisation d'écriture* car l'ordinateur ne pourra enregistrer la disquette *que si* cette encoche existe. Certaines disquettes, telles que la disquette « Système maître » fournie avec Disk II, sont protégées de façon permanente. Elles ne disposent pas de cette encoche. Les enveloppes à encoche peuvent être protégées ; il suffit de recouvrir l'encoche d'un morceau de bande adhésive (fig. 5.9).

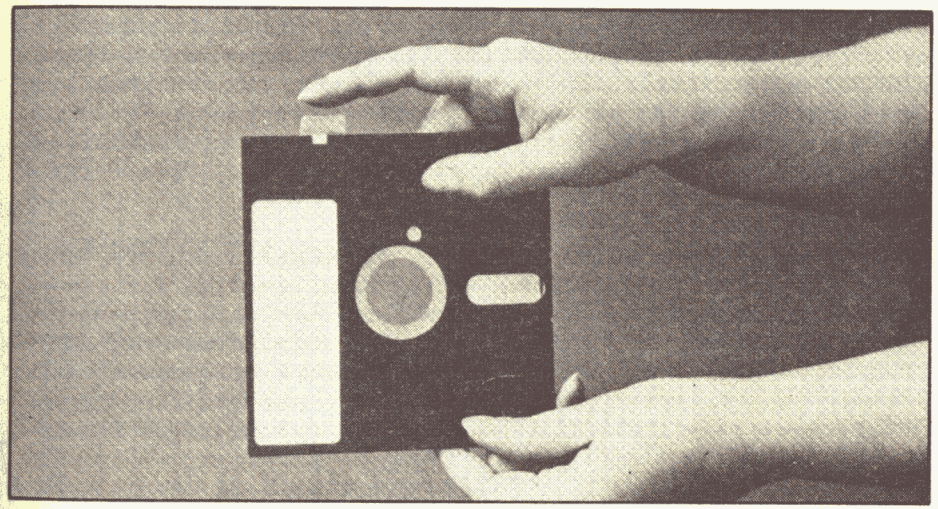


Fig. 5.9. — Protection en écriture d'une disquette 5 1/4 pouces.

LE SYSTÈME D'EXPLOITATION DISQUES

Toutes les opérations relatives aux disques sont contrôlées par un programme spécial appelé *système d'exploitation disques,* ou DOS, de « Disk Operating System ». Le Basic transmet au DOS toutes les demandes d'opérations impliquant la disquette. Le DOS retourne les résultats au Basic.

VERSIONS DU DOS

Il existe plusieurs versions du DOS. Le DOS 3.3 est la plus récente ; ce chapitre le décrit. La différence majeure entre les DOS 3.3 et 3.2.1, la version antérieure, réside dans le nombre de secteurs établis sur la disquette. Le DOS 3.2.1 dispose de 13 secteurs, alors que le 3.3 en a 16. *Apple Computer Inc.* a réalisé un programme spécial convertissant les disques du DOS 3.2.1 au DOS 3.3.

INITIALISATION DES DISQUES

Avant d'employer une disquette, Apple II doit l'*initialiser*. Cette initialisation consiste à effacer tout ce qu'elle contient et à placer une copie du DOS sur les pistes 0, 1 et 2. Nous donnerons plus loin les instructions d'initialisation du Disk II.

FICHIERS

Les données sont enregistrées comme des *fichiers*. Un fichier peut posséder toute longueur acceptable par la disquette. Chaque fichier dispose de son nom. Les informations stockées peuvent représenter du texte, un programme, une image graphique. Les divers types de fichiers sont discutés plus loin.

LE RÉPERTOIRE

Le nom de tous les fichiers ou programmes stockés sur disquette est mémorisé dans la disquette sous forme de répertoire (ou « directory », en anglais). Cette table est située sur la piste 17 de la disquette. La première entrée de la table est dans le secteur 15, la dernière, dans le secteur 1. Cette table peut mémoriser jusqu'à 84 fichiers.
Associé au nom de chaque fichier se trouve un code indiquant le type de données couvertes par le fichier, le nombre de secteurs occupés et la position du secteur d'un autre fichier appelé liste *pistes-secteurs*.

LISTE PISTES-SECTEURS

La liste pistes-secteurs contient des paires d'octets spécifiant l'adresse de la piste et du secteur de chaque secteur occupé par l'enregistrement d'un fichier. Chaque paire d'octets est appelée un *lien*. Le premier lien d'une liste pistes-secteurs donne l'adresse du secteur suivant utilisé par cette liste même. Celle-ci peut occuper autant de secteurs que nécessaire. Le second lien est l'adresse du premier secteur du fichier, cette fois ; le troisième lien adresse le secteur suivant du fichier, et ainsi de suite. Un lien à zéro marque la fin de la liste pistes-secteurs.

Le processus de stockage sur disque

Le DOS commande tous les mouvements de données de, et vers la disquette. Lorsque vous enregistrez un fichier, voici ce qui se passe :

1. Le DOS recherche le nom de ce fichier dans le répertoire. (Votre programme attribue un nom à ce fichier afin de l'identifier, comme on l'a vu.)
2. Si ce nom est trouvé, le DOS lit les 256 octets de son secteur et les range dans une zone de la mémoire appelée une zone *tampon,* ou encore un *buffer,* selon la terminologie anglo-saxonne. Si le nom n'est pas trouvé, ou si ce secteur du fichier n'a jamais été enregistré auparavant, le DOS remplit le buffer de zéros.
3. Les données à enregistrer, jusqu'à 256 octets, sont copiées de la mémoire centrale dans le buffer. Si moins de 256 octets de données sont enregistrés, les données antérieures subsistent.
4. Si, et seulement si 256 octets sont inscrits dans le buffer, le contenu de ce buffer est retranscrit dans le secteur approprié de la disquette.
5. Le processus (étapes 3 et 4) est répété jusqu'à ce que toutes les données du fichier aient été copiées dans le buffer et retranscrites dans la disquette.
6. Après quoi, le DOS met à jour la liste pistes-secteurs et la table des matières.

Notez que, sauf si le nombre d'octets du fichier est divisible par 256, le dernier bloc d'octets ne remplira pas le buffer et les étapes 4 à 6 ne se produiront pas. C'est pourquoi

une commande du Disk II appelée CLOSE (clore) oblige les données se trouvant dans le buffer à passer dans le fichier de la disquette, après quoi la liste pistes-secteurs et la table des matières sont mises à jour. Ce processus est appelé *clore la liste*. Si on omet de le faire, on risquera de perdre des données. N'oubliez jamais de clore une liste. La commande CLOSE est discutée plus loin.

ERREURS

Un disque peut introduire deux types d'erreurs : les erreurs *matérielles* et les erreurs *logicielles*.
Une erreur matérielle se manifeste lorsque la surface du disque est abîmée, ou présente un défaut physique. Dans ce cas, la tête de lecture-écriture de l'unité pourra aussi être abîmée et, à son tour, pourra endommager d'autres disques. C'est pourquoi on conseille de traiter les disques avec précaution.
Une erreur logicielle se produit lorsque la piste table des matières est surchargée par des données incorrectes. Ceci survient le plus fréquemment lorsqu'un ou plusieurs fichiers de données ont été enregistrés mais n'ont pas été clos, ou lorsqu'un disque différent est placé dans l'unité et que le fichier est alors clos. Pour pleinement apprécier ce qu'il advient ensuite, il faut l'expérimenter.

L'APPEL DE DISK II

Pour qu'Apple II reconnaisse une commande de disque, le programme spécial appelé *système d'exploitation disques,* ou DOS, doit être dans sa mémoire. Si vous disposiez de beaucoup de temps, vous pourriez frapper ce DOS au clavier. Mais il est plus facile d'*appeler ce DOS,* ce qu'en jargon franglais des informaticiens, on nomme souvent « bouter le DOS » (« booting DOS », ou « booting the disk »), ou encore de *charger le DOS ;* on dit encore qu'on *appelle Disk II*. Charger le DOS consiste à lire une copie du DOS sur une disquette et à la ranger en mémoire.

COMMENT APPELER LE DOS

La méthode de chargement du DOS dépend de la configuration de votre système et du langage servant à initier ce chargement. Chaque méthode suppose que l'unité à disquettes repérée DRIVE 1 (unité 1) est reliée au connecteur 6.
Insérez la disquette « System Master » dans l'unité et rabattez le volet. Ce que vous ferez ensuite dépendra du type de système en votre possession. A l'issue d'un « appel » réussi, l'écran offrira l'un des aspects représentés dans la figure 2.5.

Autostart

L'auto-démarrage (« autostart ») est le mode de chargement le plus simple. Comme son nom l'indique, le chargement démarre automatiquement, mais pour cela, la carte « Autostart Monitor » doit être implantée dans l'ordinateur. Vous le vérifierez en mettant Apple II sous tension, l'unité Disk II lui étant connectée : si le voyant IN USE de cette unité s'allume, et si des bruits divers traduisant que la mécanique s'est mise en route se font entendre, c'est que vous disposez du moniteur « Autostart ».
Pour appeler le DOS avec le moniteur autostart, mettez simplement Apple II sous tension.

Chargement à partir du moniteur

Lorsque le caractère d'appel du moniteur (*) apparaît sur l'écran, le moniteur en langage

assembleur de Apple II attend vos commandes. Voici les diverses méthodes à employer pour charger le DOS.

Chargement par Jump moniteur

Frappez la lettre C, suivie par le numéro de connecteur auquel votre unité à disquettes est reliée (6, en standard), puis deux zéros et enfin, la lettre G. La commande complète est :

```
*C600G
```

où C600 est l'adresse mémoire du programme pilotant le chargement à partir de l'unité reliée au connecteur numéro 6 ; G est la commande transférant le contrôle à ce programme. Pressez maintenant RETURN. Le voyant de l'unité à disquettes devrait s'allumer et celle-ci démarrer (bruits divers).

Chargement par CTRL-K et CTRL-P moniteur

Les autres commandes du moniteur que vous pouvez utiliser sont CTRL-K et CTRL-P. Pour charger le DOS à partir d'un moniteur employant l'une de ces deux commandes, frappez le numéro du connecteur (le 6, en standard), puis frappez CTRL-K ou CTRL-P. L'ordre CTRL-K ou CTRL-P ne sera pas affiché sur l'écran.
Après quoi, pressez la touche RETURN.

Chargement à partir de l'Integer Basic ou de l'Applesoft

Les mêmes commandes sont reconnues par l'Integer Basic et l'Applesoft.

Chargement du Basic avec les commandes PR# et IN#

Après que le caractère d'appel du Basic (> en Integer Basic, ou] en Applesoft) apparaisse, frappez les lettres PR ou IN, puis le symbole dièze (#), et finalement le numéro du connecteur du disque contenant le programme à charger. La commande ressemble à :

```
PR#6
```

ou

```
IN#6
```

Pressez ensuite la touche RETURN.

Chargement avec le « Système Langage » Apple

Si le système langage (« Language System ») est installé dans votre Apple II, les procédures ci-dessus pourront ne pas s'appliquer. Il faut deux disques pour charger le DOS en versions 3.2.1, et 3.2, ou moins. Mais le DOS 3.3 se charge comme indiqué ci-dessus.
Pour appeler les DOS 3.2.1 et 3.2, ou de références moindres, insérez la disquette labellée « BASICS : Integer and Applesoft II » (fournie avec le « Langage Système ») à la place de la « System Master Diskette », puis passez par l'une des procédures de chargement décrites ci-dessus (autostart, PR#6, etc.). Après un appel réussi, l'écran affichera :

```
INSERT BASIC DISK AND PRESS RETURN
```

Cette formule est ambiguë. Vous devrez réellement introduire une disquette DOS (la

« System Master Diskette » fera l'affaire, mais vous pourrez employer toute autre disquette ayant été initialisée). Après avoir introduit cette seconde disquette, pressez RETURN ; le chargement se fera alors normalement et vous verrez apparaître l'un des écrans de la figure 2.5.

PRATIQUE DE LA COMMANDE DES DISQUES

Le système d'exploitation des disques de l'Apple II interprète et exécute les commandes relatives aux disques. De nombreuses commandes acceptent ou demandent des paramètres additionnels qui complètent la définition de l'opération à exécuter. Voici quelques-unes de ces commandes de base.

CATALOG

La commande CATALOG sort le listage des noms des fichiers enregistrés dans le répertoire de la disquette, et cela sur le périphérique de sortie en service, généralement l'écran. Le catalogue indique d'abord le numéro de volume de la disquette. (Ce numéro de volume sera discuté plus loin.)
Pour chaque fichier en disquette, CATALOG donne le *type* de données, indique si le fichier est *verrouillé,* le *nombre de secteurs* qu'il occupe, et son *nom.* Un catalogue-type ressemble à ce que montre la figure 5.10.

```
*I 002 HELLO
*I 053 APPLE-TREK
*I 018 ANIMALS
*B 009 UPDATE 3.2.1
*I 014 COPY
*I 009 COLOR DEMO
*I 053 BRICK OUT
*I 026 SPACE WAR
*I 050 THE INFINITE NO. OF MONKEYS
*I 051 COLOR SKETCH
*I 053 SUPERMATH
*I 026 APPLEVISION
*I 017 BIORHYTHM
*I 027 PINBALL
```

Fig. 5.10. — Un catalogue-type d'une disquette.

Types de données

Le type de données d'un fichier est représenté par un code sur une seule lettre, apparaissant dans la colonne de gauche du catalogue. Les codes sont données dans le tableau 5.1.

Fichiers verouillés

Les fichiers verrouillés ne peuvent recevoir de nouvelles inscriptions, ni être effacés. Le

catalogue indique qu'un fichier est verrouillé en faisant précéder son code par un astérisque (*). Si, à la place, c'est un espace qui apparaît, le fichier n'est pas verrouillé. Les fichiers verrouillés sont discutés plus loin dans ce chapitre.

Code	Signification
A	Programmes en Applesoft
B	Fichiers images en binaire
I	Programmes en Integer Basic
T	Fichiers de texte

Tableau 5.1. — Les codes des types de données.

Numéros des secteurs

Le nombre de secteurs occupés par un fichier est indiqué par un nombre de trois chiffres. Le plus petit fichier (plein) occupe un secteur. Si un fichier occupe plus de 255 secteurs, ce nombre est remis à zéro et on recommence le compte, ce qui n'affecte en aucune façon la longueur réelle de ce fichier.

Noms de fichiers

Le DOS d'Apple II demande que chaque fichier soit désigné par un nom. Voici les règles qui gouvernent l'attribution des noms :

1. Le nom d'un fichier doit comprendre de 1 à 30 caractères. Les caractères excédentaires seront ignorés.
2. Il doit commencer par une lettre.
3. Tous les caractères disponibles sur le clavier sont autorisés, à l'exception de la virgule.

Vous pouvez employer des caractères sans impression (tels que ceux créés avec CTRL), mais ils n'apparaîtront pas dans le listage du catalogue. Ce sera utile si vous voulez éviter que d'autres ne connaissent les noms réels. (Mais n'oubliez pas quels caractères sans impression vous avez adoptés !)

Utilisation de la commande CATALOG

Pour utiliser la commande CATALOG, frappez simplement ce mot (à la condition que le DOS ait été chargé), soit :

```
CATALOG
```

Le résultat devrait donner quelque chose comme le montre la figure 5.10.
Si le nombre de lignes du catalogue excède 20, l'ordinateur affichera les 21 premières et attendra que vous pressiez n'importe quelle autre touche (excepté RESET, CTRL et SHIFT) pour afficher les 21 lignes suivantes. Cette pause vous donne le temps de lire les noms des fichiers.

LOAD

La commande LOAD (« charger ») charge un programme de la disquette en mémoire.

Vous devez spécifier le nom du fichier (du programme) à charger. Cet exemple charge le programme appelé COLOR DEMOSOFT.

```
LOAD COLOR DEMOSOFT
```

Si le nom spécifié ne se trouve pas dans le répertoire de la disquette, vous obtiendrez un message d'erreur FILE NOT FOUND (« fichier non trouvé »).
Si le fichier demandé est sur la disquette, le DOS teste le type de données. S'il ne s'agit pas d'un programme, vous recevrez le message d'erreur FILE TYPE MISMATCH « type de fichier en désaccord »).
Mais si tout va bien, LOAD effacera le programme précédant en mémoire à lecture-écriture et recopiera le programme de la disquette dans la mémoire à lecture-écriture. Après le retour du curseur et du caractère d'appel, vous pourrez lister, modifier ou lancer le programme que vous venez de charger.

VERSION DISQUE DE LA COMMANDE RUN

Fréquemment, après émission de l'ordre LOAD, vous ferez immédiatement un RUN. Vous pouvez abréger ce processus en deux étapes en spécifiant un fichier avec une commande RUN. L'ordre LOAD devient implicite, puisque le fichier doit d'abord être chargé avant de tourner. En voici quelques exemples :

```
RUN PROGRAM 2
RUN SPOT RUN
RUN COLOR DEMOSOFT
```

POUR SPÉCIFIER LE NUMÉRO DE L'UNITÉ À DISQUETTES

De nombreux DOS vous permettront de spécifier l'unité à disquettes à employer. Deux paramètres interviennent : l'*unité à disquettes* (« Drive »), et le *numéro du connecteur*. Deux unités à disquettes peuvent être connectées à un unique contrôleur. Pour sélectionner l'une d'elles, ajoutez une virgule et les deux caractères, D1 et D2 selon le cas, comme suit :

```
LOAD CHARGE, D2
RUN CELA, D1
CATALOG, D2
```

Après avoir spécifié une fois une unité, les autres commandes sans spécification d'unité s'adresseront, par défaut, à la même. Sauf si vous spécifiez une autre unité à disquettes. L'unité choisie par défaut est toujours la dernière employée. Si aucune n'a encore été utilisée, l'unité 1 entrera en service.

SPÉCIFICATION DU CONNECTEUR

Les cartes « contrôleurs » du Disk II sont enfichées dans des connecteurs femelles de l'Apple II. Il existe huit connecteurs femelles, mais le connecteur 0 ne peut être utilisé pour les unités à disquettes. Il reste donc un maximum de sept connecteurs. Puisque chaque contrôleur peut supporter deux unités, vous pourrez disposer d'un maximum de 14 unités à disquettes connectées à votre Apple II.
Si vous connectez plus de deux unités Disk II à votre Apple, vous ne pourrez vous référer à D3 ou D4, etc. A la place, vous devrez recourir à un autre paramètre pour choisir le bon

contrôleur. Ce paramètre est celui du connecteur (« slot », en anglais) dans lequel est enfichée la carte « contrôleur ».
Pour employer ce paramètre, ajoutez une virgule, puis la lettre S et le numéro du connecteur (1 à 7) à votre commande pour disquette, par exemple :

```
CATALOG, S5
LOAD GESTION, S6
RUN APRES, S3
```

Le numéro de connecteur repris par défaut sera celui qui est intervenu lors du chargement du DOS.
Après que vous ayez appelé un paramètre de connecteur une fois, ce sera lui qui sera choisi par défaut ensuite, jusqu'à ce que vous en spécifiez un autre.
Vous pouvez utiliser à la fois des références de l'unité à disquettes et du connecteur dans une unique commande. En spécifiant un connecteur de 1 à 7 et une unité, 1 ou 2, vous pourrez faire intervenir n'importe quelle unité Disk II connectée à votre ordinateur. Les paramètres, unité et connecteur, peuvent intervenir dans n'importe quel ordre. Ainsi, les deux commandes suivantes sont équivalentes :

```
CATALOG,D2,S5
CATALOG,S5,D2
```

Toutes deux mettront en service la disquette de l'unité numéro 2 reliée au contrôleur du connecteur 5.

Problèmes avec le paramètre connecteur

Si vous sélectionnez un connecteur auquel aucun contrôleur n'est connecté, l'ordinateur va se verrouiller. Il attend d'un contrôleur inexistant un signal indiquant qu'il est prêt. Afin de retrouver votre programme intact, pressez alors RESET. Si vous obtenez un caractère d'appel Basic (> ou]), tout devrait bien se passer. Si c'est un astérisque moniteur (*) qui revient, frappez 3DOG et pressez RETURN. Si cela ne suffit pas, vous perdrez votre programme car il vous faudra recharger le DOS.

SPÉCIFICATION DU VOLUME

Le *volume* est un autre paramètre que vous pourrez spécifier avec chaque commande DOS permettant la spécification du connecteur et de l'unité à disquettes, sauf CATALOG. Volume vous servira à vous assurer que la bonne disquette est en place dans l'unité sélectionnée.
La commande CATALOG ignore la référence de volume. Lorsqu'elle liste le répertoire de la disquette, le numéro de volume de la disquette est la première information listée.
Pour employer le paramètre volume, ajoutez une virgule, puis la lettre V suivie par le numéro du volume, comme suit ;

```
LOAD LIVRE, V191
```

Si le numéro du volume spécifié n'est pas le même que le numéro de volume assigné à la disquette lorsqu'elle a été initialisée, le DOS retournera à un message d'erreur VOLUME MISMATCH (« désaccord de volume »).
Les nombres références de volumes doivent se situer dans la gamme 1 à 254. Si vous ne spécifiez pas le numéro de volume, ou si vous spécifiez le numéro 0, le paramètre volume sera ignoré.

Vous pouvez vous servir du paramètre volume en conjonction avec le paramètre connecteur ou avec le paramètre unité à disquettes, ou les deux, dans n'importe quelle séquence. En voici des exemples :

```
LOAD LIVRE, D2, V24
RUN PAIE, S6, V111
```

AUTRES COMMANDES DISK II

Vous savez maintenant comment lire le contenu d'un disque avec la commande CATALOG, et comment charger et lancer des programmes. Vous êtes prêt à ranger vos propres programmes sur les disques, à l'aide des commandes INIT, SAVE, DELETE, LOCK, RENAME et VERIFY, que nous allons décrire.

INIT

Avant que vous puissiez écrire sur une disquette, rappelez-vous que vous devez l'*initialiser*. Lorsqu'une disquette est initialisée, tout ce qu'elle stockait est effacé, aussi assurez-vous que vous n'initialisez pas une disquette contenant des données que vous voulez conserver.
La commande INIT sauvegarde en disquette tout programme se trouvant en mémoire lorsqu'elle est émise. Ce programme devient le programme de *salutations* (« greeting », en anglais... et dans le jargon des informaticiens), qui sera automatiquement lancé chaque fois que vous appellerez la disquette. Ce programme peut être aussi simple ou complexe que vous le désirez. Supposez, par exemple, que votre disquette contienne un fichier d'adresses postales. Vous pourrez employer cette liste comme programme de salutations. Aussi, dès que vous introduirez la disquette dans l'unité et l'appellerez, ce programme sera chargé et lancé. Un autre exemple de programme de salutations consiste en une instruction NEW et une instruction END. Chaque fois que vous appellerez le disque, vous obtiendrez un caractère d'appel Basic (> ou]).
Cependant, un bon programme de salutations devrait vous fournir quelqu'indication sur la disquette. Un programme type ressemblera au suivant :

```
100 TEXT
200 CALL - 936
300 PRINT "VOICI MA PREMIERE DISQUETTE"
400 PRINT
500 PRINT "INITIALISEE LE 1-4-82"
600 PRINT
700 PRINT "SUR UN SYSTEME 48K AVEC DOS 3.3"
800 END
```

Le nom de fichier du programme de salutations doit être spécifié lorsque vous employez la commande INIT. Il vous appartient de vous assurer qu'il y a toujours un programme de ce nom sur la disquette. Si vous effacez le programme de salutations (nous décrirons comment plus loin), vous verrez apparaître le message FILE NOT FOUND (« fichier non trouvé ») chaque fois que vous ferez appel à la disquette. La seule façon de stopper le message d'erreur consiste à placer le programme sur la disquette avec le nom de fichier du programme de salutations. Ce pourra être difficile si vous ne le connaissez pas ou ne pouvez vous en souvenir, car il n'existe aucune façon de déterminer le nom du programme de salutations. La meilleure solution est la prévention. Spécifiez *toujours* ce nom quand vous initialisez une disquette. Le nom standard du programme de salutations est HELLO.

Utilisation de la commande INIT

Une commande INIT typique ressemble à :

```
INIT HELLO,S6,D1,V36
```

Ainsi que vous pouvez le supposer, le connecteur, l'unité à disquettes et le volume sont des spécifications facultatives. Si vous incluez un paramètre volume, le DOS *affectera* le volume spécifié à la disquette. INIT est la seule commande qui affecte un numéro de volume. Lorsque le paramètre volume est employé avec d'autres commandes DOS, le nombre spécifié doit être le même que le nombre attribué par INIT. Si vous omettez le paramètre volume, INIT lui attribuera le nombre 254.
Comme toujours, l'omission des paramètres connecteur ou unité à disquettes leur fera attribuer les valeurs par défaut, spécifiées par la commande DOS antérieure. Assurez-vous que vous savez quelle unité la commande INIT va sélectionner. Si vous n'en êtes pas certain, spécifiez-la !
Pour initialiser une nouvelle disquette, enlevez d'abord la « System Master Diskette » de l'unité et remplacez-la par une nouvelle disquette, vierge. Utilisez la commande NEW pour pouvoir placer en mémoire votre programme de salutations. Puis, tapez ce programme comme on l'a montré ci-dessus, ou un autre programme de votre cru. Une bonne idée consistera à tester son lancement avant de la ranger en disquette.
Attribuons le nombre 123 au numéro de volume. Frappez :

```
INIT HELLO,S6,D1,V123
```

Assurez-vous que la porte de l'unité à disquettes est refermée et pressez RETURN. Le voyant rouge de l'unité s'allume, accompagné de bruits divers.
Le processus entier demande environ deux minutes, aussi soyez patient. Après que le voyant se soit éteint, utilisez la commande CATALOG pour vérifier ce qui se trouve sur la disquette. Vous devriez obtenir :

```
DISK VOLUME 123

 I 002 HELLO
```

ou, en Applesoft :

```
DISK VOLUME 123

 A 002 HELLO
```

La lettre I signifie que le programme de salutations a été rédigé en Integer Basic et la lettre A, en Applesoft. Le nombre 002 indique que le programme occupe deux secteurs. Vous pourrez trouver un nombre différent si votre programme de salutations a une autre longueur.
Préparez une étiquette pour votre disquette. Ecrivez dessus le numéro du volume, avec toutes les informations utiles relatives à cette disquette. Otez-la de l'unité, appliquez-lui l'étiquette, et replacez-la dans l'unité. Si vous voulez l'appeler, vous pouvez maintenant le faire.

SAVE

Si vous avez exécuté ce qu'on vient d'indiquer, vous disposez maintenant d'une disquette fraîchement initialisée dans l'unité 1, connecteur 6 de votre Apple. Il vous reste probablement une copie en mémoire du message de salutations. Utilisez la commande LIST pour le vérifier. S'il n'y est pas, frappez :


```
LOAD HELLO
```

pour obtenir cette copie de votre disquette.
La commande SAVE sauvegarde, range un programme sur disquette. Pour ranger une autre copie de votre programme de salutations, frappez SAVE suivi par un nom de fichier. Pour cet exemple, on va adopter le nom de SALUT, mais vous avez le droit d'employer tout autre nom autorisé :

```
SAVE SALUT
```

Le disque devrait se mettre en marche. Lorsque l'exécution est terminée, le caractère d'appel du Basic et le curseur devraient n'apparaître. Utilisez la commande CATALOG pour vérifier le contenu de votre disque. Vous devriez obtenir quelque chose comme :

```
DISK VOLUME 123

 A 002 HELLO
 A 002 SALUT
```

Vous pourrez sauvegarder de cette façon tout programme Basic.
Si vous adoptez un nom de fichier déjà sur disquette, le programme que vous avez en mémoire, quel qu'il soit, remplacera celui portant le même nom et qui se trouvait déjà sur la disquette. Ainsi, l'ancien programme sera automatiquement effacé. C'est vrai tant que ces deux programmes sont dans une même version du Basic. Sinon, le nouveau programme ne sera pas sauvegardé et vous obtiendrez le message FILE TYPE MISMATCH (désaccord de type de fichier).

DELETE

Au bout d'un certain temps, vous aurez probablement accumulé sur la disquette de nombreux programmes qui ne vous servent plus. La commande DELETE servira à les supprimer.
Pour supprimer une copie du programme de salutations que vous venez juste de charger, utilisez n'importe laquelle des commandes suivantes (à moins que vous n'ayez référencé différemment l'unité à disquettes entre-temps) :

```
DELETE PROGRAMME SALUTATIONS, S6, D1, V12
DELETE PROGRAMME SALUTATIONS, V123
DELETE PROGRAMME SALUTATIONS
```

Souvenez-vous que vous pouvez recourir aux paramètres de connecteur, d'unité à disquettes et de volume dans un ordre quelconque, ou même pas du tout, si vous vous adressez à la dernière unité que vous avez utilisée.

LOCK

Certains programmes et fichiers doivent être conservés en permanence sur une disquette. Pour ce faire, le DOS comporte une technique de protection, appelée *verrouillage de fichier*. Verrouiller un fichier interdit qu'il puisse être accidentellement effacé, ou qu'on écrive par-dessus. Pour verrouiller un fichier (« to *lock* a file », en anglais), ajoutez la commande LOCK au nom du fichier, avec les paramètres facultatifs de connecteur, d'unité et de volume. La commande suivante verrouille le message de salutations :

```
LOCK HELLO
```

C'est effectivement une bonne idée que de verrouiller ce programme.
Des tentatives ultérieures pour effacer et ré-écrire sur un fichier verrouillé se solderont par un message d'erreur FILE LOCKED (« fichier verrouillé »).
Si le fichier verrouillé est un programme en Integer Basic ou en Applesoft, et si vous tentez de sauvegarder un programme portant le même nom mais rédigé dans un autre Basic, vous obtiendrez le message d'erreur FILE TYPE MISMATCH (« désaccord de type de fichier »).
Sur le catalogue d'une disquette, les fichiers verrouillés sont marqués par un astérisque (*) précédant le type de fichier.

UNLOCK

Si vous décidez d'écrire sur un fichier déjà verrouillé, ou de le supprimer, vous pouvez le déverrouiller avec la commande UNLOCK. La commande suivante déverrouille le message de salutation :

```
UNLOCK HELLO
```

Comme de coutume, les spécifications d'unité, connecteur et volume sont facultatives et peuvent intervenir dans n'importe quel ordre.

RENAME

Vous pouvez changer le nom de n'importe quel fichier en disquette. Une façon consisterait à changer ce fichier, l'effacer en disquette, puis le re-sauvegarder sous son nouveau nom. Mais il est plus simple d'employer l'ordre RENAME, qui agit pour n'importe quel fichier en disquette, quel que soit le Basic utilisé. RENAME fonctionne aussi bien pour des fichiers de texte que pour des fichiers binaires. En voici un exemple :

```
RENAME NOMANCIEN, NOMNOUVEAU
```

Le DOS *ne vérifiera pas* si le nouveau nom se trouve déjà sur la disquette. S'il s'y trouve déjà, vous serez à la tête de deux fichiers portant le même nom, ce qui sera source de confusion ; il vous sera difficile de vous en sortir.
Ne renommez pas le programme de salutations, sauf si vous sauvegardez sur la disquette un nouveau programme portant le même nom. RENAME ne modifiera pas le nom que le DOS recherche lorsque la disquette a été appelée.
Vous ne pouvez changer le nom d'un fichier verrouillé.
Vous pouvez spécifier l'unité, le connecteur et le volume dans n'importe quel ordre.

VERIFY

A l'occasion, vous voudrez vérifier qu'un fichier est intact. La meilleure façon de procéder consiste à utiliser la commande VERIFY. La commande suivante va vérifier le programme de salutations :

```
VERIFY HELLO,V123
```

Lorsque le DOS écrit chaque secteur d'un fichier, il calcule un nombre appelé *somme de test*. Cette somme de test se fonde sur la valeur numérique de chaque caractère du secteur, et est rangée avec le secteur. Lorsque vous émettez la commande VERIFY, le DOS recalcule la somme de test de chaque secteur du fichier et compare la valeur obtenue à celle qu'il avait rangée antérieurement. S'il y a désaccord, le DOS vous retourne le message d'erreur « I/O ERROR ».

Si toutes les sommes de test sont identiques, le DOS ne renvoie aucun message ; simplement, le caractère d'appel du Basic et le curseur réapparaissent.
Comme avec presque toutes les autres commandes du DOS, vous pouvez spécifier le connecteur, l'unité et le volume dans un ordre quelconque.

UTILISATION DES COMMANDES DU DOS DANS LES PROGRAMMES

Jusqu'à présent, nous avons frappé toutes les commandes du DOS au clavier. Mais les programmes Basic qui emploient des fichiers peuvent inclure des commandes du DOS. Pour introduire une commande du DOS dans un programme Basic, on se sert de l'ordre PRINT, préfixé par un code ASCII de quatre caractères. Employez CTRL-D pour créer ce préfixe. CTRL-D doit être le premier caractère sorti par l'ordre PRINT. Assurez-vous que le PRINT précédent ne se terminait pas par un point-virgule ou une virgule.
Parce que CTRL-D ne produit pas de caractère affichable, il est conseillé que vous documentiez chacune de ses apparitions avec une REM, par exemple :

```
1000 PRINT "RUN MENU" : REM IL Y A UN CTRL-D
     ENTRE LE 1ER ASTERISQUE ET LE R DE RUN
```

Vous pouvez simplifier cela en définissant une variable pour CTRL-D, puis en affichant la variable, suivie par la commande du DOS. On emploie couramment la variable D$, mais vous pourrez vous servir de n'importe quel autre nom. Le recours à la variable standard D$ dans tous vos programmes les rend compatibles entre eux et avec d'autres, provenant d'autres sources. Ajoutez une ligne telle que celle-ci dans vos programmes Basic :

```
10 D$ = "": REM ON A PLACE UN CTRL-D INVISIBLE
   DANS LES GUILLEMETS
```

Ou encore, et en Applesoft, vous pourrez faire :

```
10 D$ = CHR$ (4): REM C'EST UN CTRL-D
```

Désormais et quand vous appellerez D$, il sera aisé de voir qu'il s'agit de CTRL-D.
Essayez de lancer le programme suivant :

```
10 D$ = "": REM  CTRL-D
20  FOR I = 1 TO 10
30  PRINT D$;"CATALOG"
40  NEXT I
50  END
```

A moins que votre disquette ne contienne plus de 18 fichiers, vous devriez voir afficher 10 fois son catalogue, sans toucher à rien. Si elle dispose de plus de 18 fichiers, vous devrez presser une touche à chaque pause.

UTILISATION DES FICHIERS

Le DOS d'Apple supporte deux types de fichiers : les fichiers *séquentiels* et les fichiers à *accès aléatoire*. Tous deux contiennent des blocs de données, appelés *champs*. Un champ peut être d'un ou plusieurs caractères.

Fichiers séquentiels

Comme leur nom l'indique, les fichiers séquentiels ne peuvent être atteints que dans l'ordre séquentiel. Pour lire ou écrire dans le dernier champ du fichier, vous devez d'abord lire ou écrire tous les champs précédents. Le fichier séquentiel convient pour certaines applications.

Fichiers aléatoires

Les fichiers aléatoires sont plus simples d'usage que les séquentiels. Vous pouvez lire ou écrire dans n'importe quel champ, quelle que soit sa position. Ils constitueront la meilleure solution pour de nombreuses applications.

UTILISATION DES FICHIERS SÉQUENTIELS

Pour exploiter des fichiers séquentiels, vous devez apprendre quelques commandes supplémentaires du DOS : OPEN, CLOSE, READ et WRITE.

Ouverture de fichiers séquentiels

Les fichiers sur disque doivent être *ouverts* avant qu'on puisse y accéder. L'ouverture d'un fichier impose au DOS la recherche des informations le concernant : est-il ou non sur le disque, et si oui, où. La commande OPEN (« ouvrir ») réserve également une zone de la mémoire qui servira de *buffer* (élément tampon) pour le fichier. Le buffer vous permet d'accéder à une petite fraction du fichier sans actionner, pour chaque accès, l'unité à disquettes, ce qui est bien plus rapide. La commande OPEN se traduit par :

```
OPEN NOMDUFICHIER,S6,D2,V99
```

Si un tel fichier n'existe pas encore, le DOS va créer une nouvelle entrée dans le répertoire de la disquette.
Dans un programme, la commande OPEN devra être placée dans un ordre PRINT et préfixée par un caractère CTRL-D.
Le programme ci-dessous ne fait rien d'autre que créer un fichier séquentiel, nommé SEQUENTIEL, sur la disquette et dans l'unité en service (choisie par défaut) :

```
100 D$ = "":REM CRTL-D
200 PRINT D$;"OPEN SEQUENTIEL"
300 END
```

Nous appellerons cela le « *Programme 1* ».
Vous pouvez combiner les paramètres de connecteur, unité et volume comme bon vous semble :

```
100 D$ = "": REM CTRL-D
200 PRINT D$;"OPEN SEQUENTIEL,S6,D1,V123'
300 END
```

ou encore : `200 PRINT D$;"OPEN SEQUENTIEL,V123,S6"`

Remarquez que les paramètres interviennent *avant* la fermeture des guillemets, et que, d'autre part, ils sont séparés par des virgules.
Après exécution de ce programme, le fichier SEQUENTIEL sera dans le catalogue. Vérifions-le en tapant CATALOG. On devrait obtenir quelque chose comme cela :


```
DISK VOLUME 123

*A 002 HELLO
 T 001 SEQUENTIEL
```

Notez la présence de la lettre T avant le nombre de secteurs de SEQUENTIEL. T est le code qui précise que SEQUENTIEL est un fichier de texte, et non un programme en Integer Basic ou Applesoft, ou un fichier binaire. L'astérisque (*) précédant le type du fichier de HELLO indique qu'il a été verrouillé.

Clôture de fichiers

En réalité, le Programme 1 est lamentable pour une raison majeure : il ne clot pas le fichier terminé. La commande CLOSE (« clore ») est très importante. Son omission peut provoquer la perte d'informations, et peut-être la destruction de données sur une autre disquette (voyez la section consacrée aux défaillances logicielles, dans ce même chapitre).
La commande CLOSE dispose de deux formats ; le premier est :

```
CLOSE
```

Sans aucun paramètre, la commande CLOSE clot tous les fichiers ouverts, sur toutes les disquettes quels que soient les connecteurs, unités et volumes.
Parfois, vous souhaitez ne fermer qu'un fichier spécifique, sinon plusieurs. Dans ce cas, vous ajoutez son nom (ou leurs noms) à la commande CLOSE :

```
CLOSE NOMDUFICHIER
```

Cette commande ne requiert ni n'accepte aucun paramètre de connecteur, unité ou volume, dans aucune de ses formes. Le DOS sait où se trouve le fichier, puisqu'il a été ouvert.
On corrigera le Programme 1 en ajoutant cette ligne :

```
290 PRINT D$;"CLOSE"
```

Ou encore, vous pourriez ajouter :

```
290 PRINT D$;"CLOSE SEQUENTIEL"
```

Ecriture de fichiers séquentiels

Le Programme 1 n'a aucune utilité. Les disquettes servent à stocker et retrouver des informations. Puisqu'on ne peut retrouver que ce qui est déjà sur le disque, nous allons discuter de la façon d'enregistrer des informations au préalable.
Les informations sont envoyées à Disk II de la même façon qu'à l'écran ou à l'imprimante : via un ordre PRINT. Tout ce que vous pouvez afficher peut être stocké dans un fichier sur disquette. En fait, vous pourriez vous représenter un fichier séquentiel comme un écran TV, ou mieux, comme la feuille de papier de votre imprimante.
Lorsque vous enregistrez quelque chose dans un fichier, le DOS met à jour un pointeur interne qui pointe la position suivante de la disquette où les données seront rangées, de la même façon que l'imprimante fait avancer le papier à la ligne suivante.
Un pointeur de fichier séquentiel ne peut qu'avancer. La commande OPEN le ramène à la position de départ du fichier.

Avant que vous ne puissiez ranger des données dans un fichier, vous devrez utiliser une commande WRITE (« écriture ») pour indiquer au DOS que les instructions PRINT serviront à écrire dans le fichier, et non à afficher sur l'écran. Pour un fichier séquentiel, la commande WRITE se traduit par ;

```
WRITE NOMDUFICHIER
```

Après émission de l'ordre WRITE, les sorties seront dirigées vers le fichier correspondant. Notez qu'elles comprendront d'éventuels messages d'erreur. Cependant, après rangement d'un message d'erreur dans le fichier, la commande WRITE est annulée. Vous verrez réapparaître le caractère d'appel Basic et le curseur sur l'écran.
La commande WRITE doit être incluse dans un ordre PRINT et précédée par un caractère CTRL-D. Si vous donnez l'ordre WRITE en mode immédiat, vous obtiendrez le message d'erreur NOT DIRECT COMMAND.
Ajoutez les lignes suivantes à Programme 1 :

```
210 PRINT D$;"WRITE SEQUENTIEL"
220 PRINT "TEXTE A RANGER DANS LE FICHIER"
```

Le programme va exécuter la séquence :

1. Il crée le fichier s'il n'existe déjà.
2. Il ouvre le fichier.
3. Il range le texte dans le fichier.
4. Il clot le fichier.

Vous pouvez introduire autant de PRINT que vous le voulez entre les lignes 210 et 290. Vous pouvez faire enregistrer du texte, des nombres et des variables selon toutes combinaisons, tant que la syntaxe de PRINT reste correcte. Par exemple :

```
220  FOR I = 1 TO 100
230  PRINT I
240  NEXT I
250  PRINT "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

Remarquez qu'on n'utilise le caractère préfixe CTRL-D qu'une fois, avec la commande du DOS, et jamais avec les ordres d'écriture dans un fichier.
Faites attention à ne pas enregistrer des caractères dans un fichier lorsque les instructions FLASH ou INVERSE sont en action, car ces caractères ne seraient pas traités correctement par le DOS.
Rappelez-vous que, chaque fois que vous lancerez ce programme, tout ce qui suivra les ordres PRINT surchargera et effacera les anciennes données qui pré-existaient dans le fichier. Si vous enregistrez un nombre moindre de caractères qu'il n'y en avait dans les donnée précédentes, la fin de ces données subsistera, à la suite des nouveaux caractères. Une façon de résoudre ce problème des données restantes consiste à effacer le fichier avant de ranger de nouvelles données. Une commande DELETE (dans un ordre PRINT) pourra être incorporée au programme tout juste avant OPEN. Chaque fois que le programme sera lancé, le fichier sera effacé puis recréé par OPEN.
Cependant, un autre problème se manifestera si vous essayez de lancer le programme alors qu'il n'existe pas de fichier appelé SEQUENTIEL sur la disquette. Dans ce cas, vous recevrez le message FILE NOT FOUND (« fichier non trouvé »). Vous pourrez prévenir ce message en ajoutant une commande OPEN supplémentaire, juste avant la commande DELETE. Voici ce qui se produira :

1. Le premier OPEN crée un fichier s'il n'existe déjà.
2. DELETE efface le fichier, quel que soit le moment où il a été créé.
3. Le second OPEN crée un nouveau fichier, alors vide.

En modifiant ainsi Programme 1, rappelez-vous qu'une fois que vous avez spécifié le connecteur, l'unité et le volume, il n'est plus nécessaire de les spécifier à nouveau, à moins que vous ne vouliez les changer. Pensez-y, et utilisez une commande OPEN avec les paramètres de connecteur, unité et volume dans la première commande du DOS, ce qui permettra aux commandes suivantes de reprendre ces paramètres par défaut.
Si vous avez introduit ces modifications dans le Programm 1, son listage devrait être le suivant :

```
100 D$ = "": REM CTRL-D
110  PRINT D$; "OPEN SEQUENTIEL,V123,S6,D1"
120  PRINT D$; "DELETE SEQUENTIEL"
200  PRINT D$;"OPEN SEQUENTIEL"
210  PRINT D$;"WRITE SEQUENTIEL"
220  PRINT "TEXTE A RANGER DANS LE FICHIER"
290  PRINT D$;"CLOSE"
300  END
```

Chaque fois que vous lancerez ce programme, il rangera le contenu des ordres PRINT entre les lignes 210 et 290 dans le fichier SEQUENTIEL. Vous pouvez adopter un autre nom de fichier, mais n'oubliez pas de reprendre le même partout.
Vous pouvez employer une variable pour nom du fichier, et laisser au programme le soin de rechercher le nom pour accéder au fichier. Avec une telle modification, le programme donne :

```
10  INPUT "NOM DU FICHIER": ";F$
100 D$ = "": REM CTRL-D
110  PRINT D$;"OPEN ";F$;",V123,S6,D1"
120  PRINT D$;"DELETE ";F$
200  PRINT D$;"OPEN"; F$
210  PRINT D$;"WRITE ";F$
220  PRINT "TEXTE A RANGER DANS LE FICHIER"
290  PRINT D$;"CLOSE"
300  END
```

Vous pouvez aller encore plus loin et confier au programme le soin de demander le connecteur, l'unité et le volume avec :

```
10  INPUT "NOM DU FICHIER: ";F$
20  INPUT "NUMERO DU CONNECTEUR: ";S
30  INPUT "NUMERO DE L'UNITE A DISQUETTES: ";D
40  INPUT "NUMERO DU VOLUME: ";V
100 D$ = "": REM CTRL-D
110  PRINT D$;"OPEN ";F$;",S";S;",D";D",V";V
```

Assurez-vous que vous introduisez ces modifications exactement comme elles apparaissent ci-dessus. N'oubliez pas les virgules avant les paramètres S, D et V, faute de quoi le DOS ne pourra pas les distinguer du nom du fichier.
Afin que ce programme soit réellement utile, il devrait vous permettre d'entrer le texte à

stocker sans avoir à modifier des instructions PRINT. Voici une façon de faire :

```
150 INPUT "ENTREZ LE TEXTE A STOCKER: ";T$
 .
 .
 .
220 PRINT T$
```

Mais cela ne vous permettra d'entrer qu'une ligne de texte. On pourrait allonger ce texte en ajoutant d'autres PRINT, mais à nouveau, la longueur serait fixe. Pourquoi, alors, ne pas placer un test après INPUT, de façon que vous puissiez indiquer la fin de votre texte ? Posez un GOTO après le PRINT, ainsi :

```
150 INPUT "ENTREZ LE TEXTE A RANGER: ";T$
160 IF T$ = "FIN" THEN 290
210 PRINT D$;"WRITE ";F$
220 PRINT T$
230 GOTO 150
```

Désormais, lorsque vous aurez achevé d'entrer votre texte, vous pourrez frapper FIN et le fichier sera clos. Mais il subsiste un gros problème. Rappelez-vous que la commande WRITE oriente toutes les sorties vers le fichier. Puisque l'ordre INPUT sort ENTRER LE TEXTE A RANGER:, cette ligne n'apparaîtra pas sur l'écran après exécution de WRITE ; elle sera aussi rangée sur la disquette.
Aussi, WRITE doit être annulé avant une sortie non destinée à la disquette. N'importe quelle commande du DOS l'annulera, mais le plus sûr consiste à employer la *commande d'annulation,* qui est le caractère CTRL-D seul. Ajouter cette ligne au programme :

```
225  PRINT D$
```

Toutes ces modifications ayant été apportées, vous disposez d'un programme qui vous permettra de ranger n'importe quel volume de texte souhaité (ou tout du moins, ce que peut recevoir la disquette), sous le nom de fichier que vous aurez choisi et avec n'importe quelle unité à disquettes connectée à votre Apple II.

Lecture des fichiers séquentiels

De la même façon que les sorties peuvent être dirigées vers le disque, les entrées peuvent être acceptées d'un disque. La commande READ identifie un fichier en disque comme source d'entrée des données. Pour un fichier séquentiel, la commande READ est de ce type :

```
READ NOMDUFICHIER
```

Cette commande doit se trouver dans un ordre PRINT et précédée par un caractère CTRL-D. Si vous émettez un READ en mode immédiat, vous obtiendrez un message d'erreur NOT DIRECT COMMAND.
Après exécution d'un READ, les instructions INPUT recevront des données des fichiers spécifiés jusqu'à ce qu'une autre commande DOS, ou une erreur, annule READ. Le « Programme 2 » ci-dessous montre l'usage de READ :

```
100 D$ = "": REM CTRL-D
110  INPUT "NOM DU FICHIER A LIRE: ";F$
120  INPUT "CONNECTEUR NUMERO: ";S
130  INPUT "UNITE A DISQUETTES NUMERO: ";D
140  INPUT "VOLUME NUMERO: ";V
150  PRINT D$;"OPEN ";F$;",S";S;",D";D;",V";V
160  PRINT D$;"READ ";F$
170  INPUT A$
180  PRINT A$
190  GOTO 170
200  END
```

Le Programme 2 affiche tout le texte enregistré dans le fichier créé par Programme 1. Après affichage de ce texte, le message END OF DATA (« fin de données ») apparaîtra et le programme s'arrêtera avec le message BREAK IN 170 (« arrêt à 170 »).
Vous pouvez remarquer qu'il n'y a pas de CLOSE dans Programme 2. Nous avons expliqué pourquoi il faut clore un fichier lorsqu'on en a terminé avec lui ; mais dans le cas actuel, on n'écrit pas et ainsi, les informations dans le répertoire ou dans la liste pistes-secteurs n'ont pas besoin d'être mises à jour (en supposant que le nom du fichier spécifié existe bien). Un fichier devrait être clos simplement pour rester entier.
Ce n'est pas une bonne coutume que d'écrire des programmes qui affichent des messages d'erreur et stoppent dans des circonstances aussi prévisibles que la fin des données dans un fichier.

Pour prévenir l'erreur de END OF DATA

La condition de fin de données (END OF DATA) peut être détectée avant qu'une erreur n'intervienne. Le Programme 2 pourrait transmettre les commandes à un CLOSE après détection de la fin des données. La façon la plus simple de détecter la fin des données consiste à utiliser l'instruction Applesoft ONERR GOTO (« sur erreur, aller à »), discutée dans le chapitre 4 ; mais ONERR GOTO n'est pas disponible en Integer Basic. (Les codes d'erreurs du DOS sont expliquées dans l'appendice C.)
Une autre façon de déceler la fin des données dans des fichiers enregistrés avec Programme 1 consiste à modifier Programme 1 de façon que, lorsque vous frappez le mot FIN, un mot spécial ou un caractère soit inscrit dans le fichier juste avant sa clôture. Modifiez Programme 2 pour rechercher ce marqueur spécial de fin de texte et clôturez le fichier. Cette méthode s'applique à l'Integer Basic et à l'Applesoft.

Différences entre Integer Basic et Applesoft

La commande READ identifie un fichier sur disquette comme la source des données pour les ordres INPUT suivants. Mais la syntaxe d'INPUT doit rester conforme à celle du Basic dans lequel le programme est rédigé (voir le chapitre 8). Or, les données fournies par un fichier dépendent de la façon dont elles ont été stockées.
En Applesoft, par exemple, écrivez :

```
100 D$ = "": REM CTRL-D
200  PRINT D$;"OPEN FICHIER1"
300  PRINT D$;"WRITE FICHIER1"
400  PRINT "HELLO, C'EST UN TEST"
500  PRINT D$;"CLOSE"
600  END
```

Puis, avec une modification mineure dans ce même programme :

```
400 PRINT "HELLO, C'EST UN TEST"
```

Ces programmes rangent *deux* lignes de texte dans FICHIER 1. Applesoft ne traite pas la virgule comme un séparateur de la même façon qu'Integer Basic. Si vous tentez de lire FICHIER 1 avec :

```
300 PRINT D$;"READ FICHIER1"
400 INPUT A$
```

alors que vous êtes en Applesoft, vous obtiendrez le message ?EXTRA IGNORED et la variable A$ se verra attribuer HELLO, quelle que soit la ligne 400 utilisée. En Applesoft, les virgules séparent des valeurs multiples dans une instruction PRINT.
Integer Basic acceptera les deux lignes comme une seule, et ainsi, la valeur de A$ de la seconde version, ligne 400, sera HELLO, C'EST UN TEST.

Utilisation de GET en Applesoft pour lire des fichiers de texte

En Applesoft, l'ordre GET peut servir à lire des données dans un fichier sur disquette. GET diffère de INPUT par le fait qu'il ne retourne qu'un seul caractère à la fois. De ce fait, si un fichier contient le texte CE FICHIER CONTIENT DU TEXTE et si vous exécutez un OPEN, un READ et des GET, le premier GET ramènera la lettre C, le second E, le troisième un espace, le quatrième un F, etc., ce jusqu'au dernier caractère. Si GET ramène une virgule ou un retour chariot, le programme pourra le détecter et l'interpréter correctement.
Le Programme 3 ci-desous montre comment un fichier peut être lu avec GET, servant à bâtir une ligne de texte, caractère par caractère :

```
100 D$ = CHR$ (4): REM CTRL-D
200  INPUT "NOM DU FICHIER A LIRE: ";F$
300  INPUT "CONNECTEUR NUMERO: ";S
400  INPUT "UNITE A DISQUETTES NUMERO: ";D
500  INPUT "VOLUME NUMERO: ";V
600  PRINT D$;"OPEN ";F$;",V";V;",D";D;",S";S
700  PRINT D$;"READ ";F$
800 B$ = ""
900  GET A$
1000  IF A$ = CHR$ (13) THEN 1300
1100 B$ = B$ + A$
1200  GOTO 900
1300  REM RETOUR DU CARACTERE TROUVE
1400  REM B$ EST COMPLET
1500  PRINT B$
1600  GOTO 800
1700  END
```

L'ordre GET rencontre cependant un problème avec les fichiers sur disque. Le premier caractère après qu'un GET ait été exécuté est ignoré. Si le premier caractère est une commande du DOS, c'est le caractère CTRL-D qui sera ignoré, ce qui signifie que la commande entière sera affichée et non exécutée comme une commande du DOS.
La solution consiste à introduire un caractère supplémentaire sans autre utilité que d'être rejeté et qui sera ignoré. Un bon choix consiste à employer CTRL-A (code 1 en ASCII), car il n'est pas affichable et ne possède aucune signification particulière (ce qui n'est plus le cas de CTRL-D, par exemple).

On peut ajouter, ou modifier la ligne 1500 de Programme 3 pour aboutir à :

```
1500  PRINT  CHR$ (1);B$: REM  CHR$(1) = CTRL-A
```

Rangement de nombres en fichiers

Vous pouvez vous servir de Programme 1 pour ranger des nombres dans un fichier, soit en tant que partie d'un texte, comme dans :

```
220 PRINT "TELEPHONEZ AU 329-63-70"
```

soit directement en tant que valeurs numériques :

```
230 PRINT 1,2,3,4,5
```

ou encore via des variables numériques :

```
240  PRINT A,B,C,D,E
```

Si vous rangez des nombres directement, vous obtiendrez des résultats étranges lorsque vous les lirez (READ) avec Programqme 2.
Avec l'Integer Basic, les nombres séparés par des virgules ou des points-virgules formeront un seul nombre (par exemple, et pour la linge 230 ci-dessus : 12345), sur une ligne de sortie.
Avec l'Applesoft, les choses sont encore plus déroutantes. Les trois premiers nombres sont concaténés (chaînés) (et l'on obtient 123) et affichés en une ligne de sortie. Les deux valeurs suivantes (4 et 5) produisent une ligne de sortie chacune, soit un total de trois lignes de sortie.
Ces problèmes sont dûs au format employé pour stocker l'information sur disque. Les virgules ne sont pas rangées entre les nombres si elles ne seraient pas affichées sur l'écran. Sur l'écran, les virgules provoquent une tabulation. Lorsque la sortie est dirigée vers le disque, le DOS abandonne totalement les virgules. Rien d'équivalent à la tabulation ne se produit. En résultat, les valeurs reviennent concaténées jusqu'à ce qu'un retour chariot (code 13, en ASCII) les sépare. Le retour chariot est le seul caractère interprété par le DOS comme un séparateur de valeurs. L'Integer Basic sort un retour chariot après le cinquième stop de tabulation (virgule) alors qu'Applesoft le fait après le troisième.
Pour éviter ces problèmes, assurez-vous que chaque champ numérique que vous rangez dans un fichier sur disquette est suivi d'un retour chariot. La façon la plus simple de procéder consiste à sortir chaque nombre avec un PRINT distinct :

```
230 PRINT 1: PRINT 2: PRINT 3: PRINT 4:
    PRINT 5:
```

Avec Applesoft, une autre méthode consiste à inclure un retour chariot comme séparateur dans une instruction PRINT :

```
230  PRINT 1;CHR$(13);2;CHR$(13);3;CHR$(1
 3);4;CHR$(13);5
```

Vous pourrez préférer définir une variable pour ce retour chariot, puis utiliser cette variable :

```
11  R$ = CHR$(13): REM  RETURN CHARACTER
    .
    .
    .
230  PRINT 1;R$;2;R$;3;R$;4;R$;5
```

Le programme a ainsi une allure plus nette.

COMMENT FAIRE DES AJOUTS À UN FICHIER SÉQUENTIEL (APPEND)

Un fichier séquentiel devrait être clos après avoir été écrit. Lorsque vous commandez CLOSE, vous perdez la piste de l'endroit où la dernière information a été stockée. Afin d'ajouter des informations à la fin du fichier, à la suite, vous devrez d'abord *retrouver* cette fin. Vous pourriez lire toutes les informations du fichier jusqu'à la dernière, mais cela pourrait prendre beaucoup de temps si le fichier est important. La commande APPEND fera le travail pour vous.
APPEND place le pointeur du fichier sur le premier caractère inutilisé suivant sa fin. Si vous lisez le fichier après une commande APPEND, vous obtiendrez un message d'erreur END OF DATA. Si vous écrivez après un APPEND, les nouvelles données se placeront à la suite des anciennes.
La commande APPEND est employée à la place de OPEN. Deux différences importantes les distinguent :

1. APPEND exige que le fichier existe déjà. Sinon, le message d'erreur FILE NOT FOUND (« fichier non trouvé ») est retourné. APPEND ne créera pas de fichier ; il suppose que le fichier pré-existe.
2. APPEND place le pointeur à la *fin* du fichier. OPEN le place au *commencement.*

Le format de la commande APPEND est le même que pour OPEN. En voici un exemple :

```
APPEND NOMDUFICHIER,S6,D2,V99
```

Comme toujours, les paramètres du connecteur, de l'unité à disquettes et du volume sont optionnels.

LA COMMANDE POSITION

Une autre commande utile est POSITION. Elle déplace le pointeur *en avant* (jamais en arrière) d'un nombre de champs spécifié et relatif à la position courante du pointeur. POSITION a l'allure suivante :

```
POSITION  NOMDUFICHIER,R30
```

R indique un champ relatif, le nombre qui suit R est le nombre de champs à sauter. Les champs sont marqués par des caractères de retour chariot ; ici, la commande POSITION va compter 30 « retours chariot » après la position courante du pointeur et placera le pointeur à ce nouvel endroit. Si vous spécifiez R0, le pointeur ne bougera pas.
POSITION examine, en fait, le fichier caractère par caractère, à partir de la position courante. S'il n'y a pas assez de champs, ou si un octet inutilisé est rencontré, le message d'erreur END OF DATA (« fin de données ») est affiché. L'exécution d'un INPUT ou d'un GET n'est pas nécessaire pour causer cette erreur.
Un fichier doit être ouvert avant de pouvoir être référencé par la commande POSITION. Lorsque vous ouvrez un fichier, le pointeur est placé à son début. Si vous émettez une commande POSITION à ce moment, elle sélectionnera effectivement le champ *absolu* du fichier.
Rappelez-vous que, tout comme les autres commandes du DOS, POSITION annule les commandes READ et WRITE. Assurez-vous de l'exécution de POSITION avant d'émettre les ordres READ ou WRITE, et non après.

UTILISATION DES FICHIERS À ACCÈS ALÉATOIRE

Les fichiers à accès aléatoire sont structurés en sections appelés *enregistrements.* Chaque enregistrement d'un fichier contient le même volume d'informations, défini en octets (caractères) lorsque le fichier est créé.
Le total des informations pouvant être rangées dans un enregistrement est appelé la *longueur* de l'enregistrement.
Les enregistrements sont identifiés par un nombre indiquant leur position absolue dans le fichier. Le premier enregistrement d'un fichier porte le numéro zéro, le suivant est le 1, suivi par le 2, etc.
Le plus petit fichier à accès aléatoire est d'un enregistrement. Le fichier s'étend au fur et à mesure qu'on ajoute de nouveaux enregistrements. Pour éliminer des enregistrements indésirables d'un fichier à accès aléatoire, et réduire la taille du fichier, on devra copier les enregistrements à préserver dans un nouveau fichier à accès aléatoire.
Les programmes devront spécifier quels enregistrements d'un fichier à accès aléatoire ont été sélectionnés, et quelle partie de l'enregistrement est à atteindre.

Ouverture d'un fichier à accès aléatoire

Pour définir un fichier à accès aléatoire, vous devez inclure un paramètre additionnel lors de son ouverture : le paramètre *longueur.* Ce paramètre (l) spécifie la longueur de chaque enregistrement :

```
OPEN NOMDUFICHIER,L10,S6,D1,V100
```

Le paramètre longueur a une valeur comprise entre 1 et 32767. Il n'est pas obligatoirement le premier paramètre de la liste mais il doit être présent si le fichier est à accès aléatoire.
Les programmes ne devront jamais écrire des enregistrements d'une longueur supérieure au nombre d'octets spécifié par le paramètre de longueur, y inclus les retours chariot et les virgules. S'il y a trop de caractères, pour un enregistrement, l'enregistrement suivant pourra se voir surchargé ou combiné, ce qui crée une véritable confusion.

Clôture d'un fichier à accès aléatoire

La commande CLOSE est identique pour les fichiers séquentiels et aléatoires.

Lectures et écritures en accès aléatoire

Les commandes READ et WRITE demandent un paramètre *enregistrement* lorsqu'elles visent un fichier à accès aléatoire. Le paramètre enregistrement place un pointeur sur le début de l'enregistrement. L'exemple suivant y fait appel (le symbole R vient de « record » = enregistrement) :

```
READ NOMDUFICHIER,R13
```

ou :

```
WRITE NOMDUFICHIER,R6
```

Le paramètre enregistrement peut ne pas être le seul de la liste. Vous pouvez aussi spécifier le connecteur, l'unité, et le volume. Tous ces paramètres peuvent intervenir dans n'importe quel ordre. Si le paramètre enregistrement est absent, le pointeur ne bouge pas.

EXEMPLE PRATIQUE D'ACCÈS ALÉATOIRE

Les programmes suivants font la démonstration de l'usage de fichiers en accès aléatoire. Ces programmes tourneront aussi bien en Applesoft qu'en Integer Basic. Pour utiliser le second programme en Integer Basic, vous devrez ajouter une instruction DIM avant la ligne 100 pour les variables B$ et C$ (255 caractères chacun). Le premier programme crée un fichier appelé ALEATOIRE :

```
 10 REM CREATION DU FICHIER ALEATOIRE
 20 REM
 30 REM FAITES TOURNER CE PROGRAMME D'ABORD
 40 REM CAR IL RANGE DES INFORMATIONS
 45 REM DANS L'ENREGISTREMENT ZERO
 50 REM ELLES DEVRONT EXISTER POUR QUE LE
 55 REM SECOND PROGRAMME FONCTIONNE
 60 REM
100 D$="" : REM CTRL-D
200 PRINT D$;"OPEN ALEATOIRE,S6,D1,L256" : REM OUVERTURE
    FICHIER
300 PRINT D$;"WRITE ALEATOIRE,R0" : REM ECRITURE, ENREGISTR.
    ZERO
400 PRINT 0 : REM PLACE UN ZERO DANS ENREGISTR. ZERO
500 PRINT D$;"CLOSE" : REM CLOTURE DU FICHIER
600 END
```

Dès lors que le fichier a été créé, le second programme servira à le lire, le modifier, lui ajouter ou lister les enregistrements, déjà présents. Chaque enregistrement contient une ligne d'informations que vous frappez au clavier :

```
10 REM PROGRAMME DE DEMONSTRATION DU FICHIER ALEATOIRE
20 REM
30 REM CE PROGRAMME ENTRETIENT UN FICHIER ALEATOIRE
40 REM CONSISTANT EN ENREGISTREMENTS D'UNE SEULE LIGNE
50 REM
60 REM L'ENREGISTREMENT ZERO CONTIENT UN NOMBRE
70 REM INDIQUANT LE DERNIER ENREGISTREMENT EN SERVICE
80 REM
85 REM NOTE: LE FICHIER "ALEATOIRE" DOIT ETRE CREE
90 REM AVANT USAGE DE CE PROGRAMME
95 REM
```



```
100 D$="" : REM CTRL-D
200 PRINT D$;"OPEN ALEATOIRE,S6,D1,L256"
225 PRINT D$;"READ ALEATOIRE, R0" : REM LECTURE ENREGISTR
    EMENT ZERO
250 INPUT M : REM M = LE DERNIER ENREGISTREMENT EN USAGE
275 PRINT D$ : REM ANNULATION ORDRE LECTURE
300 CALL -936 : REM EFFACER L'ECRAN
400 PRINT "DEMONSTRATION DU FICHIER ALEATOIRE"
500 PRINT : PRINT : PRINT : PRINT
600 PRINT "COMMANDES:" : PRINT
700 PRINT " 0 = STOP"
800 PRINT " 1 = LIRE UN ENREGISTREMENT"
900 PRINT " 2 = AJOUTER UN ENREGISTREMENT"
1000 PRINT " 3 = MODIFIER UN ENREGISTREMENT"
1100 PRINT " 4 = LISTER LES ENREGISTREMENTS"
1200 PRINT : PRINT
1300 PRINT "LEQUEL?"
1400 INPUT C
1500 IF C=0 THEN 3800 : REM BRANCHER
1600 IF C=1 THEN 2200 : REM A LA
1700 IF C=2 THEN 3300 : REM SECTION
1800 IF C=3 THEN 4600 : REM SELECTIONNEE
1900 IF C=4 THEN 6700 : REM DU PROGRAMME
2000 GOTO 300         : REM (OU RE-AFFICHER LE MENU)
2050 REM
2100 REM * * * * * LECTURE D'UN ENREGISTREMENT * * * * *
2150 REM
2200 CALL -936 : REM EFFACER L'ECRAN
2300 PRINT : PRINT "LECTURE D'UN ENREGISTREMENT" : PRINT
2400 PRINT "QUEL NUMERO D'ENREGISTREMENT (0 A STOP)";
2500 INPUT R
2600 IF R<1 THEN 300 : REM REVENIR AU MENU
2650 IF R>M THEN 2200 : REM L'ENREGISTREMENT N'EXISTE PAS
2700 PRINT D$;"READ ALEATOIRE,R";R;REM PREPARATION A LA
    LECTURE
2800 INPUT B$ : REM LECTURE DES DONNEES
2900 PRINT D$ : REM ANNULATION LECTURE
3000 PRINT : PRINT B$ : PRINT : REM AFFICHAGE DONNEES
3100 GOTO 2400 : REM DEMANDE D'UN AUTRE ENREGISTREMENT
3150 REM
3200 REM * * * * * ADJONCTION D'UN ENREGISTREMENT * * * * *
3250 REM
3300 CALL -936 : REM EFFACER L'ECRAN
3400 PRINT : PRINT "ADJONCTION D'UN ENREGISTREMENT" : PRINT
3500 PRINT "NUMERO SUIVANT D'ENREGISTREMENT = ";M+1
3600 PRINT : PRINT "ENTREZ LES DONNEES ";M+1
3625 PRINT "(PRESSEZ [RETURN] POUR FIN DE DONNEES)"
3700 INPUT B$ : REM REPONSE DE L'UTILISATEUR
3750 IF B$ = "" THEN 300 : REM TOUT VA BIEN [RETURN]
3800 PRINT D$;"WRITE ALEATOIRE,R";M+1 : REM PREPARATION EC
    RITURE
3900 PRINT B$ : REM ENVOI DONNEES A FICHIER
4000 M = M+1: REM INCREMENTER DERNIER NO ENREGISTREMENT
4100 PRINT "WRITE ALEATOIRE,R0" : REM PREPARATION A ECRITU
    RE SUR ENREGISTREMENT ZERO
4200 PRINT M : REM RANGEMENT NOUVELLE VALEUR
```

```
4300 PRINT D$ : REM ANNULATION COMMANDE ECRITURE
4400 GOTO 3500 : REM BOUCLE POUR AUTRE ENREGISTREMENT
4450 REM
4500 REM * * * * * MODIFICATION D'UN ENREGISTREMENT * * * * *
4550 REM
4600 CALL -936 : REM EFFACER L'ECRAN
4700 PRINT : PRINT "MODIFICATION D'ENREGISTREMENT" : PRINT
4800 PRINT "LEQUEL (0 A STOP)?"
4900 INPUT R
5000 IF R<1 THEN 300 : RETOUR AU MENU
5050 IF R>M THEN 4600 : REM RECOMMENCEZ SI ENREGISTREMENT
     NON EN FICHIER
5100 PRINT D$;"READ ALEATOIRE,R";R : REM PREPARATION LECT
     URE
5200 INPUT B$ : REM LECTURE ENREGISTREMENT
5300 PRINT D$ : REM ANNULATION ORDRE LECTURE
5400 PRINT : PRINT B$ : PRINT : REM AFFICHAGE DONNEES
5500 PRINT "ENTREZ LES NOUVELLES DONNEES"
5600 PRINT "PRESSEZ [RETURN] POUR ANNULER MODIFICATIONS"
5700 PRINT
5800 INPUT C$ : REM APPEL REPONSE UTILISATEUR
5900 IF C$>"" THEN 6200 : REM BRANCHER SI NOUVELLES DONNE
     ES
6000 PRINT "ENREGISTREMENT ";R;"INCHANGE ! ! !" : REM BOU
     CLER SI
6100 GOTO 4800 : REM PAS DE MODIFICATIONS DESIREES
6200 PRINT D$;"WRITE ALEATOIRE,R";R : REM PREPARATION ECR
     ITURE
6300 PRINT C$ : REM RANGER LES DONNEES MODIFIEES
6400 PRINT D$ : REM ANNULATION ORDRE ECRITURE
6500 GOTO 4800 : REM BOUCLE POUR AUTRE MODIFICATION
6550 REM
6600 REM * * * * * LISTAGE DES ENREGISTREMENTS * * * * *
6650 REM
6700 CALL -936 : REM EFFACER L'ECRAN
6800 PRINT : PRINT "LISTAGE DES ENREGISTREMENTS" : PRINT
6900 R = 0 : REM REMISE A ZERO DU COMPTEUR
7000 R = R + 1 : REM INCREMENTATION COMPTEUR
7100 IF R > M THEN 7700 : REM STOP APRES DERNIER ENREGIST
     REMENT
7200 PRINT D$;"READ ALEATOIRE R";R : REM PREPARATION LECT
     URE
7300 INPUT B$ : REM LECTURE DES DONNEES
7400 PRINT "NUMERO D'ENREGISTREMENT ";R : REM AFFICHAGE
     NUMERO D'ENREGISTREMENT
7500 PRINT B$ : PRINT : REM AFFICHAGE DONNEES ENREGISTREM
     ENT
7600 GOTO 7000 : REM BOUCLE POUR ENREGISTREMENT SUIVANT
7700 PRINT D$ : REM ANNULATION ORDRE LECTURE
7800 PRINT : PRINT "* * * * * FIN DE FICHIER" : REM PRINT
     FIN DE MESSAGE
7900 PRINT "PRESSEZ RETURN POUR CONTINUER";:
     REM DEMANDE DE REPONSE
8000 INPUT B$ : REM APPEL REPONSE
8100 GOTO 300 : REM RETOUR AU MENU
8150 REM
```



```
8200 REM * * * * * CLOTURE DU PROGRAMME * * * * *
8250 REM
8300 PRINT D$;"CLOSE" : REM POUR CLORE LE FICHIER
8400 CALL -936 : REM EFFACER L'ECRAN
8500 FOR I=1 TO 24 : PRINT : NEXT I : REM ALLER
     A LIGNE DU BAS
8600 PRINT "PROGRAMME TERMINE."
8700 END
```

LE PARAMÈTRE OCTET

Pour les les fichiers à accès aléatoire, un autre paramètre utile est *octet* (« byte », en anglais). Ce paramètre est employé avec READ, WRITE et POSITION, et sert à placer le pointeur sur l'octet (le caractère) spécifié dans un enregistrement donné. Une virgule et la lettre B sont alors ajoutés aux commandes READ, WRITE ou POSITION. Le paramètre de l'enregistrement doit être présent. En voici un exemple, le symbole de ce paramètre étant la lettre B (de « byte », soit « octet ») :

```
READ NOMDUFICHIER,R19,B3
```

Dans cet exemple, la lecture commencera avec le quatrième octet du vingtième enregistrement. (Rappelez-vous que le premier octet porte le numéro zéro.) Le paramètre octet peut déplacer le pointeur en avant et en arrière dans un enregistrement.

Si vous envisagez l'emploi du paramètre octet, il faudra que les enregistrements disposent d'un format rigoureux pour les données. Vous devrez connaître la position de chaque octet dans chaque champ d'un enregistrement, faute de quoi vous terminerez probablement avec des données sans signification.

En utilisant le paramètre octet avec la commande POSITION, rappelez-vous que POSITION annule toute commande antérieure READ ou WRITE. Vous pourriez exécuter un ordre READ ou WRITE incluant un paramètre enregistrement, puis employer POSITION pour aller vers le champ souhaité, mais il vous faudra un nouveau READ ou WRITE alors, car POSITION aura annulé les précédents.

POSITION provoque un déplacement du pointeur vers l'avant, vers un autre champ de l'enregistrement courant. Dans ce champ, le paramètre octet sert à référencer des sous-champs, caractère par caractère.

AUTRES COMMANDES DU DOS

Quelques autres commandes du DOS n'ont pas encore été expliquées. Il s'agit de EXEC, MAXFILES, TRACE, MON et NOMON.

EXEC

EXEC est une commande très spéciale. Elle sert à transférer le contrôle de votre Apple II à un fichier en mode texte. Elle se présente ainsi :

```
EXEC NOMDUFICHIER,R6,S5,D2,V23
```

Le paramètre R est semblable au paramètre d'enregistrement POSITION. Il se réfère au numéro d'un champ du fichier EXEC par lequel le traitement devrait débuter. Parce que EXEC ouvre un fichier et place le pointeur au début du fichier EXEC, le paramètre R se réfère à un numéro absolu de champ dans le fichier ; R0 est la position par défaut et commence l'exécution au début du fichier. R1 démarre l'exécution au second champ.

Les spécifications correspondantes à l'enregistrement, connecteur, unité et volume restent optionnelles et peuvent être données dans n'importe quel ordre.
Lorsque EXEC est émis, le fichier spécifié est ouvert, puis un READ implicite et une instruction INPUT surviennent. La première ligne du fichier est reprise si aucun paramètre R n'est présent. Si R est présent, c'est la ligne spécifiée par R qui est reprise.
La ligne reprise est traitée comme si elle avait été frappée sur le clavier en mode immédiat. Si la ligne ne possède aucune signification, vous obtiendrez le message ?SYNTAX ERROR ou ***SYNTAX ERR. Si une ligne de programme valide telle que :

```
100 PRINT "C'EST UN VRAI TEST"
```

est reprise, elle sera rangée en mémqoire comme si vous veniez de la frapper au clavier. S'il s'agit d'une commande directe telle que LIST ou RUN, elle sera aussitôt exécutée. Après que l'action déclenchée par cette première ligne ait été exécutée, la ligne suivante est lue et déclenchera à son tour une autre action. Ceci, jusqu'à ce que la fin du fichier soit atteinte ; à ce moment, le contrôle de Apple II retourne au clavier.
Supposez qu'un fichier de texte appelé BABA contienne ces quelques lignes :

```
PRINT "J'AI PRIS LES COMMANDES DE VOTRE APPLE ! ! !"
FP
100 GR
200  FOR I=0 TO 39
300  FOR J=0 TO 39
400  COLOR=RND(0)*15
500  PLOT I,J
600  NEXT J
700  NEXT I
800  FOR I=1 TO 5000 : NEXT I
900  TEXT : CALL -936
999  END
LIST
FOR I=1 TO 5000 : NEXT I
RUN
NEW
PRINT "TERMINE, VOICI LE CATALOGUE DE VOTRE DISQUE:"
CATALOG
```

Lorsque vous frapperez la commande :

```
EXEC BABA
```

plusieurs choses se produiront. D'abord, un message sera affiché. Puis, Applesoft sera appelé. Après quoi, un court programme sera introduit en mémoire et listé. Après une brève pause, le programme sera lancé, puis effacé de la mémoire. Un autre message sera affiché, suivi par le catalogue du disque courant.

Notes sur EXEC

EXEC dispose de plusieurs caractéristiques que vous devez connaître.
Un seul fichier à la fois peut être ouvert par EXEC. Si un fichier adressé par EXEC contient une autre commande EXEC, le premier fichier est fermé et le second prend le contrôle.

Après qu'EXEC ait lancé un programme, le champ suivant du fichier (la ligne suivante) est exécuté. Si le programme est avorté avec un CTRL-C, EXEC ne se poursuivra généralement pas.
Si un programme lancé par EXEC contient un ordre INPUT, l'entrée est prise dans le champ suivant du fichier EXEC. Il y aura un problème si ce champ suivant est une commande du DOS en mode immédiat (et non une ligne du programme). La commande sera exécutée au lieu d'être acceptée comme une entrée de données.
Si un programme exécute une commande CLOSE, ou si le fichier EXEC contient un CLOSE en mode immédiat, le fichier EXEC ne sera pas clos.

Utilisation d'EXEC pour convertir un programme d'un Basic à l'autre

EXEC peut servir à convertir un programme d'un Basic à l'autre, Applesoft et Integer Basic. Commencez par ranger le programme dans un fichier en mode texte. Vous pouvez le faire en ajoutant quelques lignes au programme créant un fichier en mode texte, puis en exécutant une commande WRITE ; listez le programme. Le listage ira dans le fichier en mode texte. Voici les instructions nécessaires :

```
1 D$ = "": PRINT D$;"OPEN FICHIER": PRINT D$;"WRITE FICHIER"
2 LIST 10,32767
3 PRINT D$;"CLOSE": END
10 REM VOTRE PROGRAMME COMMENCE ICI
```

Lorsque vous lancerez ce programme, seules les lignes 1, 2 et 3 seront exécutées. Elles vont ouvrir un fichier et ranger dedans le listage du programme. Lorsque ce sera terminé, vous pourrez introduire les modifications pour passer dans l'autre Basic (avec FP, ou INT, par exemple). Frappez ensuite :

```
EXEC FICHIER
```

Les lignes de l'ancien programme seront chargées du disque dans la mémoire. Vous devrez alors éditer le programme pour corriger tout ce qui n'est pas conforme aux règles du nouveau langage.

MAXFILES

MAXFILES vous sert à spécifier le nombre maximal de fichiers qui peuvent être ouverts à tout instant. Chaque fichier dispose de 595 octets réservés en mémoire servant de buffer. Chaque buffer comporte deux sections de 256 octets, l'un pour la lecture et l'autre pour l'écriture. Les 83 octets restants servent aux informations d'intendance, par exemple à la liste pistes-secteurs.
Lorsqu'un fichier est ouvert, ou lorsqu'une opération avec le disque telle que CATALOG ou LOCK survient, le DOS lit 256 octets d'informations sur le disque et les place dans le buffer. Si les informations sont modifiées et doivent être ré-écrites sur le disque, les 256 octets sont copiés du buffer et reportés sur la disquette.
L'exemple suivant spécifie huit fichiers au maximum :

```
MAXFILES 8
```

Le nombre de fichiers doit être un entier entre 1 et 16. Lorsque le DOS est appelé, il prévoit trois buffers. MAXFILES peut être supérieur si vous avez l'intention d'employer

plus de trois fichiers simultanément, ou inférieur si vous voulez disposer des octets mémoires pour votre application.
L'un des buffers sert uniquement à exécuter l'une des commandes suivantes du DOS :

APPEND	FP	POSITION
BLOAD	INIT	READ
BRUN	INT	RENAME
BSAVE	LOAD	RUN
CATALOG	LOCK	SAVE
CHAIN	MAXFILES	UNLOCK
CLOSE	MON	VERIFY
DELETE	NOMON	WRITE
EXEC	OPEN	

Ainsi, si vous avez ouvert des fichiers sur disque jusqu'aux limites et si vous avez ensuite émis la commande CATALOG, le message d'erreur NO BUFFERS AVAILABLE (« pas de buffers disponibles ») est retourné. Aucun buffer n'est requis pour des commandes employées à l'extérieur du contexte du DOS (par exemple, la commande cassette LOAD). Lorsque MAXFILES est modifié, les programmes en Integer Basic sont effacés et les chaînes en Applesoft sont bouleversées. De ce fait, exécutez MAXFILES avant le chargement ou le lancement d'un programme.
MAXFILES peut être exécuté dans un programme en Applesoft s'il est précédé par un caractère CTRL-D dans une instruction PRINT. MAXFILES empêchera des instructions telles que GOTO, GOSUB et autres de fonctionner correctement, sauf si MAXFILES est la première instruction du programme ; on emploiera alors MAXFILES ainsi :

```
]1 REM PREMIERE COMMANDE MAXFILES
]2 PRINT CHR$(4);"MAXFILES 9"
]3 REM LE PROGRAMME NORMAL DEBUTE ICI
]4 D$=CHR$(4):REM CTRL-D
```

Vous ne pouvez vous servir de MAXFILES, en Integer Basic, que dans le contexte d'un fichier EXEC. Par exemple, le fichier EXEC contient trois instructions en mode pseudo-immédiat :

```
MAXFILES 8
LOAD PROGRAMME
RUN 10
```

Il sera appelé dans un programme comme suit :

```
5 PRINT D$;"EXEC MXF":REM ETABLISSEMENT
  MAXFILES
10 REM LE PROGRAMME DEBUTE ICI
```

UTILISATION DES AIDES AU DÉBUGAGE DU DOS

Parce que les programmes accédant aux disques sont généralement quelque peu complexes, le DOS dispose de trois commandes, MON, NOMON et TRACE qui vous serviront à les débuguer (mettre au point).

MON

MON est l'abrégé de moniteur. MON vous permet de visualiser les informations allant et venant de la disquette. MON emploie trois paramètres ; on les trouve dans l'exemple suivant :

```
MON C,I,O
```

Les paramètres de MON spécifient le type d'information à afficher. C fait afficher les commandes à la disquette. I (pour « input », entrée) fait afficher les informations lues sur la disquette ; O (pour « output », sortie) fait afficher les informations écrites sur la disquette.
Les paramètres peuvent apparaître dans n'importe quel ordre et combinaison. L'un d'eux, au moins, doit être présent, sinon la commande est ignorée.
MON reste actif jusqu'à ce qu'un NOMON, un INT ou FP soit exécuté, ou que le DOS soit ré-appelé.

NOMON

NOMON annule l'effet de MON. NOMON accepte les mêmes trois paramètres que MON, mais NOMON spécifie les données qu'il ne faut plus afficher. Par exemple, en supposant que MON I, O, C ait été émis, la commande :

```
NOMON O
```

annulera l'affichage des écritures (sorties) sur disque. Les lectures (entrées) et les commandes du DOS continueront à être affichées.

Utilisation de la commande TRACE

La commande TRACE (voir chapitre 4) est employée pour débuguer les programmes. Parce que TRACE affiche les numéros de lignes, sans retour chariot, les commandes du DOS sont concaténées au numéro de ligne et ignorées. Le remède à ce problème est simple. Un retour chariot devra être introduit avant chaque commande du DOS. Pour cela, modifiez la ligne :

```
100 D$ =  CHR$ (4): REM  CTRL-D
```

pour lire :

```
100 D$ =  CHR$ (13) +  CHR$(4): REM  RETURN + CTRL
```

Désormais, lorsque PRINT D$ surviendra, le caractère CTRL-D sera précédé par un retour chariot, le séparant des numéros de lignes tracés.
Un autre problème subsiste cependant. Si une commande WRITE est active, tous les numéros de lignes seront stockés dans le fichier. (Nous le regrettons, mais cela, on ne peut l'éviter.)

FICHIERS EN LANGAGE MACHINE IMAGES BINAIRES SUR DISQUE

Le Disk II supporte des fichiers en langage machine et des images binaires (graphiques) sur disquettes. Ces fichiers apparaissant précédés de la lettre B sur le catalogue du disque ; B est le code du type de fichier.
Des images tout à la fois à haute et basse résolution peuvent être rangées sur disquette et appelées, puis affichées.
Les programmes en langage machine peuvent être chargés et exécutés directement, ou encore peuvent être appelés par les programmes en Basic avec une instruction CALL ou une fonction USR.
Le DOS dispose de trois commandes spécifiquement destinées aux fichiers binaires. Il s'agit de BSAVE, BLOAD et BRUN. Leur effet est le même que celui de leur équivalent non binaire (BSAVE = SAVE, BLOAD = LOAD, et BRUN = RUN).

BSAVE

Comme son nom l'indique, BSAVE sauvegarde une image binaire sur le disque. En voici un exemple :

```
BSAVE NOMDUFICHIER,A378,L21,S6,D2,V6
```

Remarquez les deux paramètres qui n'existent pas avec les autres commandes du DOS. Notez aussi que ces paramètres *ne sont pas* optionnels ; ils doivent être spécifiés. Les paramètres de connecteur, unité et volume sont, comme de coutume, optionnels.
Le paramètre A est le paramètre *adresse ;* il fait référence à l'adresse de départ en mémoire où l'image binaire va être sauvegardée. L'adresse peut être une constante décimale ou hexadécimale. Les valeurs hexadécimales sont précédées par le symbole dollar ($). Les valeurs décimales vont de 0 à 65535. Les valeurs négatives sont interdites.
Le paramètre L spécifie la *longueur* de l'image binaire à sauvegarder. Cette longueur est le nombre d'octets de l'image. Elle peut être spécifiée en décimal ou en hexadécimal, les valeurs hexadécimales étant précédées par le symbole dollar ($). La longueur doit se situer dans la gamme de 1 à 32767 faute de quoi une SYNTAX ERROR sera générée. La valeur de 32767 est la plus grande que le DOS puisse ranger dans un unique champ. Deux ordres BSAVE doivent être employés pour ranger plus de 32767 octets.
Si les positions mémoires dans la gamme spécifiée n'existent pas physiquement, aucune erreur ne sera retournée. Il n'est pas très utile de spécifier une position externe à la gamme installée dans votre Apple II (par exemple, 49151 ou $BFFF pour un système 48K).

BLOAD

BLOAD retrouve le contenu de fichiers binaires et le charge en mémoire. La commande BLOAD a l'allure suivante :

```
BLOAD NOMDUFICHIER,A378,L21,S6,D2,V6
```

Avec la commande BLOAD, le paramètre adresse est facultatif. S'il est absent, l'image sera chargée en commençant par l'adresse spécifiée lorsque l'image a été sauvegardée.
Les programmes en langage machine pourront ne pas fonctionner correctement s'ils sont chargés à de mauvaises adresses en mémoire.
A la différence de la commande LOAD, BLOAD n'effacera pas les programmes ou les valeurs des données, sauf s'ils résident en mémoire aux positions où l'image sera chargée. Seules, les positions dans la gamme BLOAD seront affectées ; aucune autre valeur en mémoire n'est modifiée.
Aucune erreur ne sera retournée si vous spécifiez des positions en mémoire morte (ROM) comme partie de la gamme BLOAD. Les positions en ROM ne seront pas modifiées, bien naturellement.

BRUN

BRUN est identique à BLOAD, à l'exception qu'après qu'un fichier ait été chargé, BRUN exécute un saut (JMP, pour « jump ») en langage machine à l'instruction se trouvant à l'adresse de départ. Si aucune adresse n'a été spécifiée, le saut se fera à l'adresse à partir de laquelle l'image a été sauvegardée. En voici un exemple :

```
BRUN NOMDUFICHIER,A378,L21,S6,D2,V6
```

N'utilisez jamais BRUN avec une image graphique. Le résultat serait imprévisible.

CHAPITRE 6

GRAPHIQUE ET SONS

L'Apple II est doté de la faculté de générer des graphiques en couleur, en vidéo, et des sons. Ces deux possibilités confèrent une nouvelle dimension aux programmes que vous utilisez comme à ceux que vous pourriez rédiger. Ce chapitre est destiné au débutant qui n'a peut-être appris que le Basic (peut-être en lisant ce livre), et au programmeur en langage assembleur. Maîtriser le graphique n'est pas difficile, surtout en langage évolué. Le moniteur de l'Apple II, avec ses sous-programmes incorporés en langage machine, aidera le programmeur en langage assembleur à employer le graphique et le haut-parleur inclus dans l'Apple II. Après avoir terminé ce chapitre, vous disposerez de suffisamment de connaissances sur ces deux thèmes pour pouvoir commencer à vous servir du graphique et des sons dans les programmes que vous écrirez.

GRAPHIQUE À BASSE RÉSOLUTION

Apple II dispose de deux zones distinctes en mémoire pour le graphique à faible résolution. Ces deux zones sont appelées des *pages*. Chaque page à faible résolution peut apparaître sur l'écran sous forme d'un graphisme sur 40 colonnes et 48 rangées, comme le montre la figure 6.1.

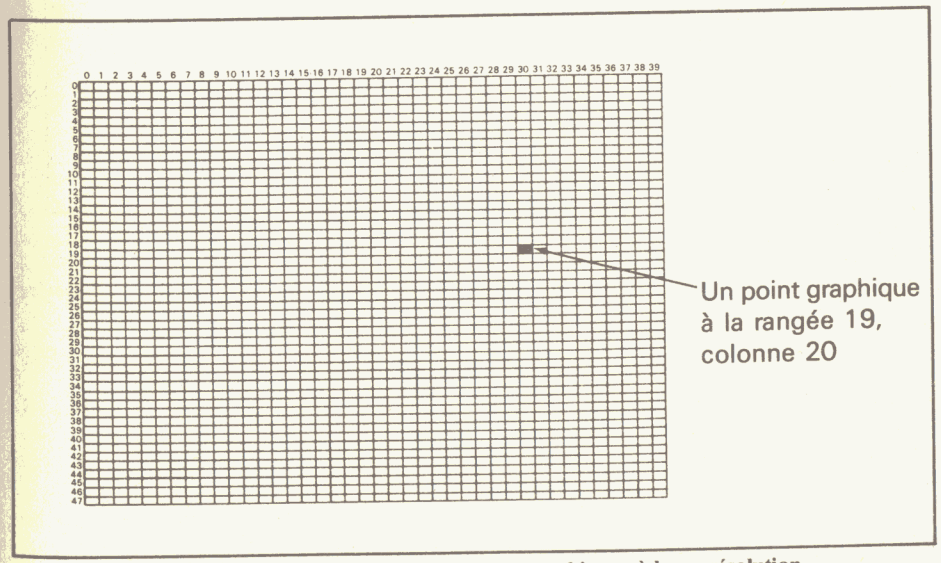


Fig. 6.1. — Organisation de l'écran pour graphismes à basse résolution.

Chaque coordonnée (intersection d'une rangée et d'une colonne) se manifeste comme un petit rectangle sur l'écran. Un page contient 1 920 coordonnées (40 colonnes par 48 rangées), et vous pouvez attribuer l'une quelconque des 16 couleurs à chaque coordonnée dans une page. Le tableau 6.1 montre les couleurs disponibles. Il n'est pas nécessaire que vous connaissiez le mode de travail interne de Apple II pour savoir utiliser le graphique à basse résolution ; une connaissance pratique de la programmation en Basic (plus quelques connaissances en tracé de courbes) vous suffiront pour démarrer.

ÉTABLISSEMENT DE LA PAGE GRAPHIQUE

Pour passer au mode graphique à partir du mode texte, et en Basic, vous utiliserez l'ordre :

```
GR
```

Une fois cet ordre exécuté, l'écran devient noir à l'exception des quatre lignes du bas qui contiennent du texte. Cette zone basse de l'écran est appelée la *fenêtre de texte.* Avec celle-ci au bas de l'écran, il reste encore de la place pour 40 des 48 rangées disponibles en graphique à basse résolution. Vous pourrez utiliser cet ordre plusieurs fois dans un programme, même lorsque vous êtes en mode graphique à basse résolution, afin d'expliciter l'écran.

Tableau 6.1. — Couleurs pour graphismes à faible résolution

Couleur	Numéro	Couleur	Numéro
Noir	0	Marron	8
Magenta	1	Orange	9
Bleu foncé	2	Gris n° 2	10
Pourpre	3	Rose	11
Vert foncé	4	Vert clair	12
Gris n° 1	5	Jaune	13
Bleu moyen	6	Vert-bleu	14
Bleu léger	7	Blanc	15

Graphique plein écran

Après avoir exécuté l'ordre GR, vous pouvez éliminer la fenêtre de texte pour permettre aux huit dernières lignes de graphique d'intervenir en faisant :

```
POKE -16302,0
```

La fenêtre de texte disparaît, remplacée par du graphique.

Restauration de la fenêtre de texte

Lorsqu'Apple II est en mode graphique plein écran, vous pouvez rétablir la fenêtre de texte de deux façons. Pour effacer l'écran et restaurer la fenêtre de texte simultanément, utilisez l'ordre GR. Si vous ne voulez que rétablir la fenêtre de texte sans toucher aux

40 lignes graphiques, entrez l'ordre :

```
POKE -16301,0
```

Une fois exécuté, cet ordre ré-ouvre la fenêtre de texte en bas de l'écran.

Retour au texte plein écran

Pour quitter le mode graphique à basse résolution et revenir au mode texte plein écran, frappez :

```
TEXT
```

L'affichage reviendra du graphique aux caractères. Alors que l'ordre GR efface l'écran lorsqu'il est exécuté, TEXT n'en fait rien. Rappelez-vous que le texte et le graphique emploient tous deux la même zone de mémoire. Une fois cet ordre exécuté, vous verrez probablement apparaître un écran plein de caractères sans aucun sens ; c'est parce que Apple II interprète maintenant le graphique comme si c'était du texte en mémoire. Effacez l'écran avec la séquence ESC – @, avec un CALL – 936 en Basic, ou en Applesoft, avec la commande HOME.

INSTRUCTIONS DE PROGRAMMATION EN GRAPHIQUE

L'Integer Basic et l'Applesoft reconnaissent tous deux des commandes graphiques à basse résolution pour tracer de simples coordonnées sur l'écran, modifier la couleur, ou tracer des lignes horizontales ou verticales de diverses longueurs. Ces commandes ne fonctionnent qu'en graphique à basse résolution en page 1 (appelée aussi page primaire). Si vous employez la page 2, vous éliminerez la possibilité d'utiliser ces ordres, qui économisent du temps, et devrez faire intervenir des ordres moins descriptifs tels que PEEK, POKE et CALL.

L'ordre COLOR

Dans le tableau 6.1, on voit qu'à chaque couleur correspond un nombre entre 1 et 15. C'est ce nombre que vous emploierez dans un ordre établissant la couleur dans le mode à basse résolution. Par exemple :

```
COLOR=13
```

commande un tracé en jaune. Si vous négligez de sélectionner une couleur, Apple choisira le noir, équivalent à COLOR = 0, par défaut. Bien que vous puissiez choisir un nombre jusqu'à 255 pour la couleur sans provoquer une erreur de syntaxe, COLOR ne prêtera attention qu'au *quartet* de faible poids du nombre sélectionné. Ainsi, si vous entrez COLOR = 222, égal à DE en hexadécimal, il comprendra COLOR = 15 (égal à 0E en hexadécimal).

L'ordre PLOT

Cet ordre Basic place un simple point graphique — réellement, un petit rectangle — sur l'écran de l'Apple II aux coordonnées spécifiées.
L'ordre :

```
PLOT 23,18
```

allume le point graphique à l'intersection des 24e rangée et 19e colonne dans la teinte sélec-

tionnée par la dernière instruction COLOR exécutée. La gamme des numéros de rangées va de 0 à 47 et celle des colonnes de 0 à 39. Si vous excédez ces valeurs, dans un ordre PLOT, vous obtiendrez un message d'erreur et votre programme stoppera. Comme avec toutes les instructions graphiques en basse résolution (sauf GR), vous pourrez remplacer les expressions litérales par des variables :

```
PLOT Y/2+12,X-4
```

Exemple d'application de Plot

Le programme suivant, en Integer Basic, emploie toutes les instructions graphiques à basse résolution discutées jusqu'ici. Il trace une ligne diagonale de l'angle gauche supérieur à l'angle droit inférieur de l'écran :

```
10 REM TRACE D'UNE DIAGONALE
11 REM EN BASSE RESOLUTION
20 GR
30 COLOR= RND (16)
40 FOR Y=0 TO 39
50 PLOT Y,Y
60 NEXT Y
70 GOTO 30
```

L'écran va ressembler à ce que montre la figure 6.2, sauf que la couleur de la ligne varie aléatoirement.
Pour convertir ce programme en Applesoft, modifiez ainsi la ligne 30 :

```
30  COLOR=  RND (16) * 16
```

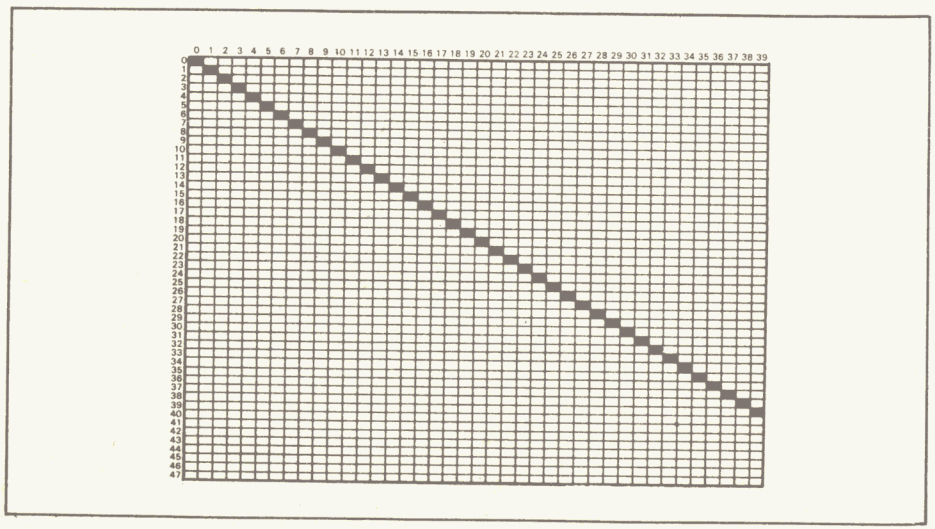


Fig. 6.2. — Exemple de tracé en basse résolution.

Tracé de lignes horizontales

L'instruction HLIN vous permet de tracer des lignes de diverses longueurs de gauche à droite, sur la page graphique à basse résolution. L'ordre :

```
HLIN 0,39 AT 0
```

trace un trait horizontal au bord supérieur de l'écran, de la gauche vers la droite. HLIN signifie « ligne horizontale ». Le format général de cette instruction demande l'entrée de la colonne de gauche, d'une virgule, puis de la colonne de droite (39, ici) pour tracer une ligne horizontale, enfin le mot AT (« à ») et le numéro de la rangée sur laquelle tracer la ligne.
Les paramètres colonnes, gauche et droite, ne peuvent être négatifs et doivent être inférieurs à 256 ; le numéro de la colonne de droite ne peut être inférieur à celui de la colonne de gauche. Le paramètre suivant AT ne peut être négatif ou supérieur à 48. Si l'un de ces paramètres excède les valeurs permises, un message d'erreur apparaîtra, stoppant l'exécution. Si vous spécifiez un numéro de colonne supérieur à 39, dans l'instruction HLIN en Integer Basic, vous aboutirez à des résultats imprévisibles. L'Applesoft vous fournira, lui, un message d'erreur. Exécutez cet ordre, par exemple :

```
10 GR

20 COLOR=12

30 HLIN 45,100 AT 0
```

En Integer Basic, deux barres vertes seront affichées. La première ne sera pas sur la rangée 0 (la ligne supérieure de l'écran) où elle devrait se trouver, et la barre se continuera plusieurs lignes plus bas sur l'écran. Ce n'est pas une bonne idée que de laisser ces valeurs excéder les positions réelles de l'écran en basse résolution.

Tracé de lignes verticales

L'instruction VLIN (pour « ligne verticale ») trace une ligne dans la couleur sélectionnée, d'une rangée à une autre et ce, dans la colonne spécifiée.
Par exemple :

```
VLIN 12,30 AT 33
```

trace une ligne de la rangée 12 à la rangée 30 sur la colonne 33. Les valeurs des paramètres de VLIN sont les mêmes que pour HLIN ; la première et la seconde expression doivent être un entier entre 0 et 255 inclusivement, le second paramètre ne pouvant être inférieur au premier, et le dernier paramètre (correspondant au numéro de colonne) doit être compris entre 0 et 39 inclusivement. Si l'un de ces paramètres sortait de la gamme autorisée, un message d'erreur serait affiché.

Utilisation de HLIN et VLIN dans un programme

Le programme ci-dessous montre l'usage de HLIN et VLIN. Ici, des lignes aléatoires apparaissent sur l'écran dans des couleurs aléatoires. Pour stopper le programme, pressez CTRL – C.

```
10 REM USAGE DE HLIN ET VLIN EN BASSE RESOLUTION
20 GR : REM UTLISATION DU GRAPHIQUE PAGE 1
```

```
30 POKE -16302,0: REM POUR PLEINE PAGE
40 CALL -1998: REM EFFACER 48 RANGEES
50 REM DEBUT PROGRAMME
60 COLOR= RND  (16): REM COULEUR ALEATOIRE
70 HLIN 0, RND (40) AT RND (48)
80 COLOR = RND (16)
90 VLIN 0, RND (48) AT RND (40)
100 GOTO 60
```

L'ordre SCRN

L'ordre SCRN est un peu plus subtil que les autres instructions en mode en basse résolution. Supposez que vous vouliez demander à l'ordinateur d'indiquer quelle couleur est affichée en un point donné de l'écran. C'est SCRN qui interviendra. L'ordre :

```
X=SCRN(12,24)
```

attribue à la variable X le numéro de la couleur du point dont les coordonnées sont données entre parenthèses (ici, la 13e rangée et la 25e colonne). La couleur retournée affectée à la variable est numérotée de 0 à 15 ; le numéro correspond à l'une des couleurs en basse résolution du tableau 6.1. Par exemple, si vous entrez l'ordre suivant en mode immédiat :

```
GR
COLOR=14
PLOT 12,12
PRINT SCRN (12,12)
```

Apple II répondra :

```
 14
```

Cette instruction vous sera très utile lorsque vous rédigerez des programmes complexes graphiques en basse résolution.

GRAPHIQUE HAUTE RÉSOLUTION

L'aspect le plus attrayant d'Apple II réside dans ses facultés graphiques à haute résolution. Comme pour le graphique à basse résolution, vous disposez de deux zones distinctes, ou pages. Cependant, les similitudes s'arrêtent ici. Dans ce mode, la résolution est de 280 positions horizontales et 192 verticales, une amélioration de 7 fois sur l'axe horizontal et 4 fois sur l'axe vertical. Bien qu'un nombre moindre de couleurs soit disponible en haute résolution, vous pouvez tracer des lignes plus fines sur l'écran.
Seules les fonctions graphiques haute résolution incluses dans l'Applesoft sont disponibles. L'Integer Basic n'a pas de commandes intrinsèques de ce type. Cependant, et quel que soit le langage employé, l'ordinateur sera toujours capable de traiter du graphique à haute résolution car Apple II utilise une partie de sa mémoire pour stocker des points en haute résolution, des lignes et des tracés et comprend des programmes internes qui interprètent et affichent cette mémoire sur l'écran TV. Certaines instructions Applesoft emploient automatiquement ces programmes internes et la mémoire de l'écran ; nous discuterons d'abord d'eux. Ensuite, vous verrez comment introduire des graphismes en haute résolution dans vos programmes en Integer Basic. Vous trouverez enfin quelques trucs pour haute résolution utilisables *uniquement* avec l'Integer Basic.

QUELLE PAGE UTILISER ?

L'utilisation des graphismes à haute résolution présente quelques difficultés, liées au volume mémoire dont dispose votre Apple II. Si vous disposez du firmware Applesoft (en ROM, ou du « Language System » Apple II), vous pourrez utiliser en haute résolution la page 1 si votre ordinateur contient plus de 16 K, et la page 2 s'il dispose de 24 K ou davantage. Ajouter 12 K si vous devez employer aussi le DOS et le graphique haute résolution en même temps.
Si vous utilisez les versions d'Applesoft sur cassette ou disque, vous ne pourrez travailler avec la page 1 en haute résolution, car elle sert à loger une partie de l'interpréteur Applesoft. Si vous tentez d'utiliser la page 1 haute résolution, vous corromprez ou perdrez totalement la capacité de programmer en Applesoft aussi bien que le programme Basic résidant à ce moment en mémoire. Vous *pouvez* utiliser la page 2 haute résolution si votre Apple II dispose d'au moins 24 K, ou 36 K si le DOS doit se trouver en mémoire en même temps.

Pour séparer la mémoire graphique haute résolution

L'Applesoft ne protège pas automatiquement la mémoire pour graphismes haute résolution. Quand un programme s'accroît, par l'adjonction de nouvelles instructions ou de variables, les risques de destruction de la page graphique s'accentuent. La façon de résoudre ce problème consiste à établir les deux pointeurs de mémoire, HIMEM: et LOMEM: à des valeurs qui protégeront la page, ou les pages graphiques haute résolution que vous emploierez. Ces pointeurs agissent comme des frontières que votre programme ne traversera pas, vous permettant aussi de créer deux zones de mémoire dont vous userez librement.
L'usage de HIMEM: et LOMEM: diffère en Applesoft et en Integer Basic. Vérifiez ces différences dans le résumé du chapitre 8. On trouvera aussi, figure G.1 de l'appendice G, la description de la mémoire pour usage graphique. Les trois paragraphes suivants seront plus faciles à suivre si vous vous y référez.
Si vous envisagez de travailler avec la page 1 haute résolution (rappelez-vous que vous devez disposer alors du firmware Applesoft), et si votre Apple II ne possède que 16 K, établissez HIMEM: à 8191 et laissez LOMEM: . De ce fait, votre programme Applesoft restera en dessous de la position mémoire 8191, ce qui constitue une restriction importante mais inévitable dans la mesure où la mémoire de votre écran haute résolution demande 8 K. Si vous employez la page 1 haute résolution avec un Apple II de plus de 16 K, vous pourrez ignorer HIMEM: et établir LOMEM: à 16384 si vous le désirez. Votre programme Applesoft se situera alors au-dessus de la page 1 haute résolution.
Attention : n'employez pas ce dernier schéma avec le DOS, sauf si vous disposez d'au moins 32 K de mémoire. Sinon, ne retenez que la première variante qui place votre programme Applesoft en-dessous de la page 1.
Si vous voulez recourir à la page 2 haute résolution (ce qui est possible avec 24 K ou davantage), établissez LOMEM: à 24576, ou encore HIMEM: à 16383 et laissez LOMEM: inchangé. En ajustant LOMEM:, vous placerez votre programme Applesoft *au-dessus* de la page 2 haute résolution. Pour utiliser en même temps le DOS, il vous faudra au moins 48 K de mémoire. Si, à la place, vous ajustez HIMEM:, votre programme résidera *en-dessous* de la page 2 haute résolution. Dans ce dernier cas, et si vous employez un interpréteur Applesoft non en firmware (sur cassette ou disque), votre programme Applesoft devra s'introduire au-dessus de l'interpréteur et en dessous de la page 2 haute résolution, ce qui pourra être exigu.
Enfin, si vous envisagez l'emploi des deux pages haute résolution pour le graphisme (ce qui n'est possible qu'avec le firmware Applesoft), situez LOMEM: à 24576 ou HIMEM: à

8191. Si vous placez votre programme Applesoft au-dessus des pages haute résolution en ajustant LOMEM: et en laissant HIMEM:, il vous faudra alors au moins 48 K pour permettre le rangement du DOS et disposer encore d'un espace significatif en mémoire pour vos programmes en Applesoft.

ÉTABLISSEMENT DE L'AFFICHAGE GRAPHIQUE

Bien que deux pages soient disponibles pour les graphiques à haute résolution, l'Applesoft en cassette ou disquette occupe une partie de la page 1 haute résolution pour le langage lui-même (l'interpréteur). Si vous possédez un Apple II Plus ou un Apple II standard, soit avec la carte firmware Applesoft, soit avec le « Language System », vous pourrez employer la page 1 sans affecter Basic. Pour préserver la compatibilité avec les autres Apple II, vous souhaiterez peut-être de toute façon recourir à la page 2 haute résolution. En Applesoft, l'ordre

```
HGR
```

efface puis affiche la page 1 haute résolution, avec une fenêtre de texte de quatre lignes en bas de l'écran. Dans ce mode, vous disposerez à la fois du graphique et du texte sur l'écran à l'aide des instructions PRINT, en Applesoft, pour afficher du texte dans la fenêtre. Cependant, lorsque Apple II exécute HGR, l'écran ne comportera que 160 des 192 lignes à haute résolution disponibles. Pour occuper tout l'écran en graphique, exécutez un POKE – 16302,0 après HGR, qui éliminera la fenêtre de texte et lui substituera les 32 lignes graphiques restantes.
Pour afficher la page 2 haute résolution, utilisez l'ordre :

```
HGR2
```

L'exécution d'HGR2 efface l'écran puis affiche les 192 lignes haute résolution de la page 2, sans fenêtre de texte en bas de l'écran. Pour ouvrir une telle fenêtre, exécutez un POKE – 16301,0 après HGR2. La fenêtre de texte est plus difficile à utiliser avec la page 2 haute résolution, cependant. Le texte apparaissant dans cette fenêtre provient de la page secondaire de texte. Basic ne peut accéder à cette page que via les ordres POKE (et *non* PRINT), ce qui est tout à fait restrictif. De plus, la page secondaire de texte n'est pas protégée contre les sur-impressions (sur-écritures) par le Basic comme l'est la page de texte 1, aussi devrez-vous établir LOMEM: à 3071 ou davantage.

ALTERNATIVES À HGR ET HGR2

Le principal inconvénient de HGR et HGR2 réside dans le fait que l'exécution de l'un de ces deux ordres efface la page haute résolution sélectionnée, que vous l'appréciez ou non. De plus, ces ordres ne sont pas disponibles en Integer Basic. Vous pouvez employer PEEK, POKE et CALL pour établir les pages graphiques avec plus de souplesse, et vous trouverez ces ordres bien utiles, quel que soit le langage que vous ayez adopté.

Une autre façon d'établir l'affichage graphique

Il est possible d'entrer dans le mode graphique haute résolution sans effacer l'écran d'affichage. Vous pourrez préférer cette procédure, faisant appel à des « commutateurs » sans existence physique réelle. En effet, vous allez établir une série de positions mémoires réservées, appelées *commutateurs logiciels,* qui se trouvent aux positions – 16304 à – 16297 (soit $C050 à $C057). La figure E – 1 de l'appendice E montre les commutateurs disponibles. Afin de conserver la compatibilité de ces ordres avec l'Integer

Basic et l'Applesoft, nous allons recourir à la représentation négative entière de ces positions mémoires (donc, à – 16304 au lieu de 49231, par exemple).
Pour afficher la page graphique 1 haute résolution sans effacer ses contenus précédents, exécutez les ordres suivants :

```
POKE -16304,0    Etablit le mode graphique
POKE -16297,0    Etablit le mode haute résolution
POKE -16300,0    Sélectionne la page 1 haute résolution
                   (N'est nécessaire que si l'on vient
                   de commuter de la page 2)
```

Essayez ces ordres en mode immédiat, à titre d'expérience.
Pour afficher la page graphique 2 haute résolution sans effacer ses contenus précédents, entrez les ordres suivants :

```
POKE -16304,0    N'est nécessaire que si le mode graphique n'a
                   pas encore été établi
POKE -16297,0    N'est nécessaire que si le mode haute
                   résolution n'a pas encore été établi
POKE -16299,0    Sélectionne la page 2 haute résolution
```

Basic normal après le graphique haute résolution

En Integer Basic, les ordres TEXT et GR peuvent se révéler insuffisants pour remettre complètement à zéro l'écran. Si vous aviez utilisé la page graphique 2, vous devrez explicitement re-sélectionner la page 1 avec un POKE – 16300,0, faute de quoi vous verrez apparaître la page 2 de texte basse résolution. C'est particulièrement déroutant en mode texte. Le clavier semble mort car tout ce que vous frappez va dans la page 1 de la mémoire d'écran, alors que vous observez la page 2 de la mémoire d'écran.
L'ordre GR n'exécute pas la commutation haute résolution à basse résolution, en graphique et en Integer Basic. Il se borne à choisir le mode graphique avec une fenêtre de texte de quatre lignes (et non plus le mode totalement texte). Vous devez explicitement sélectionner la basse résolution graphique avec un POKE – 16298,0.

Effacement des pages haute résolution

Si vous travaillez en graphique haute résolution sous Applesoft, HGR ou HGR2 effaceront la page sélectionnée à leur exécution. En Integer Basic, il n'existe pas d'instruction unique accomplissant cette fonction. Le sous-programme suivant appelle une fonction incluse dans le moniteur pour effacer l'écran haute résolution :

```
18990 REM ****************************************
18991 REM *   EFFACEMENT ECRAN HAUTE RESOLUTION  *
18992 REM *        (AVEC LE "MOVE" MONITEUR)     *
18993 REM *         SOUS-PROGRAMME EN $FE2C      *
18994 REM * POUR DEPLACER RAPIDEMENT LES DONNEES *
18995 REM *      DANS LA ZONE HAUTE RESOLUTION   *
18996 REM * ------------------------------------ *
18997 REM *      ETABLIR PAGE 1 OU 2; LE SOUS-   *
18998 REM *         PROGRAMME FERA LE RESTE      *
18999 REM ****************************************
19000 START=32
19010 IF PAGE=2 THEN START=64
```

```
19020 POKE 60,0
19030 POKE 61,START
19040 POKE 62,254
19050 POKE 63,START+33
19060 POKE 66,1
19070 POKE 67,START
19080 POKE -16304,0
19090 POKE -16297,0
19100 IF PAGE=1 THEN POKE -16300,0
19110 IF PAGE=2 THEN POKE -16299,0
19120 POKE START*256,0
19130 CALL -468
19140 RETURN
```

Ce sous-programme en Basic déplace des zéros à travers la page graphique haute résolution sélectionnée. Si la variable PAGE a été mise à 1, le sous-programme efface la page 1 ; si la variable est 2, la page 2 est effacée. Ce sous-programme ne fonctionne qu'en Integer Basic ; cependant, ne considérez pas que ce soit un désavantage. Applesoft exécute la même fonction bien plus efficacement avec HGR et HGR2.

COULEURS HAUTE RÉSOLUTION

En haute résolution, vous avez le choix entre huit couleurs, mais seulement quatre différentes sont disponibles, plus le noir et le blanc. Le tableau 6.2 montre ces couleurs et leurs numéros correspondants (qui serviront à les sélectionner).

Tableau 6.2. — Couleurs pour graphismes haute résolution

Couleur	Numéro	Couleur	Numéro
Noir	0	Noir	4
Vert	1	Orange	5
Violet	2	Bleu	6
Blanc	3	Blanc	7

L'ordre HCOLOR

L'ordre Applesoft HCOLOR sélectionne l'une des huit couleurs disponibles pour la haute résolution. A la différence de COLOR en basse résolution (qui vous permet de spécifier un numéro plus grand que celui attribué aux couleurs), HCOLOR n'acceptera pas un numéro de couleur supérieur à 7. Si vous spécifiez une couleur hors-gamme, votre programme stoppera brutalement avec un message ?ILLEGAL QUANTITY ERROR. HCOLOR ne modifie pas la couleur d'un graphisme déjà sur écran et n'affecte pas non plus les graphismes à basse résolution.

Établissement d'une couleur de fond en haute résolution

Avec une légère modification, le sous-programme présenté précédemment qui effaçait l'écran haute résolution remplira entièrement l'écran avec l'une des couleurs haute résolution. Ce sous-programme figure ci-dessous. Avant de l'appeler, attribuez un numéro de page graphique à la variable PAGE. Choisissez aussi le numéro de couleur haute résolution que vous attribuerez à la variable HCOLOR. Le programme d'appel doit positionner les commutateurs logiciels en mode graphique à haute résolution. Ce sous-programme ne fonctionne qu'en Integer Basic :


```
18990 REM ************************************
18991 REM * ETABLISSEMENT COULEUR DE FOND *
18992 REM * ----------------------------- *
18993 REM * ETABLIR PAGE = 1 OU 2;        *
18994 REM * ETABLIR AUSSI COULEUR EN      *
18995 REM * EQUIVALENT APPLESOFT          *
18996 REM * POUR HAUTE RESOLUTION         *
18997 REM ************************************
19000 START=32
19010 IF PAGE=2 THEN START=64
19020 POKE 60,0
19030 POKE 61,START
19040 POKE 62,253
19050 POKE 63,START+33
19060 POKE 66,2
19070 POKE 67,START
19075 Y1=85:X1=42
19080 IF HCOLR>0 AND HCOLR<>4 THEN 19090:X1=0:Y1=0
19090 IF HCOLR<>3 AND HCOLR<>7 THEN 19100:X1=127:Y1=127
19100 IF HCOLR=1 OR HCOLR=5 THEN 19120
19110 X3=X1:X1=Y1:Y1=X3
19120 IF HCOLR<4 THEN 19140
19130 Y1=Y1+128:X1=X1+128
19140 POKE START*256,X1
19150 POKE START*256+1,Y1
19160 IF PAGE=1 THEN POKE -16300,0
19170 IF PAGE=2 THEN POKE -16299,0
19180 CALL -468
19190 RETURN
```

TRACÉ DE POINTS ET DE LIGNES

Un puissant avantage de l'Applesoft en graphique haute résolution réside dans sa faculté de tracer des lignes selon n'importe quel angle aussi bien que des points individuels ou des lignes verticales et horizontales. L'ordre HPLOT peut servir de trois façons :

```
HPLOT 12,12
```

Cet ordre tracera un simple point dans la page haute résolution sélectionnée à l'intersection de la 13e rangée et de la 13e colonne, dans la couleur choisie.
Le second usage de HPLOT est :

```
HPLOT 0,0 TO 279,191
```

Cette instruction trace une diagonale partant de l'angle supérieur gauche à l'angle inférieur droit. Ainsi, l'utilisation de HPLOT avec deux jeux de coordonnées, comme ci-dessus, permet le tracé d'une droite d'un point à un autre de l'écran.
Le troisième usage est plus sophistiqué :

```
HPLOT 0,0 TO 279,0 TO 279,191 TO 0,191 TO 0,0
```

Cette version de HPLOT ne fonctionne qu'avec les versions firmware d'Applesoft. Les versions disque et cassette ne la permettent pas. Ici, vous pouvez définir des tracés multiples le plus simplement. Tous les segments apparaissent dans la même couleur, ce qui peut être très utile si vous voulez tracer une forme composée de plusieurs traits.

Alternatives à HPLOT

Le sous-programme ci-dessus vous permet de programmer du graphique haute résolution en Integer Basic, sans avoir à commuter la page. Ce sous-programme pourra vous être utile ; cependant, il est plutôt lent car tous les calculs se font en Basic. Pour l'utiliser, donnez les coordonnées X et Y du point à tracer. Assurez-vous que Page est positionné à la page haute résolution que vous voulez employer. Il vous appartient d'établir le mode haute résolution, graphique, et le mode plein écran, ce qui accroît la souplesse du traitement. Si vous appelez ce sous-programme *avant* d'avoir établi le mode haute résolution, il écrira dans la mémoire d'écran haute résolution mais ne changera rien à l'affichage. Ultérieurement, après que le programme ait terminé d'écrire dans la mémoire d'écran, il peut commuter vers le graphique haute résolution avec une série de POKE (comme on l'a décrit précédemment) et le tracé haute résolution qui s'était fait en mémoire deviendra soudainement visible. Cela vous permet de programmer des points en page 2 pendant que la page 1 haute résolution est affichée, ou vice-versa.

```
20000 REM ******************************
20001 REM *  TRACE HI-RES EN INTEGER   *
20002 REM * ------------------------- *
20003 REM *   ETABLIR X=COL, Y=RANG    *
20004 REM *      PAGE=1 OU 2; LES      *
20005 REM * VARIABLES SONT Y1, X1, X3 *
20006 REM ******************************
20007 REM
20010 Y1=PAGE*8192: REM ADRESSE BASE
20020 Y1=Y1+(Y/64)*40+(Y MOD 8)*1024
20030 Y1=Y1+(Y MOD 64/8)*128+X/7
20040 X1=PEEK (Y1): REM LECTURE OCTET HI-RESOLUTION
20050 X3=2 ^ (X MOD 7)
20060 REM 'OR' DE L'OCTET DE X1 AVEC
20070 REM LA VALEUR DU BIT DANS X3
20080 IF X1 MOD (X3 ^ 2)<X3 THEN X1=X1+X3
20090 POKE Y1,X3
21000 X1= PEEK (Y1)
21010 X3=2 ^ (X MOD 7)
21020 IF X1 MOD (X3*2)<X3 THEN X1=X1+X3
21025 X3=2 ^ (X MOD 7)
21026 GOTO 21100
21030 POKE Y1,X3
21040 RETURN
21100 POKE Y1,X3
21130 RETURN
25000 GOSUB 19000
```

Exemple de haute résolution en Integer Basic

Le programme suivant appelle deux des sous-programmes présentés récemment pour tracer des points sur l'écran haute résolution. Les commandes de jeu déterminent leurs coordonnées. En faisant tourner ces commandes, vous pourrez tracer des lignes sur l'écran.


```
10 REM LE PROGRAMME EMPLOIE DES SOUS-
20 REM PROGRAMMES SPECIAUX POUR EFFACER
30 REM ET TRACER EN HAUTE RESOLUTION
40 REM UTILISER CTRL-C POUR TERMINER
89 REM ETABLIR MODE GRAPHIQUE
90 POKE -16304,0
99 REM ETABLIR MODE HAUTE RESOLUTION
100 POKE -16297,0
109 REM SELECTION PLEIN ECRAN GRAPHIQUE
110 POKE -16302,0
200 PAGE=2
204 REM EFFACER MEMOIRE D'ECRAN HI-RESOL.
205 GOSUB 19000
209 REM APPEL COORDONNEES POINT
210 X= PDL (1)
220 Y= PDL (0)
229 REM TRACER POINT .HAUTE RESOLUTION
230 GOSUB 20010
239 REM NOUVELLES COORDONNEES
240 GOTO 210
260 END
```

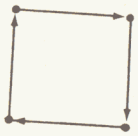
Pour créer une variante intéressante à ce programme, essayez d'appeler le sous-programme qui remplit l'écran d'une couleur donnée plutôt que le sous-programme d'effacement de l'écran.
Essayez d'améliorer ce programme en testant les boutons-poussoirs de commande de jeux avec des ordres POKE, et en effaçant l'écran chaque fois qu'un poussoir a été actionné.

UTILISATION DE FORMES HAUTE RÉSOLUTION

Avec des tracés répondant à des coordonnées, Apple II vous permet de définir, tracer et manipuler des formes en deux dimensions en mode graphique haute résolution. Cette section décrit comment créer, concevoir et utiliser une forme en Applesoft. Cependant, elle ne fait que commencer à explorer les possibilités de création qui s'ouvrent à vous.
Si vous avez déjà écrit des programmes graphiques en haute résolution qui tracent des figures géométriques, vous vous êtes probablement heurté à des difficultés pour manipuler ces figures sur l'écran. Par exemple, vous voulez faire pivoter une figure sur un axe, la réduire ou l'agrandir sur l'écran. Les formes haute résolution disposent de telles caractéristiques de manipulation.

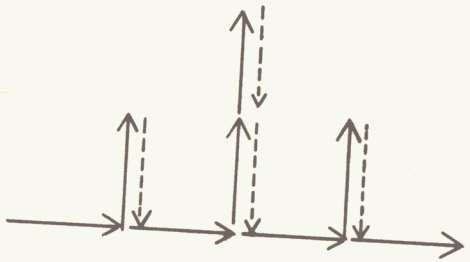
DÉFINITION DES FORMES

Les formes haute définition demandent un plan. Par essence, vous irez au-delà du fait de dire simplement à l'ordinateur de tracer simplement une ligne entre le point A et le point B. Lorsque vous utilisez une forme sur l'Apple II, vous décrivez une figure entière avant de demander à l'ordinateur de l'afficher. Vous définirez des formes haute résolution dans un *fichier de formes,* ainsi nommé car il contient les caractéristiques codées de la figure à tracer. La première étape dans la définition d'une forme en haute résolution consiste à la dessiner sur un papier. Par exemple, imaginez un carré, qui consiste en quatre lignes d'égale longueur, chacune à angle droit avec la précédente :



La table de formes contient les instructions codées pour tracer une figure ; ces instructions sont appelées *vecteurs de tracés*. Chaque vecteur décrit un mouvement vers le haut, le bas, la gauche, la droite, ou rien du tout, et également s'il faut l'afficher ou non. Vous pouvez interpréter chaque côté du carré de l'illustration ci-dessus comme une direction vers laquelle s'effectue le tracé : un vers le haut, un à droite, un vers le bas, et un vers la gauche. C'est de cette façon que le dispositif de manipulation des figures d'Apple II considère les figures.

Les figures sont plus difficiles à tracer si elles comprennent des diagonales ou des courbes. Un triangle, bien qu'il possède un côté de moins que le carré, implique plus d'efforts car il contient au moins une ligne diagonale. Les lignes tiretées de l'illustration suivante indiquent un mouvement sans tracé *(vecteurs fantômes)*.



Puisque vous ne pouvez définir des structures qu'avec des vecteurs se déplaçant vers le haut, le bas ou de côté, certaines formes, telles que des cercles, ne vaudront peut-être pas la peine d'être approximés. Dans certains cas, il pourra être plus simple de tracer des dessins compliqués avec HPLOT plutôt que d'employer des tables de formes.

ASSEMBLAGE DU FICHIER DE FORMES

La figure que vous avez dessinée sur un papier doit être convertie en vecteurs de dessin codés. Cette section vous explique comment procéder. La suivante vous présente un programme en Applesoft qui exécutera cette conversion pour vous. Aussi pouvez-vous sauter la présente section si vous n'êtes pas intéressé par le fonctionnement interne du fichier de formes.

Le codes des vecteurs couvrent une gamme allant de 0 à 7 ; chaque octet de définition d'une forme (faisant partie du fichier de formes) peut contenir jusqu'à trois vecteurs. Le tableau 6.3 montre les codes possibles des vecteurs. Une fois la structure du dessin réduite à un jeu de vecteurs, ceux-ci sont placés en mémoire, et de là, certaines commandes Applesoft pourront les décoder et commander l'affichage du dessin.

Adoptez un point de départ sur le dessin. Dressez une liste des vecteurs à tracer pour le reconstituer, en utilisant des flèches (↑←↓→). Listez les vecteurs dans l'ordre selon lequel vous parcourez le dessin (sens des aiguilles d'une montre, ou l'inverse, peu importe). Marquez chaque vecteur à tracer, mais non tracé (vecteurs fantômes). Si l'on commence par l'angle gauche inférieur de notre carré, on obtiendra les vecteurs suivants :

Direction	Tracé
↑	Oui
→	Oui
↓	Oui
←	Oui

Maintenant, rédigez les codes binaires correspondants à chaque vecteur (en vous référant au tableau 6.3). Vous obtiendrez :

Vecteur	Code
↑	100
→	101
↓	110
←	111

Comme le montre le tableau 6.4, chaque octet du fichier de formes contient trois sections, chacune d'elles pouvant contenir un vecteur à tracer. Notez que les sections 1 et 2 contiennent chacune trois bits, alors que la section 3 n'en contient que deux.

Tableau 6.3. — Vecteurs de dessin et codes binaires associés

Symbole	Action	Code binaire	Code décimal
⇡	Déplacement vers le haut, sans tracé	000	0
⇢	Déplacement vers la droite, sans tracé	001	1
⇣	Déplacement vers le bas, sans tracé	010	2
⇠	Déplacement vers la gauche, sans tracé	011	3
↑	Déplacement vers le haut, avec tracé	100	4
→	Déplacement vers la droite, avec tracé	101	5
↓	Déplacement vers le bas, avec tracé	110	6
←	Déplacement vers la gauche, avec tracé	111	7

Tableau 6.4. — Les octets de formes

	Section 3		Section 2			Section 1		
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
M = bit de mouvement P = bit de tracé/non tracé	M	M	P	M	M	P	M	M

En examinant le tableau 6.3, vous constaterez que quelques-uns des codes employés pour le dessin sont sur trois bits. Ceci convient aux sections 1 et 2 des octets définissant les formes. Cependant, la section 3, qui ne contient que deux bits, ne pourra stocker que cer-

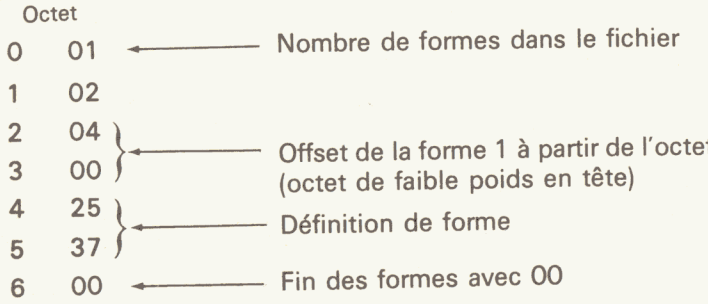
tains vecteurs. Les vecteurs qu'on peut attribuer à la section 3 sont vers la droite, la gauche, vers le bas, sans tracé. La section 3 ne reconnaît aucun autre vecteur.

La plupart du temps, vous trouverez que la section 3 est inemployée. Cela ne signifie pas que vous devez omettre de l'utiliser, mais très souvent, vous vous en passerez. Si la section 3 d'une définition de structure est à zéro, Applesoft ignore cette section, va sur l'octet suivant de la définition de forme et l'interprète pour son tracé.
Le tracé de vecteurs égaux à zéro peut signifier deux choses. Dans la section 3 de chaque définition de forme, un vecteur de dessin à zéro signifie « pas de mouvement et pas de tracé ». Cependant, et dans le tableau 6.3, un vecteur à zéro signifie « déplacement sans tracé ». Cette ambiguïté peut être à l'origine de problèmes avec les sections 1 et 2 de chaque octet de définition de forme, car dans certains cas, Applesoft ignore les vecteurs de tracé à zéro et dans d'autres, il exécute un mouvement sans tracé. La règle sera donc de ne jamais terminer un octet de définition de forme avec un vecteur de tracé à zéro si vous supposez qu'il signifie « déplacement sans tracé ». Les programmes de traitement de dessin d'Applesoft supposent que si la portion la plus significative (section 3) ou les portions (2 et 3 ensemble) sont mises à zéro, aucune action de tracé ne doit intervenir pour ces sections à zéro.
Si les trois sections de l'octet de définition de forme sont à zéro, Applesoft en conclut qu'il s'agit d'une « fin de définition de tracé ». En fait, vous devrez terminer chaque définition de tracés par un octet de terminaison, à zéro. Sinon, Applesoft continuera à dessiner après la fin de votre dessin original, jusqu'à ce qu'il rencontre un octet à zéro.
Vous pouvez utiliser le vecteur « déplacement sans tracé » si un vecteur de tracé différent intervient ensuite dans le même octet. Par exemple, la section 2 peut être à zéro (pour un « mouvement sans tracé ») ; si la section 3 est à 01, à 10 ou à 11 (en binaire), cette section 2 sera reconnue comme signifiant effectivement « mouvement sans tracé ». Si la section 3 est mise à zéro, comme la section 2, aucun mouvement ne surviendra et Applesoft passera à la section 1 de l'octet suivant, au vecteur de tracé valide suivant.
Armé de ces connaissances, vous pouvez maintenant organiser le tracé des vecteurs codés en binaire pour chaque segment du dessin en groupes de deux ou trois. De cette façon, vous transposerez les codes trois bits des vecteurs de tracé en octets qui seront rangés en mémoire. Les vecteurs de tracé pour un carré entrent dans les définitions de formes comme le montre le tableau 6.5.
Avec les tracés désormais codés en octets, vous pouvez facilement convertir ces octets en notation hexadécimale. L'appendice J contient une table de conversion binaire à hexadécimal. Le tableau 6.5 montre le codage hexadécimal d'un tableau.
La définition de forme est désormais complète. L'étape suivante consiste à créer une série de pointeurs pour le tracé (et d'autres : jusqu'à 255 tracés) qu'Applesoft utilisera comme un répertoire.

Assemblage du répertoire du fichier des formes

Le répertoire d'un fichier de formes consiste en une série d'octets décrivant combien de formes contient le fichier, et pointant chaque définition de forme dans le fichier. Le premier octet du répertoire contient le nombre total de formes dans le fichier. Ce nombre va de 0 à 255 ($FF). Le second octet est inemployé et devra être mis à zéro.
Les autres octets du répertoire contiennent des pointeurs pour chaque définition se trouvant dans le fichier. Chaque pointeur est sur deux octets et contient l'offset (le décalage, c'est-à-dire la valeur absolue en octets) de l'octet à partir du *début* du fichier. L'octet de faible poids du pointeur précède l'octet de fort poids. Par exemple, si l'offset d'une forme est dix octets, le pointeur sera codé 0A 00. Dans le cas du carré, il n'y a qu'une

forme est dix octets, le pointeur sera codé 0A 00. Dans le cas du carré, il n'y a qu'une forme à lister dans le répertoire du fichier, aussi l'offset de la forme 1 depuis le début du fichier est-il de quatre bits. De ce fait, 04 00 commandera le déplacement dans cette section.



Une bonne coutume veut qu'on laisse des octets supplémentaires à la fin d'un répertoire de fichier de formes pour réserver de la place aux futurs pointeurs du fichier. S'il ne reste plus de place à la fin du fichier pour permettre une extension, vous serez obligé de réorganiser le fichier de formes complètement de façon à insérer un nouveau pointeur de forme. Même si vous n'aviez besoin que d'un fichier ne contenant que dix tracés, vous devriez laisser un espace libre à la fin du répertoire ; vingt octets supplémentaires vous permettront de loger dix autres pointeurs que vous pourrez utiliser ultérieurement. Si vous désirez ajouter un nouveau tracé dans le fichier, placez la nouvelle définition de forme juste après la dernière qui se trouvait dans ce fichier, calculez l'offset de cette nouvelle forme à partir du début du répertoire, placez le nouveau pointeur immédiatement après le dernier pointeur dans le répertoire, et ajouter 1 à l'octet 0 du répertoire (celui qui contient le nombre de tracés présents dans le fichier).
La figure 6.3 illustre la façon dont les fichiers de formes et leur répertoire sont organisés en mémoire.

Tableau 6.5. — Codage d'une forme (ici, le tracé d'un carré).

	Vecteurs			Codes binaires			Codes hexadécimaux
	Sect. 1	Sect. 2	Sect. 3	Sect. 1	Sect. 2	Sect. 3	
Octet 0	Aucun	↑	→	00	100	101	25
Octet 1	Aucun	↓	←	00	110	111	37
Octet 2	Aucun	Aucun	Aucun	00	000	000	00
Octet 3	—	—	—	—	—	—	—

Assemblage des vecteurs par l'ordinateur

Le programme suivant, rédigé en Applesoft, assemble la définition d'une forme pour vous. Il vous demande d'introduire chaque vecteur de tracé, et s'il doit être tracé ou non. Après l'entrée du dernier vecteur, frappez E pour « end » (fin) et pressez RETURN. Le programme vous demande d'entrer aussi un vecteur à modifier, s'il y en a. Si vous commettez la moindre erreur en introduisant les vecteurs de tracés, vous pourrez les corriger en entrant le numéro du vecteur de tracé puis en ré-entrant ce vecteur et en indiquant s'il faut ou non le tracer.

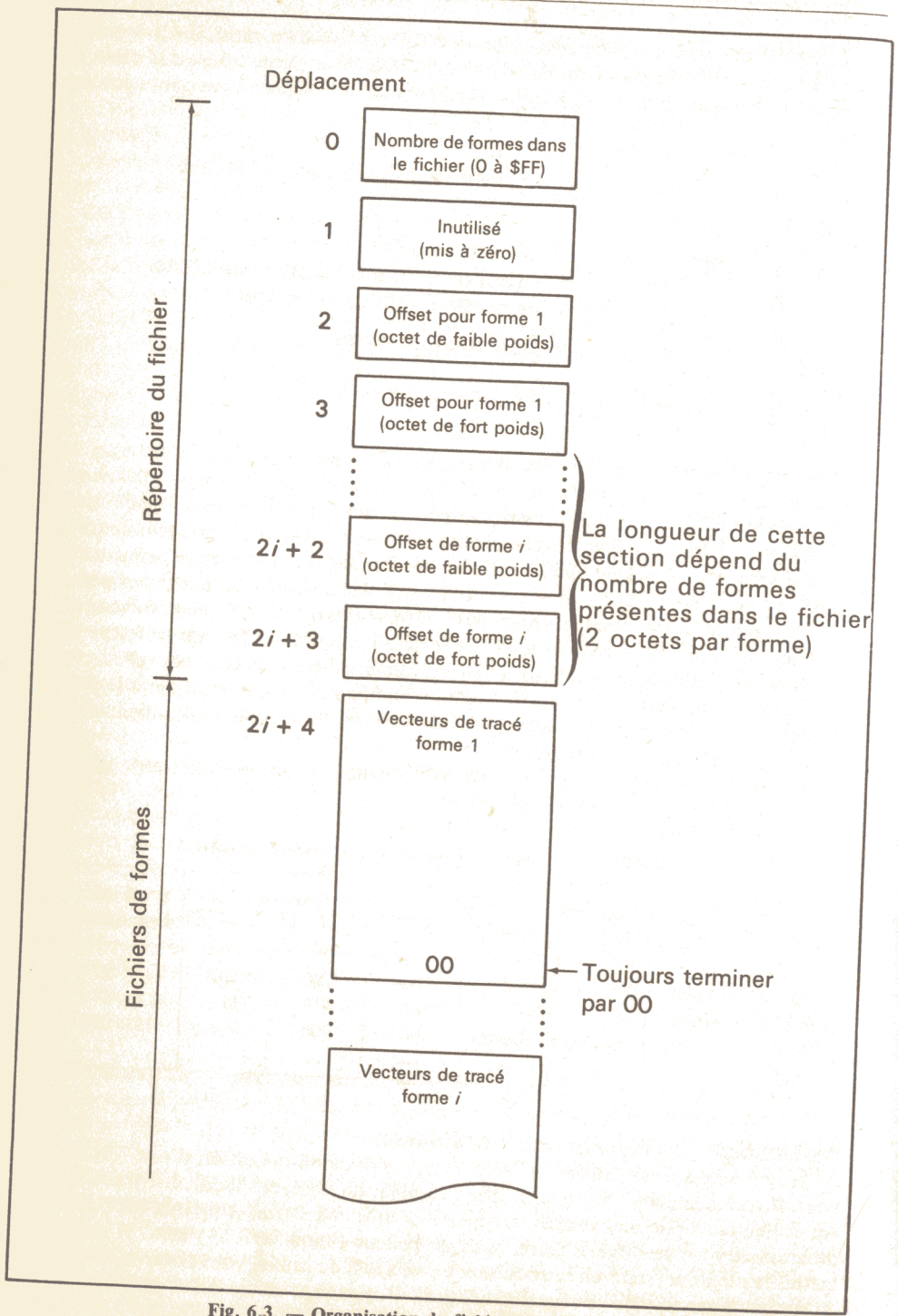


Fig. 6.3. — Organisation du fichier des formes.

S'il ne vous reste plus de corrections à faire, entrez 0 en réponse à VECTOR TO CHANGE (0 = END). Après quelques secondes, les vecteurs de tracé sont affichés en hexadécimal. Voici un listage du programme, accompagné de quelques exemples d'exécution.

```
1 REM ************************************
2 REM * PROGRAMME DE CREATION DE FORMES *
3 REM *                                  *
4 REM ************************************
10 DIM S1(100),V1(100)
20 I=0
30 PRINT "CREATION DE VECTEURS DE TRACES"
40 PRINT
41 REM ENTREE DES TRACES
50 V = I: GOSUB 270
58 REM CONTINUER L'ENTREE JUSQU'A CE QUE M$
59 REM EGALE LA VALEUR TERMINALE "E"
60 IF M$ < > "E" THEN S1(I) = M:I = I + 1: GOTO 50
70 PRINT
71 REM POUR PERMETTRE LES CORRECTIONS
80 INPUT "VECTEUR A MODIFIER (0=FIN):";V
90 IF V > 0 THEN V = V - 1: GOSUB 270:S1(V) = M: GOTO 80
99 REM RANGER VECTEURS DANS TABLEAU V1 ()
100 FOR V = 0 TO I
110 IF B = 2 AND S1(V) > 0 AND S1(V) < 4 THEN 140
120 IF B < 2 AND (S1(V) > 0 OR S1(V) > 4) THEN 140
130 B = 0:Q = Q + 1
140 V1(Q) = V1(Q) + S1(V) * (8 ^ B)
150 B = B + 1
160 IF B > 2 THEN B = 0:Q = Q + 1
170 NEXT V
178 REM AFFICHAGE DES VECTEURS
179 REM EN HEXADECIMAL
180 PRINT "OCTET", "VECTEUR"
190 FOR V = 0 TO Q
200 H% = V1(V) / 16
210 L% = V1(V) - H% * 16
220 IF H% > 10 THEN H% = H% + 7
230 IF L% > 10 THEN L% = L% + 7
240 PRINT V, CHR$ (H% + 176); CHR$ (L% + 176)
250 NEXT V
260 END
269 REM SOUS-PROGRAMME D'ENTREE VECTEURS
270 PRINT "VECTEUR ";V + 1;":";
275 PRINT"U=HAUT, D=BAS, L=GAUCHE, R=DROITE"
280 INPUT "DEPLACEMENT: U/D/L/R?";M$
290 M = 0
300 IF M$ = "R" THEN M = 1
310 IF M$ = "D" THEN M = 2
320 IF M$ = "L" THEN M = 3
330 IF M$ = "E" THEN RETURN
340 INPUT "TRACE (Y=OUI,N=NON)? ";P$
350 IF P$ = "Y" THEN M = M + 4: RETURN
360 IF P$ = "N" THEN RETURN
370 GOTO 340
```

```
]RUN
CREATION DE VECTEURS DE TRACES
VECTEUR 1:DEPLACEMENT: U/D/L/R? D
TRACE (Y=OUI, N=NON)? N
VECTEUR 2:DEPLACEMENT: U/D/L/R? D
TRACE (Y=OUI, N=NON)? N
VECTEUR 3:DEPLACEMENT: U/D/L/R? L
TRACE (Y=OUI, N=NON)? Y
VECTEUR 4:DEPLACEMENT: U/D/L/R? L
TRACE (Y=OUI, N=NON)? Y
VECTEUR 5:DEPLACEMENT: U/D/L/R? U
TRACE (Y=OUI, N=NON)? N
VECTEUR 6:DEPLACEMENT: U/D/L/R? U
TRACE (Y=OUI, N=NON)? Y
VECTEUR 7:DEPLACEMENT: U/D/L/R? U
TRACE (Y=OUI, N=NON)? Y
VECTEUR 8:DEPLACEMENT: U/D/L/R? U
TRACE (Y=OUI, N=NON)? Y
VECTEUR 9:DEPLACEMENT: U/D/L/R? R
TRACE (Y=OUI, N=NON)? N
VECTEUR 10:DEPLACEMENT: U/D/L/R? R
TRACE (Y=OUI, N=NON)? Y
VECTEUR 11:DEPLACEMENT: U/D/L/R? R
TRACE (Y=OUI, N=NON)? Y
VECTEUR 12:DEPLACEMENT: U/D/L/R? R
TRACE (Y=OUI, N=NON)? Y
VECTEUR 13:DEPLACEMENT: U/D/L/R? D
TRACE (Y=OUI, N=NON)? N
VECTEUR 14:DEPLACEMENT: U/D/L/R? D
TRACE (Y=OUI, N=NON)? Y
VECTEUR 15:DEPLACEMENT: U/D/L/R? D
TRACE (Y=OUI, N=NON)? Y
VECTEUR 16:DEPLACEMENT: U/D/L/R? D
TRACE (Y=OUI, N=NON)? Y
VECTEUR 17:DEPLACEMENT: U/D/L/R? L
TRACE (Y=OUI, N=NON)? N
VECTEUR 18:DEPLACEMENT: U/D/L/R? L
TRACE (Y=OUI, N=NON)? Y
VECTEUR 19:DEPLACEMENT: U/D/L/R? E

VECTEUR A MODIFIER (0=FIN):0
OCTET           VECTEUR
0               12
1               3F
2               20
3               64
4               2D
5               15
6               36
7               1E
8               07
9               00

]
```


INTRODUCTION DU FICHIER DE FORMES

Avant de pouvoir afficher les tracés que vous venez de coder, vous devez les introduire dans la mémoire de l'ordinateur. Pour cela, vous devrez déterminer dans quelle zone de la mémoire résidera le fichier des formes. La façon la plus simple de dégager cet espace consiste à repositionner le pointeur de HIMEM: à une valeur tout juste inférieure à l'adresse de départ du DOS, ou juste avant la page graphique haute résolution que vous souhaitez employer. Vous devez remettre HIMEM: à zéro *avant* d'exécuter un ordre Applesoft employant des chaînes. Si vous utilisez un Applesoft sur disque, vous aurez besoin d'au moins 36 K de mémoire (mais 48 sont préférables) pour que l'interpréteur Applesoft et le DOS soient aussi présents. Les adresses 115 et 116 ($73 et $74) contiennent la dernière valeur de HIMEM: pour Applesoft, rangée avec l'octet de faible poids d'abord. Pour calculer le nouveau HIMEM: vous accordant de la place pour le fichier des formes, utilisez l'ordre suivant :

```
PRINT PEEK(116)*256+PEEK(115)-X
```

Cette instruction calcule la valeur du HIMEM: que vous devrez établir, à partir du paramètre X qui est la longueur du fichier des formes, répertoire inclus. Pour l'utiliser, remplacez X par la longueur du fichier. Positionnez HIMEM: à la valeur calculée avant d'introduire le fichier en mémoire. Le fichier des formes sera ainsi protégé contre toute sur-écriture par l'Applesoft.

En variante, vous pouvez placer votre fichier de formes en mémoire entre les positions 768 et 975 ($300 et $3CF) inclusivement. Assurez-vous que le fichier n'entre pas en conflit avec des sous-programmes en langage machine que vous ou Applesoft auriez pu ranger là. Vous pouvez recourir à POKE pour placer le fichier en mémoire. Par exemple, la séquence suivante d'ordres POKE range le fichier des formes de notre carré en mémoire à partir de la position 768 :

```
]POKE 768,01
]POKE 769,00
]POKE 770,04
]POKE 771,00
]POKE 772,37
]POKE 773,55
]POKE 774,00
```

Vous pouvez aussi introduire le fichier à partir du moniteur, avec l'ordre CALL – 151 pour commuter au moniteur. Entrez ensuite l'adresse mémoire *hexadécimale* à laquelle le fichier doit démarrer, suivi par un double point, puis entrez le premier octet du répertoire du fichier, puis un espace blanc, puis l'octet suivant du répertoire suivi par un autre espace, etc. Pressez RETURN. Introduisez alors un autre double point suivi par les octets du tableau de la première forme (en séparant les octets par un espace), puis pressez RETURN. Répétez cette dernière séquence pour toutes les formes du répertoire. Vous pourrez revoir ce travail en frappant l'adresse mémoire hexadécimale de départ, un point, et l'adresse hexadécimale de la fin du fichier des formes, puis en pressant RETURN. Si

vous souhaitez davantage d'informations sur le moniteur, voyez le chapitre 7. Voici comment vous auriez entré le tableau des tracés de notre carré :

```
JCALL -151
*                                  Appel du moniteur
*6000:01 00 04 00                  Entrez le répertoire du fichier des formes
*:2C 3E 00                         Entrez la forme 1
*6000.6006                         Test des entrées par
                                     affichage de la mémoire
6000- 01 00 04 00 2C 3E 00
*
```

Le fichier est maintenant en mémoire. La première entrée démarre à l'adresse de départ du fichier des formes (ici, $6000). Le double point (:) indique au moniteur qu'il doit placer la série des digits hexadécimaux en mémoire. Aussitôt après le double point vient le répertoire du fichier des formes : 01 (le nombre de formes dans le fichier), 00 (le second octet est inemployé), 04 et 00 (l'offset de la forme 1 à partir du début du répertoire, octet de faible poids d'abord). La ligne suivante commence avec un double point ; aucune adresse de départ n'est nécessaire si vous l'avez donnée dans une entrée précédente. Le moniteur rangera la série suivante de digits hexadécimaux immédiatement après les premières séries déjà entrées.
La dernière ligne demande au moniteur d'afficher les adresses mémoires $6000 à $6006. Le format de cette commande est : adresse de départ, suivie par un point (qui indique au moniteur d'afficher la mémoire), suivi par la dernière adresse à afficher. Vérifiez soigneusement la précision de ces entrées chaque fois que vous employez le moniteur pour entrer le fichier des formes.

Sauvegarde du fichier des formes sur bande ou disque

Si vous avez investi beaucoup de votre temps à dresser des tables de formes, vous trouverez certainement intelligent de sauvegarder votre travail sur un support magnétique plutôt que de le laisser perdre quand vous mettre votre Apple II hors tension. Vous pouvez vous servir d'une cassette ou d'une disquette pour sauvegarder votre fichier ; la disquette est préférable car elle enregistre et retrouve plus vite les données que la bande.
Applesoft dispose de la commande appelée SHLOAD qui fonctionne uniquement avec la bande. Avant de sauvegarder le fichier des formes sur bande, vous devez calculer la longueur du fichier (en hexadécimal). Si l'on reprend l'exemple de notre carré, on peut examiner le nombre d'octets que demande le fichier des formes : la longueur totale est de 7 octets. En utilisant le moniteur, entrez la longueur du fichier sous forme d'un nombre hexadécimal sur deux octets, en commençant à l'adresse 0 :

```
*00:07 00
```

Comme chaque fois qu'on entre avec deux octets dans le moniteur, l'octet de faible poids vient d'abord. Puis, démarrez en mode RECORD (enregistrement). Employez le moniteur pour enregistrer la longueur du fichier puis le fichier lui-même, en deux opérations d'écriture que vous pourrez entrer sur la même ligne :

```
M9009'0009 MI'0*
```


La première commande d'écriture place la longueur du fichier, sur deux octets, que vous avez entrée (aux positions mémoires 0 et 1, ci-dessus) sur la bande. La seconde opération d'écriture range les sept octets du fichier, situés aux adresses $6000 à $6006, sur la bande. Ces deux opérations d'écriture prennent à peine plus de vingt secondes. Le haut-parleur émet deux « bip » pendant le processus d'écriture. Après le second, stoppez l'enregistreur. Le fichier des formes est maintenant en cassette et pourra vous servir plus tard. Utilisez maintenant la longueur réelle de votre fichier de formes, et ses adresses de départ et d'arrivée dans les exemples. Pour sauvegarder le fichier sur disque, entrez dans l'Applesoft sous DOS (à partir du moniteur, entrez 3DOG). La commande pour sauvegarder le fichier est BSAVE. Vous devez connaître l'adresse de départ et la longueur du fichier que vous avez entré. Pour sauvegarder le fichier commençant à l'adresse $6000, la commande DOS est :

```
BSAVE CARRE, A$6000, L7
```

Elle crée le fichier disque CARRE, avec le type B pour binaire. Le fichier des formes est désormais sur disque, prêt pour un nouvel emploi ultérieur. Utilisez le nom de ce fichier, son adresse de départ et sa longueur dans les exemples.

Chargement du fichier des formes à partir d'une bande ou d'une disquette

Si vous avez sauvegardé le fichier de formes sur cassette, Applesoft pourra la lire et la recharger en mémoire. Tout d'abord, rembobinez la cassette, puis frappez :

```
SHLOAD
```

Pressez la touche PLAY (lecture) de votre unité à cassettes. Apple II émet deux « bip » puis vous rend le contrôle des opérations. Si une difficulté de lecture a surgi, vous recevrez le message fatal « ERR » affiché sur l'écran. Dans ce cas, tentez une nouvelle lecture et, si l'erreur persiste, réglez le bouton de volume comme décrit dans le chapitre 2. SHLOAD positionne automatiquement HIMEM: à sa valeur courante moins la longueur du fichier de formes, en octets. Assurez-vous que HIMEM: a été positionné correctement.
Si votre fichier a été sauvegardé sur disquette, positionnez HIMEM: avant de recharger ce fichier en mémoire, avec la commande :

```
BLOAD CARRE
```

Le DOS se souvient de l'adresse précédente de rangement ($6000, dans notre exemple) et y placera le fichier automatiquement. Si vous souhaitez lire le fichier à partir d'une autre adresse, par exemple $3000, faites :

```
BLOAD CARRE, A$3000
```

A nouveau, employez le nom courant et l'adresse de départ optionnelle de votre fichier réel à la place de ceux donnés dans cet exemple.
Après le BLOAD du fichier, vous devez placer son adresse aux positions 232 et 233 ($E8 et $E9). Applesoft appelle ces adresses pour pointer le fichier de formes en mémoire (SHLOAD établit automatiquement ce pointeur). Entrez dans le moniteur et établissez l'adresse ($6000, par exemple) :

```
E8:00 06
```

En variante, vous pouvez faire des POKE pour ranger l'adresse de votre fichier de formes aux positions 232 et 233. Rappelez-vous qu'avec POKE, vous devez employer des valeurs décimales.
Vous êtes désormais prêt à exploiter votre fichier dans un programme Applesoft.

COMMANDES DE TRACÉS DE FORMES

Applesoft possède quatre instructions de manipulation de formes qui tracent, effacent et modifient leur orientation :

DRAW, qui affiche la forme sur l'écran ;
XDRAW, qui l'efface ;
ROT, qui opère une rotation sur les axes X et Y ;
SCALE, qui altère les dimensions du dessin.

Les commandes de manipulation de formes se réfèrent à la page graphique haute résolution sélectionnée courante (appelée par HGR ou HGR2) et à la couleur courante (avec HCOLOR).

La commande SCALE

En mode programmé ou immédiat, vous devrez toujours établir SCALE avant de tracer une forme pour la première fois dans un programme :

```
SCALE=1
```

établit le tracé sur un point pour chaque vecteur. Si SCALE = 5, Apple tracera 5 positions pour chaque vecteur individuel. Vous pouvez attribuer jusqu'à 255 (255 points par vecteur) à SCALE. Le maximum correspond à SCALE = 0, qui trace 256 points pour chaque vecteur.

La commande DRAW

DRAW trace la forme (numérotée de 1 à 255) extraite du fichier des formes, dans la dernière couleur choisie, à l'échelle et selon l'orientation des axes choisie précédemment. L'ordre :

```
DRAW 1 AT 140,96
```

commande le tracé de la forme définie dans le fichier de formes, en commençant à la 141e colonne et à la 97e rangée sur l'écran haute résolution. Le tracé débute à ces coordonnées. Une seconde option de DRAW se réfère à un emplacement implicite :

```
DRAW 11
```

Cet ordre exécute le tracé de la onzième forme du fichier au dernier point tracé par la plus récente instruction HPLOT ou DRAW exécutée. Si les coordonnées n'ont pas été établies antérieurement, Applesoft leur attribue par défaut les valeurs zéro.

IMPORTANT : Applesoft suppose que le fichier de formes est correctement rangé en mémoire. Avant d'exécuter un ordre DRAW, assurez-vous que c'est bien le cas et que le contenu des adresses 232 et 233 ($48 et $E9) pointe bien le début du fichier des formes. Si vous spécifiez un numéro de forme plus grand que le nombre de formes rangées dans le

fichier, ou si l'ordre DRAW donne des coordonnées de rangée et colonne non valides, le tracé ne se fera pas ; à la place, l'écran affichera un message d'erreur ?ILLEGAL QUANTITY ERROR.

La commande XDRAW

Cette commande sert à effacer un tracé sans effacer le reste. Par exemple :

```
XDRAW 8 AT 90,96
```

est un ordre dont la syntaxe est identique à celle de DRAW ; les coordonnées de départ peuvent être explicites ou implicites, comme ci-dessus. XDRAW teste la couleur aux coordonnées de tracé, puis trace une forme de couleur *complémentaire* à celle qu'il a trouvée. Dans l'exemple ci-dessus, XDRAW commence à la 91e rangée et à la 97e colonne sur l'écran. Le tableau 6.6 liste les couleurs complémentaires en haute résolution.
Si les coordonnées, la rotation et l'échelle sont les mêmes que celles qui avaient servi au tracé précédent, déjà sur l'écran, XDRAW l'efface, laissant intacts les autres tracés.

Tableau 6.6. — Couleurs des tracés

Si la couleur est :	La couleur de XDRAW sera :
Noir	Blanc
Blanc	Noir
Violet	Vert
Orange	Bleu
Vert	Violet
Bleu	Orange

La commande ROT

L'ordre ROT fait exécuter une rotation de la forme tracée par rapport au centre de l'écran (sur les axes X et Y). L'ordre :

```
ROT=16
```

ajuste l'angle de roration à 90 degrés dans le sens des aiguilles d'une montre. La gamme des valeurs de ROT va de 0 à 255, bien que seules 64 soient possibles, de 0 à 63. La figure 6.4 montre à quoi correspondent ces valeurs.
Lorsque SCALE est positionné à 1, ROT fait tourner les tracés par incréments de 90 degrés, ce qui signifie qu'il n'existe que quatre notations ROT qui sont : 0 = zéro degré, 16 = 90 degrés, 32 = 180 degrés et 48 = 270 degrés. Applesoft arrondit la valeur de rotation que vous avez posée à l'incrément inférieur suivant de ROT. Les 64 positions restent disponibles si SCALE est établi à 5 ou davantage.

Utilisation des formes dans un programme

Le programme suivant utilise des formes et servira d'exemple quant à la façon de les employer dans un programme :

```
1  LOMEM: 24600
20  HGR2
50 REM ADRESSE DE DEPART DE FICHIER DE FORMES
60 POKE 232,0
70 POKE 233,96
75 REM AVEC TOUTES LES COULEURS A TOUR DE ROLE
76 FOR H = 1 TO 7
79 REM POUR INCREMENTER LA ROTATION
80 FOR I = 1 TO 80
82  HCOLOR= H
90  ROT= I
92  IF I < 50 THEN  SCALE= I
105  DRAW 1 AT 140,96
106  ROT= I + 32
110  DRAW 1 AT 140,96
130  NEXT I
140  NEXT H
150  GOTO 76
```

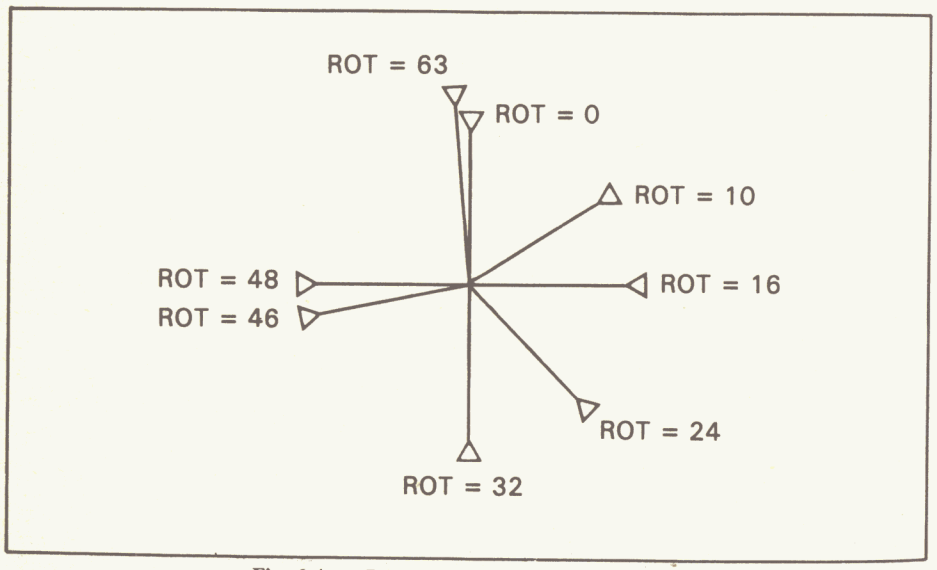


Fig. 6.4. — Les angles de rotation avec ROT.

L'usage de tracés haute résolution est particulièrement utile dans les programmes de jeux. Vous pourrez élaborer une bibliothèque de formes sans avoir à les documenter largement, comme vous auriez à le faire avec l'ordre HPLOT. Les formes réduisent la consommation de mémoire. Peu de calculs sont nécessaires pour manipuler la taille et l'orientation des objets haute résolution que vous créez, et les calculs eux-mêmes restent simples si on les compare à ce qu'il faudrait faire pour afficher une forme avec des instructions HPLOT. Si vous avez éprouvé quelques difficultés à parcourir cette section sur les formes à haute résolution, c'est probablement parce que vous n'avez pas essayé de construire des formes par vous-même. Il s'agit en effet de quelque chose d'assez complexe car de nombreux pas sont nécessaires pour aboutir à la moindre forme apparaissant sur l'écran. Si vous

employez le programme présenté précédemment et codant automatiquement les tracés, vous vous éviterez de la peine.

APPLE II ET LES SONS

Après ce qu'on vient de voir, ce pourra être une détente, ou un ennui, que de discuter de la programmation du haut-parleur incorporé à l'Apple II. Une détente, car la commande du haut-parleur est réellement très simple ; un ennui, car vous ne pouvez commander ce haut-parleur avec des instructions en Basic. Vous devrez définir en détail chaque son que le haut-parleur devra émettre. Par essence, tout ce que vous pouvez faire en programmant le haut-parleur, c'est de lui faire émettre de simples « clics ». L'astuce consistera à varier la fréquence de ces « clics » de façon à créer des sons de diverses hauteurs. Cette section explique comment piloter le haut-parleur, et inclut un programme créant une série de sons.

LA COMMANDE DU HAUT-PARLEUR

Apple II emploie la position mémoire – 16336, encore connue sous la référence 49200 et $C030, comme un commutateur, fort semblable aux commutateurs graphiques présentés antérieurement dans ce chapitre. Chaque fois que vous accéderez à cette position, un « clic » jaillira du haut-parleur de l'Apple II. En Basic, vous pouvez commander le haut-parleur en faisant :

```
A=PEEK(49200)
```

ou

```
A=PEEK(-16336)
```

Il est facile d'utiliser un programme Basic pour piloter le haut-parleur (comme le montre le court exemple ci-dessous), mais le haut-parleur n'émettra que des fréquences basses ; cela, parce que le Basic est relativement lent. En Integer Basic, la plus haute fréquence possible est d'environ 256 Hz, et en Applesoft, de quelque 72 Hz (cycles par seconde). La seule façon de générer des sons à toutes fréquences consiste à recourir à un sous-programme en langage machine commandant le haut-parleur.

```
9 REM LE H-P VA EMETTRE UN CLIC
10 A = PEEK (-16336)
20 GOTO 10
```

Le sous-programme son en langage machine

L'Applesoft et l'Integer Basic réservent de la place en mémoire, de 768 à 975 ($300 à $3CF) à des sous-programmes en langage machine et à des fichiers de formes sans qu'il soit besoin de modifier LOMEM: (le DOS utilise ces zones, éliminant les contenus précédents, lorsque vous appelez la disquette maître DOS).

Si vous n'avez pas encore étudié le langage assembleur et le langage machine, vous pourrez tout de même inclure les sous-programmes suivants dans vos programmes en Integer Basic ou Applesoft en frappant une série de touches hexadécimales. Pour ranger le sous-programme en mémoire, entrez dans le moniteur avec la commande CALL-151 ou en pressant RESET (*important :* si vous avez le DOS, appelez-le avant d'entrer dans le moniteur). Puis, introduisez le sous-programme suivant en langage machine :

```
>CALL -151

*F666G

!302:LDY 301

0302-   AC 01 03    LDY   $0301
! LDX 301

0305-   AE 01 03    LDX   $0301
! LDA #4

0308-   A9 04       LDA   #$04
! JSR FCA8

030A-   20 A8 FC    JSR   $FCA8
! LDA C030

030D-   AD 30 C0    LDA   $C030
! INX

0310-   E8          INX
! BNE 310

0311-   D0 FD       BNE   $0310
! DEY

0313-   88          DEY
! BNE 305

0314-   D0 EF       BNE   $0305
! DEC 300

0316-   CE 00 03    DEC   $0300
! BNE 302

0319-   D0 E7       BNE   $0302
! RTS

031B-   60          RTS
```

L'ordinateur affiche des lignes grisées sur vos entrées

Ce sous-programme emploie les adresses $300 et $301 pour ranger des données ; les adresses $302 à $31B contiennent le programme lui-même. Une fois entré ce sous-programme, testez s'il est correct ainsi :

```
!$FF69G

*302L

0302-   AC 01 03    LDY   $0301
0305-   AE 01 03    LDX   $0301
0308-   A9 04       LDA   #$04
030A-   20 A8 FC    JSR   $FCA8
030D-   AD 30 C0    LDA   $C030
0310-   E8          INX
0311-   D0 FD       BNE   $0310
0313-   88          DEY
0314-   D0 EF       BNE   $0305
0316-   CE 00 03    DEC   $0300
0319-   D0 E7       BNE   $0302
031B-   60          RTS
031C-   20 20 70    JSR   $7020
031F-   08          PHP
0320-   18          CLC
0321-   D8          CLD
0322-   88          DEY
0323-   08          PHP
0324-   A0 A0       LDY   #$A0
0326-   10 38       BPL   $0360
* <-- CTRL-B en BASIC ici
```


Si vous avez le DOS dans votre Apple, vous pourrez sauvegarder ce sous-programme sur un disque avec la commande BSAVE en Basic.

```
BSAVE SON,A$302,L26
```

Ceci créera un fichier binaire appelé SON ; vous pourrez donc le recharger ultérieurement. Avec une cassette, enregistrez ce programme en entrant la commande moniteur suivante :

```
*302.329W
```

Le sous-programme sera sauvegardé sur disquette pour une lecture ultérieure.

L'interface Basic

Il vous faudra un sous-programme Basic chargé des échanges avec le sous-programme en langage machine. Entrez le listage suivant en Applesoft ou Integer Basic :

```
3200  REM COMMANDE DU HAUT-PARLEUR
3210  POKE 768,D
3220  POKE 769,F
3230  CALL 770
3240  RETURN
```

Avec ce sous-programme utilisé conjointement avec le programme en langage machine, un programme Basic pourra créer des sons en établissant deux variables : F (pour fréquence) ira de 1 à 255, avec 255 pour le son le plus aigu, et D (pour durée) allant aussi de 1 à 255, avec 255 pour la plus longue durée. Par exemple, introduisez le programme Basic suivant, après avoir introduit le sous-programme listé ci-dessus :

```
10  FOR I = 1 TO 254
20 F = 1
30 D = I
```

```
40  GOSUB 3200
50  NEXT I
60  END
```

Lorsque vous lancerez ce programme, écoutez la durée de chaque note. Sur les plus basses et les plus hautes fréquences de la gamme, les sons sont plus brefs qu'au milieu de gamme. Ce problème est inévitable car le sous-programme en langage machine recourt à des instructions plutôt qu'à des temps pour modifier la fréquence de chaque note. Du fait qu'Apple II ne possède pas d'horloge en temps réel, cette approche est nécessaire. En adoptant des durées plus longues pour les notes très basses ou très élevées, vous compensez ces inégalités.

Un programme sonore plus élaboré

Le programme ci-dessous recourt aux sous-programmes de création de sons détaillés ci-dessus afin de générer une série de sons que vous pourrez ré-écouter, modifier, et finalement sauvegarder pour usage dans d'autres programmes Basic. Lorsque vous lancerez ce programme, vous verrez les appels (E)NTER pour « entrer », (L)ISTEN pour « écouter » et (P)RINT pour « afficher ». La première action consistera à entrer quelques tons ; frappez E, et pressez RETURN.

Puis l'appel TON 0 : ENTREZ FREQUENCE ET DUREE ? apparaît. Entrez deux nombres séparés par une virgule. Le premier est la fréquence et le second, la durée. Tous deux doivent être compris entre 1 et 255. Lorsque vous pressez RETURN après avoir introduit la fréquence et la durée, le haut-parleur d'Apple II émet ce son. Ce processus se répète pour une série de sons (jusqu'à 100). Après avoir introduit le dernier ton, entrez un ton 0 et une durée 0 pour terminer.

Après quoi, l'appel QUELLE NOTE DOIT ETRE CHANGEE ? apparaît. Si vous voulez en modifier une, entrez son numéro, sinon, faites 0. L'appel (E)NTER, (L)ISTEN, (P)RINT ? apparaît à nouveau dès lors que vous avez terminé vos modifications ; entrez maintenant L et et pressez RETURN. Apple II répète la série de notes que vous avez introduites. Lorsque la dernière a été jouée, le même message revient. Entrez P pour afficher la fréquence et la durée de chaque son.

Le programme ci-dessous peut aisément être modifié afin de permettre sa sauvegarde sur disquette ou cassette. Ainsi, d'autres programmes pourront retrouver ces sons et, à l'aide des sous-programme de création de sons, joueront de la musique ou émettront des bruits aléatoires.

```
 10 REM PROGRAMME CREANT DES SONS
 19 REM AVEC UN TABLEAU DE SONS
 20 DIM A(100,2)
 29 REM EFFACEMENT ECRAN
 30 CALL - 936
 40 INPUT "(E)NTER, (L)ISTEN, (P)RINT?";A$
 50 IF A$ = "L" THEN 1000
 60 IF A$ = "P" THEN 1200
 80 IF A$ < > "E" THEN 30
 81 REM ENTREZ CHAQUE SON
 90 PRINT
100 I = 0
105 M = I
110 GOSUB 3000
119 REM FIN DE L'ENTREE DES SONS?
120 IF F = 0 AND D = 0 THEN I = I - 1: GOTO 200
```



```
129 REM NON -- METTRE SON EN MEMOIRE
130 A(I,1) = F:A(I,2) = D
140 I = I + 1
150 GOTO 105
200 REM POUR TOUS CHANGEMENTS
205 PRINT "QUELLE NOTE DOIT ETRE CHANGEE (0-";I;")";
206 INPUT E
208 IF E = 0 THEN 30
210 IF E < 1 OR E > I THEN 210
220 M = E: GOSUB 3000
230 A(E,1) = F:A(E,2) = D: GOTO 205
1000 REM ECOUTE DES NOTES ENREGISTREES
1010 FOR K = 0 TO I
1020 F = A(K,1):D = A(K,2): GOSUB 3200
1030 NEXT K
1040 GOTO 30
1200 REM AFFICHAGE DES NOTES
1210 PRINT "NOTE#","FREQ","DUREE"
1220 FOR K = 0 TO I
1230 PRINT K,A(K,1),A(K,2)
1240 NEXT K
1250 PRINT
1260 GOTO 30
3000 PRINT "TON ";M;
3010 INPUT "ENTREZ FREQUENCE, ET DUREE";F,D
3015 IF F = 0 AND D = 0 THEN RETURN
3020 IF (F < 0 OR F > 255) OR (D < 1 OR D > 255) THEN 3010
3030 GOSUB 3200
3040 RETURN
3200 REM COMMANDE DU HAUT-PARLEUR
3210 POKE 768,D
3220 POKE 769,F
3230 CALL 770
3240 RETURN
```

CHAPITRE 7

MONITEUR EN LANGAGE MACHINE

Dans la mémoire à lecture uniquement (ROM) de l'Apple II réside, en permanence, un programme de commande appelé le moniteur. Ce chapitre décrit ses caractéristiques et son usage, et montre comment vous pourrez y recourir et l'employer en conjonction avec les programmes que vous rédigerez en Basic. Le moniteur est écrit en langage machine ; il sert de lien entre le Basic (et d'autres langages que l'Apple II supporte) et les nombreuses fonctions à bas niveau que la machine exécute, et telles que l'impression d'un caractère, le tracé d'une ligne, etc.

Vous pouvez aussi employer le moniteur via les commandes du clavier. Quelques raisons intervenant en ce sens seraient la création de fichiers de formes graphiques (décrits dans le chapitre 6), l'examen de la mémoire pour isoler des problèmes matériels, ou la programmation en langage assembleur. Le plus souvent, vous ne vous sentirez pas pressé de l'utiliser, mais le moniteur dispose de fonctions qui vous paraîtront parfois bien commodes.

Après avoir décrit le moniteur et ses particularités, ce chapitre explique comment se servir du mini-assembleur. Il ne vous enseignera pas la programmation en langage assembleur ; l'appendice K liste quelques ouvrages spécialisés. Vous apprendrez comment intégrer votre programme en assembleur à un programme Basic, et comment exploiter le mini-assembleur pour l'écrire, le tester et le mettre au point.

L'ACCÈS AU MONITEUR

Il existe deux versions du moniteur, la version standard et l'Autostart. Si votre Apple II dispose de la version standard, le simple fait de le mettre sous tension vous placera dans le moniteur. Dans ce cas, vous verrez un écran rempli de caractères aléatoires, avec un astérisque (*) à l'angle gauche inférieur et un curseur clignotant à côté. L'astérisque est le caractère d'appel du moniteur.

Si votre Apple II est doté du moniteur autostart (soit sous forme d'équipement supplémentaire, soit en standard sur un Apple II Plus), il vous faudra passer par l'Applesoft ou l'Integer Basic pour accéder au moniteur. Lorsque vous verrez apparaître le caractère d'appel (> pour l'Integer Basic ou] pour l'Applesoft), frappez la commande :

```
CALL -151
```

Cet ordre est réellement un appel de sous-programme à l'adresse hexadécimale FF69, mais Basic ne reconnaît pas les valeurs hexadécimales, aussi devez-vous employer l'équivalent décimal de FF69. Une fois cet ordre entré, l'astérisque et le curseur apparaîtront. Vous êtes dans le moniteur.

POUR QUITTER LE MONITEUR

Il existe plusieurs façons de sortir du moniteur et revenir au Basic ; elles dépendent de ce que vous voulez faire à sa sortie. Pour préserver les instructions du programme et les variables de votre programme Basic, quittez le moniteur en pressant CTRL-C puis RETURN. Après cela, le caractère d'appel Basic apparaît. Dès lors, vous pourrez afficher les valeurs des variables et lister le programme (s'il y en a en mémoire à ce moment). Pour illustration, supposons que vous ayez rangé une valeur dans une position mémoire donnée, via un ordre POKE, et vous voulez vérifier ce POKE. Naturellement, vous pourriez employer PEEK pour cela, mais le moniteur vous donne bien plus d'accès à la mémoire d'Apple II que PEEK et POKE ne le feront jamais. L'exemple en Integer Basic ci-dessous montre que l'usage de CTRL-C préserve les programmes et les variables lorsqu'on quitte le moniteur :

```
>10 A=123
>19 REM METTRE LE CURSEUR SUR LA 13EME COLONNE
>20 POKE 36,12
>30 PRINT A
>40 END
>RUN
           123

>CALL -151

* <---- Pressez CTRL-C, puis RETURN

>PRINT A
123

>LIST
    10 A=123
    19 REM METTRE LE CURSEUR SUR LA 13EME COLONNE
    20 POKE 36,12
    30 PRINT A
    40 END
>
```

Si vous voulez sortir du moniteur et effacer les programmes Basic courants, ainsi que les variables, rangés en mémoire, pressez CTRL-B puis RETURN. L'entrée se fera dans le Basic mais toutes les instructions et données qui étaient en mémoire sont effacées. Essayez l'exemple ci-dessus, mais en substituant CTRL-B à CTRL-C. Après retour au Basic, vous vérifierez qu'il n'existe plus de valeur pour A, et que vous ne pouvez plus rien lister.
CTRL-C fonctionne fort bien avec l'Integer Basic et le *firmware* Applesoft, mais non avec l'Applesoft sur cassette ou disquette.
Avec un Applesoft sur disquette ou cassette, vous devrez faire CTRL-B pour sortir du moniteur. Dans ce cas, vous retournerez à l'Integer Basic, et vos programmes et variables en Applesoft seront effacés. Mais il existe une méthode pour retourner à l'Applesoft qui laisse intacts vos programmes et variables.
Pour revenir à l'Applesoft sur disque à partir du moniteur, faites :


```
*3D0G
```

Pour revenir à l'Applesoft sur cassette à partir du moniteur, faites :

```
*0G
```

3D0G et 0G sont deux instructions de branchement du moniteur (analogues au GOTO du Basic). On en verra davantage plus loin.
IMPORTANT : N'utilisez la commande 0G *qu'avec* l'Applesoft sur cassette.

FONCTIONS DU MONITEUR

Le moniteur exécute un nombre limité de tâches, mais chacune est remarquablement puissante si l'on considère la petitesse du programme moniteur. Vous pouvez examiner les positions mémoires ou les registres du microprocesseur, vider le contenu de la mémoire sur l'écran, ou vers un autre périphérique tel qu'une imprimante, et modifier les contenus des mémoires et des registres. Les autres fonctions comprennent le déplacement de blocs de données d'une adresse à une autre, et la comparaison de blocs de données en mémoire l'un avec l'autre. Il s'y ajoute des fonctions diverses, qu'on verra dans ce chapitre.

EXAMEN DE LA MÉMOIRE

Le moniteur emploie trois méthodes pour examiner le contenu des positions mémoires : les modes *adresse unique, mots* et *bloc.* Une adresse unique se réfère à une position mémoire de un octet. Des mots sont un segment de mémoire de huit octets, commençant par une adresse divisible par huit. Le mode bloc permet, de façon commode, d'examiner une série d'adresses, en commençant par une adresse et en terminant par une autre plus élevée.

Examen en adresse unique

Lorsque le caractère d'appel est présent et que vous voulez examiner une seule position mémoire, frappez simplement son adresse hexadécimale, puis RETURN, comme ici :

```
*FF69
```

Le moniteur répondra en affichant le contenu de cette position :

```
FF69- A9
*
```

Lorsque vous introduisez une adresse, le moniteur la retient pour un futur usage en tant que pointeur. Aussi, si vous entrez FF69 (ou toute autre valeur hexadécimale), le moniteur se souviendra de cette adresse et l'emploiera comme référence pour de futurs examens de mémoire, jusqu'à ce que vous changiez l'adresse. Par exemple, frappez :

```
*300F
```

Le moniteur attribuera au pointeur cette nouvelle adresse, outre le fait qu'il affichera son contenu.

Examen de mots en mémoire

Supposons maintenant qu'on ait examiné l'adresse FF69, avec l'exemple ci-dessus, et qu'on veuille vérifier le contenu des positions voisines supérieures en mémoire. Le moniteur continuera à examiner la mémoire si vous pressez RETURN, comme l'exemple suivant le montre :

```
*FF69

FF69- A9
* <-------- Pressez RETURN
 AA 85 33 20 67 FD
*

FF70- 20 C7 FF 20 A7 FF 84 34
* <-------- Pressez RETURN à nouveau
```

A la première action sur RETURN, six octets de mémoire sont affichés. Il s'agit des contenus des positions FF6A à FF6F. A la seconde action sur RETURN, huit octets sont affichés, mais l'adresse du premier d'entre eux est indiquée (FF70), afin que vous sachiez où se trouve le mot de départ. Les mots partent d'une adresse mémoire divisible par huit, c'est pourquoi aucune adresse n'a été affichée au premier RETURN. Celui-ci a simplement terminé le premier bloc que vous aviez attaqué avec l'examen d'une adresse unique.

Vous pouvez examiner de vastes blocs de mémoire (généralement, plus de huit octets) avec le mode bloc. Entrez l'adresse du début du bloc, en hexadécimal, suivie par un point, puis par l'adresse de fin de bloc, toujours en hexadécimal. Par exemple :

```
*F800.F83F
```

Le moniteur répondra en affichant les contenus des positions mémoires demandées :

```
F800- 4A 08 20 47 F8 28 A9 0F
F808- 90 02 69 E0 85 2E B1 26
F810- 45 30 25 2E 51 26 91 26
F818- 60 20 00 F8 C4 2C B0 11
F820- C8 20 0E F8 90 F6 69 01
F828- 48 20 00 F8 68 C5 2D 90
F830- F5 60 A0 2F D0 02 A0 27
F838- 84 2D A0 27 A9 00 85 30
*
```

L'adresse de fin doit être plus grande ou égale à l'adresse de début pour que vous obteniez plus d'une adresse. Si l'adresse de fin était plus petite que l'adresse de départ, seul le contenu de l'adresse de départ serait affichié.

Vous pouvez spécifier une gamme d'adresses excédant la capacité de l'écran d'Apple II. Dans ce cas, les données défileront sur l'écran pour faire de la place aux suivantes. Vous ne pouvez annuler l'affichage sans presser RESET. Si votre Apple II contient un moniteur autostart, vous pouvez temporairement faire cesser le déroulement des données en pressant CTRL-S. L'usage de CTRL-S stoppe la sortie vers l'écran, mais non vers d'autres périphériques tels qu'une imprimante, vous donnant l'occasion d'examiner tranquillement l'écran. Pressez la barre d'espacement pour poursuivre le défilement de l'affichage.

La méthode d'examen par bloc, est appelée un *vidage* (ou un « *dump* », dans le franglais — ou l'anglais tout court — des informaticiens). Une fois terminé, le pointeur qui conserve la position suivante à examiner se met à jour par lui-même de façon à pointer l'octet à l'adresse qui suit celle qui terminait le bloc.

Une forme condensée de cette commande emploie le pointeur pour adresse de départ du bloc. Il ne vous reste qu'à entrer un point, suivi par l'adresse de fin de bloc. Par exemple, si vous venez d'examiner les adresses F800 à F83F, comme dans l'exemple ci-dessus, vous pouvez poursuivre votre examen à partir de F840, jusqu'à F880, en frappant :

```
*.F880
```

Vous obtiendrez la sortie suivante :

```
F840- 20 28 F8 88 10 F6 60 48
F848- 4A 29 03 09 04 85 27 68
F850- 29 18 90 02 69 7F 85 26
F858- 0A 0A 05 26 85 26 60 A5
F860- 30 18 69 03 29 0F 85 30
F868- 0A 0A 0A 0A 05 30 85 30
F870- 60 4A 08 20 47 F8 B1 26
F878- 28 90 04 4A 4A 4A 4A 29
F880- 0F
*
```

EXAMEN DES REGISTRES DU MICROPROCESSEUR

Dans certains cas, vous voudrez inspecter les registres du microprocesseur lui-même. Vous ferez CTRL-E, suivi par RETURN, et vous obtiendrez un résultat tel que :

```
A=CD X=B1 Y=C3 P=B5 S=F0
```

Les valeurs affichées sont celles que contiennent l'accumulateur (A), le registre d'index X (X), le registre d'index Y (Y), le compteur ordinal (P) et le pointeur de pile (S). Les valeurs immédiatement à droite de chaque signe égal sont les plus récentes se trouvant dans ces registres. Cependant, elles ne sont pas affectées par le moniteur. En d'autres termes, les contenus des registres sont sauvegardés par le moniteur et ne changent pas, jusqu'à ce que vous exécutiez votre propre programme en assembleur ou retourniez au Basic.

POUR MODIFIER LA MÉMOIRE

Pour modifier les contenus de la mémoire, vous devez spécifier quelle adresse vous voulez changer et fournir la nouvelle donnée qui ira dans cette adresse. Vous pouvez agir par adresse unique (un octet à la fois), ou modifier une série de positions consécutives en mémoire, en introduisant de nouvelles données pour plusieurs adresses en une seule ligne de commande.

Modification d'une seule adresse

La première étape pour modifier une seule adresse consiste à établir le pointeur d'adresses du moniteur, qui est le même que le pointeur servant à examiner la mémoire. Parce que les deux commandes du moniteur visent le même pointeur, vous établirez l'adresse pour une modification tout comme pour un examen. Frappez l'adresse hexadécimale, puis RETURN. Par exemple :

```
*1200
```

établit le pointeur à l'adresse 1200 en hexadécimal.
Le moniteur répond en affichant le contenu de cette adresse :

```
1200- 73
```

Remarquez que cette entrée produit le même résultat que la commande d'examen en adresse unique. La réponse du moniteur montre où se trouve le pointeur d'adresses (à 1200, ici) et affiche également le contenu de cette adresse (ici, 73).
L'étape suivante consiste à modifier ce contenu en introduisant une nouvelle valeur. Pour changer le contenu de la mémoire à cette adresse, frappez d'abord un double point (:) suivi par le nombre hexadécimal sur deux digits que vous voulez ranger à cette adresse. Par exemple :

```
*:5F
```

Le double point indique au moniteur qu'on veut modifier une donnée. La valeur 5F est la nouvelle donnée à placer à l'adresse 1200. Vous pouvez aussi modifier le contenu d'une adresse en une seule ligne au lieu de deux, voici comment :

```
*1200:5F
```

Cette commande a le même effet que les deux précédentes. Le pointeur d'adresses se voit affecter l'adresse 1200, et le moniteur range 5F à cette adresse. Le pointeur se positionne ensuite à l'adresse voisine supérieure. Ainsi, si vous voulez mettre 3F en 1201, par exemple, frappez :

```
*:3F
```

Le moniteur attribuera automatiquement 3F à 1201. A nouveau, l'adresse du pointeur est incrémentée de 1 et vous pourrez modifier le contenu de l'adresse 1202, et des suivantes par le même processus sans avoir à entrer explicitement d'autres adresses.

Modifications de plusieurs positions mémoires

Le moniteur vous autorise à modifier les contenus de plusieurs positions mémoires en une seule fois, à la condition qu'il s'agisse d'adresses consécutives en mémoire (par exemple, les adresses 1200 à 1207 inclusivement). Cette commande débute de la même façon qu'une commande de modification d'une adresse unique. Vous établissez d'abord le pointeur d'adresses, si nécessaire. Puis, vous placez un double point, ce qui est l'ordre de modification de mémoire. Enfin, vous introduisez les données à ranger dans les cellules consécutives, en séparant chaque nombre hexadécimal par un espace.
Par exemple, pour ranger les valeurs 00 à 07 aux adresses 1200 à 1207, faites :


```
*1200:00 01 02 03 04 05 06 07
```

Vous pouvez réellement modifier plus de huit positions mémoires. Si vous établissez le pointeur au début de la commande, vous pouvez introduire jusqu'à 83 valeurs en une seule ligne de commande. Si vous n'avez pas besoin de positionner le pointeur, vous pourrez aller jusqu'à 84 valeurs. Dans les deux cas, la commande reviendra à la ligne et occupera plusieurs lignes d'affichage, ce qui n'est guère pratique car la vérification des entrées en devient plus difficile. De plus, la seule façon de corriger des erreurs consiste à revenir, en arrière, puis à refrapper les bonnes données et à terminer la ligne, ce qui est plus ennuyeux. Mais si votre tendance est d'établir des lignes longues, le moniteur vous le permet.

Vérification des modifications en mémoire

Une bonne coutume veut qu'on vérifie les modifications introduites en mémoire (qu'il s'agisse d'un fichier de formes graphiques ou d'instructions machine). Pour cela, vous userez de l'une des trois possibilités d'accès à la mémoire discutées plus haut dans ce chapitre. Pour commencer à tester les modifications introduites, il vous faut, à nouveau, repositionner le pointeur.
Supposons que vous ayez rangé 00 à 07 aux adresses 1200 à 1207, comme dans l'exemple ci-dessus ; ré-initialisez le pointeur à 1200 :

```
*1200
```

Le moniteur répond :

```
1200- 00
*
```

Pressez RETURN ; vous verrez les autres adresses que vous avez modifiées :

```
* ←—— PRESSEZ RETURN
 01 02 03 04 05 06 07
*
```

Si vous avez modifié davantage de positions, continuez à presser RETURN jusqu'aux dernières.
Vous pouvez aussi exécuter cet examen en mode bloc :

```
*1200.1207
```

Le moniteur répond :

```
1200- 00 01 02 03 04 05 06 07
*
```

Si vous avez commis des erreurs au cours des modifications, vous pouvez les corriger individuellement sans avoir à ré-entrer toutes les données. La façon la plus simple consiste à noter l'adresse de l'erreur et à introduire une modification en adresse unique.
Par exemple, si vous avez commis une erreur en entrant les nombres 00 à 07 de l'exemple précédent, et si l'erreur est à 1204, entrez :

```
*1204:04
```

Pour la corriger, l'étape suivante consistera à revenir en 1204 pour vérifier que cette fois, tout est correct. Terminez alors vos vérifications.

MODIFICATION DES REGISTRES DU MICROPROCESSEUR

Pour modifier le contenu des registres du microprocesseur, le processus est sensiblement différent car ces registres ne disposent pas d'adresses. Pour modifier leur contenu, il vous faut d'abord examiner ces registres avec une commande CTRL-E. Aussitôt après l'examen des registres, vous pouvez les modifier en frappant un double point (signifiant une opération de modification, à l'intention du moniteur), suivi par un à cinq nombres hexadécimaux. Séparez ces nombres par des espaces.

Le premier nombre hexadécimal sera la nouvelle valeur de l'accumulateur, le second ira dans le registre d'index X, le troisième dans le registre d'index Y, le quatrième dans le compteur ordinal et le cinquième, dans le pointeur de pile.

Vous devez entrer des valeurs pour tous les registres, jusqu'au dernier dans lequel vous voulez introduire une nouvelle valeur. Dès lors, vous pouvez abandonner les suivants.

Par exemple, supposons que vous vouliez changer le contenu du registre d'index Y sans toucher aux autres. D'abord, examinez ces registres :

```
*  <------ Pressez CTRL-E, puis RETURN

 A=CD X=B1 Y=C3 P=B5 S=F0
*
```

(Notez que ces contenus ne sont que des exemples.)

Pour modifier le registre d'index Y et lui attribuer la valeur 8A sans modifier les autres, frappez les valeurs existant dans l'accumulateur et le registre d'index Y, puis la nouvelle valeur de Y :

```
*: CD B1 C3 8A
```

Pour modifier tout registre autre que l'accumulateur (A), il faut ré-entrer les contenus des registres qui le précèdent, jusqu'au registre à modifier.

Vous *devez* presser CTRL-E (puis RETURN) pour examiner les registres, sinon le moniteur supposera que vous vouliez modifier le contenu d'une position mémoire. L'examen des registres indique au moniteur qu'il doit cesser d'appeler le pointeur d'adresses et qu'à la place, il doit agir sur les registres directement.

Pour illustrer un autre changement, supposez que vous vouliez modifier le pointeur de pile (son contenu suit le S) et lui attribuer la valeur 4B. Tout d'abord, examinez les registres :

```
*  <------ Pressez CTRL-E, puis RETURN

 A=FF X=CD Y=81 P=8A S=F0
*
```

Maintenant, entrez les valeurs courantes, dans l'ordre, pour tous les registres, puis la nouvelle valeur pour le dernier :

```
*:FF CD 81 8A 4B
```

Vérifiez que la modification a été bien faite avec un nouveau CTRL-E pour ré-examiner les registres.

SAUVEGARDE ET RECHARGE DE LA MÉMOIRE AVEC LES PÉRIPHÉRIQUES D'APPLE II

Le moniteur autorise l'emploi d'une unité à cassettes pour sauvegarder les contenus d'un bloc de mémoire sur bande magnétique. Vous appliquerez ce processus si vous créez des formes graphiques en haute résolution (voir le chapitre 6), ou si vous rédigez des programmes en langage assembleur que vous voudrez conserver. Avec le système d'exploitation disques d'Apple II (le DOS), vous pourrez sauvegarder les contenus mémoires sur disque plus vite et avec une fiabilité supérieure. Pour sauvegarder la mémoire sur disque, vous devrez quitter temporairement le moniteur, et utiliser le DOS via le Basic, aussi bien en sauvegarde qu'en recharge.

Sauvegarde de la mémoire sur cassette

Pour sauvegarder le contenu de la mémoire sur cassette, utilisez la commande d'écriture mémoire du moniteur. Vous aurez à fournir au moniteur les adresses de début et de fin du bloc de mémoire à sauvegarder. La commande d'écriture mémoire sur bande commence avec l'adresse de départ, immédiatement suivie par un point, puis par l'adresse finale, et enfin la lettre W.

Par exemple, la commande :

```
*2200.2FFFW
```

ordonne au moniteur de transcrire le contenu de la mémoire à partir de l'adresse hexadécimale 2200 et jusqu'à l'adresse 2FFF sur la bande magnétique.

La comande d'écriture mémoire ne liste pas les données envoyées au port de sortie vers la cassette ; il ne vérifie pas non plus (et ne saurait le faire) si une bande est réellement présente dans l'unité à cassettes connectée au port de sortie. Assurez-vous que la bande que vous utilisez est de bonne qualité, n'est pas bruyante et ne présente pas de défauts de surface, etc., et ce dans la mesure du possible.

Lorsque vous introduisez une commande d'écriture pour sauvegarder la mémoire, ne pressez pas RETURN tant que vous n'avez pas placé l'unité à cassettes en mode enregistrement (RECORD) et que vous ne voyez pas la bande défiler. Si la cassette est en début de bande, attendez au moins cinq secondes avant de presser RETURN, et cela afin que l'amorce non magnétique de la bande ait fini de passer.

Lorsque vous pressez RETURN, l'ordinateur attend dix secondes avant d'envoyer les données. L'unité à cassettes efface toutes les informations pré-existant sur la bande (musique, parole, données). L'ordinateur émet un ton de référence pendant ce temps. Le moniteur se servira ensuite de ce ton comme d'un signal de lecture (ce qu'on va voir dans la section suivante).

Lorsque la recopie est terminée, Apple II émet un nouveau « bip » et le caractère d'appel du moniteur réapparaît.

La commande d'écriture mémoire autorise la sauvegarde d'un octet à 64 kilo-octets (65536 octets) sur bande. Le moniteur envoie les données via le port de sortie cassette à environ 210 caractères par seconde (sur la base de 16384 octets en 77,5 secondes après le ton de référence). Après transmission du dernier octet de mémoire, le moniteur émet la somme de test qui servira, lors de la relecture, à vérifier la validité des données.

Relecture des données sur cassette

La commande de lecture mémoire sert à rechercher les données sur cassette et à les recharger en mémoire. Pour cela, entrez l'adresse de départ (à laquelle les données provenant de la cassette devront être rechargées en mémoire), puis un point, après quoi introduisez

l'adresse finale (à laquelle se trouvera le dernier octet des données lues sur la cassette), et enfin la lettre R.
Par exemple, la commande :

```
*2000.20FFR
```

lira les données en cassette et les chargera en mémoire, à partir de l'adresse hexadécimale 2000 et jusqu'à 20FF.
A l'inverse de la commande d'écriture, la commande de lecture oblige le moniteur à attendre que vous ayez pressé le bouton « lecture » (« PLAY ») de l'unité à cassettes. Le moniteur attend l'arrivée du signal de référence puis procède à la lecture des données. Avant d'actionner PLAY sur l'unité à cassettes, positionnez bien la bande à l'endroit où le ton de référence débute. Vous pouvez faire la distinction entre signal de référence et données en les écoutant ; pour cela, ôtez le connecteur du jack écouteur pour entendre ce qui est enregistré sur bande, via le haut-parleur de l'unité à cassettes ; le signal de référence est un ton continu, de hauteur moyenne, alors que les données se traduisent par un bruit de type aléatoire ou statique.
Assurez-vous que le volume de l'unité à cassettes est bien réglé avant d'employer la commande de lecture. La procédure de réglage est expliquée dans le chapitre 2.
La commande de lecture mémoire attend une lecture d'exactement autant de mots que vous en avez sauvegardés. Si vous avez enregistré 1024 octets de la mémoire sur bande et souhaitez ne retrouver que les 256 premiers octets que vous avez enregistrés, le moniteur exécutera ce transfert mais vous recevrez probablement un message d'erreur. Il en ira de même si vous voulez lire davantage que ce qui a été stocké sur bande ; un message d'erreur est alors très probable.

Conditions d'erreurs en commande de lecture mémoire

Le moniteur lit la cassette pendant au moins 3,5 secondes avant d'attendre des données sur son port d'entrée. Ceci lui permet de se verrouiller sur la fréquence du ton de référence. Si la bande contient moins de 3,5 s de ce ton, le moniteur perdra le début de la transmission de la cassette, et la somme de test révèlera une erreur. De plus, vous ne saurez pas à quel endroit le moniteur a commencé à lire la bande car le moniteur transfère toujours les données de la cassette vers la mémoire, qu'elles soient valides ou non. Dans ce cas, le moniteur émettra un message d'erreur (Apple II fera entendre un « bip », suivi par le message ERR et le caractère d'appel du moniteur). Pour corriger l'erreur, rembobinez la bande jusqu'au début du ton de référence. Pressez PLAY, le connecteur « écouteur » étant retiré du jack de l'unité à cassettes, et chronométrez le temps. S'il s'est écoulé moins de 3,5 secondes avant que les données arrivent, il vous faudra recopier la mémoire sur bande à nouveau. Le plus probable, c'est que vous aurez omis de sauter l'amorce non magnétique située au début de la bande, avant d'enregistrer.
Une lecture de moins, ou de davantage de données que ce qui avait été sauvegardé sur bande, provoquera probablement l'affichage d'un message d'erreur. Le dernier octet envoyé à la cassette en réponse à une commande d'écriture mémoire est l'octet de la somme de test. Sa valeur dépend de la quantité de données et de la qualité des données enregistrées. Lorsqu'une lecture n'est pas égale en longueur à l'écriture d'origine, le moniteur ne peut prévoir quelle donnée d'entrée sera l'octet de la somme de test. Le moniteur suppose que le dernier octet lu sur la cassette (et qui dépend de l'adresse finale accompagnant l'ordre de lecture mémoire) sera suivi par l'octet de la somme de test. Le moniteur exécute ses propres calculs de somme de test sur les données qu'il reçoit pendant l'opération de lecture, et il la compare à la somme de test lue sur la cassette.

Si ces deux octets ne sont pas identiques, il affiche un message d'erreur. Il est cependant possible que ces deux octets soient identiques, par coïncidence. En règle générale, vous devriez lire autant de mémoires que vous en avez enregistrées. Ainsi, le moniteur exécutera un test d'erreur significatif. Plus avant dans ce chapitre, nous verrons comment vérifier une opération de lecture mémoire avec le moniteur.

Sauvegarde de la mémoire sur disque

Avec le système d'exploitation disques d'Apple II, la sauvegarde de la mémoire est plus rapide et bien plus fiable qu'avec des cassettes. Les règles pour sauvegarder la mémoire sur disquette sont sensiblement différentes car on emploie le Basic et non plus le moniteur pour cette fonction. C'est aussi une commande machine, supérieure aux commandes des cassettes incluses dans le moniteur, et vous devriez y recourir chaque fois que possible. Avant de lire cette section, vous devriez vous être familiarisé avec le DOS (voir le chapitre 5). Le DOS doit se trouver en mémoire avant toute autre chose (voir le chapitre 2).
Si vous êtes dans le moniteur, vous devez le quitter provisoirement et entrer dans le Basic sous DOS avec les commandes CTRL-B ou CTRL-C si vous possédez le moniteur autostart, ou 3DOG s'il s'agit du moniteur standard.
Voici un exemple de commande DOS pour une sauvegarde sur disque :

```
BSAVE SHPTABLE, A$3000, L256, S6, D1, V201
```

Cette commande va créer un fichier sur disque appelé SHPTABLE. Le paramètre suivant, A$3000, opérera la sauvegarde à partir de l'adresse 3000 en hexadécimal. Le A veut dire adresse. La troisième information, L256, spécifie la longueur (L) de l'écriture mémoire et dans ce cas, le nombre décimal 256. La longueur maximale est 32767 en décimal ($7FFF en hexadécimal). BSAVE accepte des valeurs décimales ou hexadécimales pour les adresses (A) et les longueurs (L). En employant l'hexadécimal, n'oubliez pas de mettre avant la constante un symbole « dollars ».
Les trois derniers paramètres de la commande ci-dessus (S6, D1, V201) sont optionnels. Ils spécifient le disque à utiliser. N'employez le paramètre S (pour « slot », connecteur) que si vous disposez de plus d'une carte « contrôleur de disques » et que si l'unité à disques à laquelle vous vous adressez n'est pas la dernière sélectionnée (généralement, le connecteur 6 est le connecteur standard pour l'unité à disquettes). Le paramètre D (« drive », unité à disquettes) est utile mais non nécessaire ; ne spécifiez l'unité que si ce n'est pas la dernière adressée. Le paramètre optionnel V (volume) est le numéro de volume rangé dans le répertoire du disque sur lequel vous sauvegardez la mémoire. Si le numéro de volume accompagnant BSAVE diffère du numéro de volume du disque sur lequel va se faire la sauvegarde, un message d'erreur apparaît.

Recharge de la mémoire à partir du disque

Comme avec BSAVE, le DOS vous permet de relire les données sur disque et de les recharger en mémoire, avec la commande BLOAD.
Voici un exemple de commande BLOAD :

```
BLOAD SHPTABLE, A$3000
```

Ici, on charge le fichier appelé SHPTABLE et se trouvant sur la disquette en service directement en mémoire, à partir de l'adresse 3000 en hexadécimal. Cette commande accepte aussi une adresse décimale. Le paramètre A (adresse de départ) n'est pas nécessaire si le contenu du fichier commence à la même position que lors de sa sauvegarde. Vous n'avez

à spécifier l'adresse que si elle diffère de l'adresse de départ donnée avec BSAVE. Le paramètre longueur (L) est optionnel et non nécessaire. Le DOS teste la longueur du fichier sur disquette et termine automatiquement la commande de lecture mémoire. Les paramètres optionnels de spécification du disque employés avec BSAVE (S, D et V) peuvent aussi intervenir avec BLOAD afin de préciser le connecteur, l'unité à disques et le volume.
Attention à ne pas utiliser cette commande dans des zones mémoires déjà occupées par le DOS, l'interpréteur Applesoft, des pages de textes ou graphiques, ou par des variables en Basic. Il est possible de surcharger de telles zones, ce qui vous ferait perdre des données que vous auriez souhaité conserver.

DÉPLACEMENT ET COMPARAISON DE BLOCS DE MÉMOIRE

Si vous lisez ou écrivez en mémoire avec une disquette ou une cassette, le moniteur vous propose deux fonctions utiles. La commande de déplacement (« move ») recopie un bloc de mémoire dans une zone différente d'adresses. La fonction de comparaison compare deux blocs en mémoire et annonce les différences entre ces deux blocs. Employées avec les commandes de lecture et d'écriture mémoire du moniteur, ces fonctions vous procurent une garantie accrue que les fichiers transcrits sont corrects.

La commande de déplacement mémoire

Pour déplacer des données en mémoire d'une adresse à une autre, vous devez fournir l'adresse de début destinataire (à laquelle vous voulez envoyer les données), l'adresse de début expéditeur (source), et l'adresse finale source (la dernière adresse à transférer). Le format de cette commande est : l'adresse de début destinataire, suivie par le symbole « plus petit que » (<), suivie par l'adresse de début source, puis un point, et enfin l'adresse finale source puis la lettre M (pour « move », déplacement). Comme avec les autres commandes du moniteur, toutes les adresses sont en hexadécimal.
Par exemple, la commande :

```
*1200<2000.2100M
```

déplace à l'adresse 1200 en hexadécimal (adresse de début, destinataire) les données provenant de 2000 (début, source) et se terminant en 2100. Le moniteur recopie les données en mémoire, à partir de l'adresse 2000 et jusqu'à l'adresse 2100 dans les positions 1200 à 1300 dans cet exemple. Puisque les adresses sont en hexadécimal, la longueur du bloc est de 257 octets, en décimal, ou encore 101 octets en hexadécimal. Le contenu d'origine des adresses 2000 à 2100 reste inchangé.
Lorsque vous spécifiez des adresses dans une commande de déplacement mémoire, l'adresse de début source devrait être inférieure ou égale à l'adresse finale source. Si l'adresse finale source est inférieure à l'adresse de début source, le moniteur ne transmettra qu'un seul octet, de l'adresse début source à l'adresse de début destinataire, puis terminera là-dessus l'opération de transfert.

Remplissage de la mémoire

La commande de déplacement mémoire *remplit* aussi la mémoire. Le remplissage traduit ce processus de déplacement d'un ou plusieurs octets répétitivement à des adresses consécutives. Supposons que vous vouliez ranger des zéros dans un bloc de mémoire, à partir

de l'adresse 1D00 et jusqu'en 1DFF. En exploitant judicieusement les commandes de modification et de déplacement mémoire, vous pourrez charger un jeu prédéfini de valeurs dans un bloc mémoire.
Pour commencer, placez des zéros dans le premier octet de la mémoire :

```
*1D00:00
```

C'est la première étape d'une procédure de remplissage de la mémoire. La seconde étape fait appel à la commande de déplacement afin de recopier le contenu d'un (ou plusieurs) octets dans un bloc adjacent de la mémoire.
Spécifiez une adresse de début, destinataire, supérieure de un à l'adresse du dernier octet source. Dans cet exemple, ce sera donc 1D01. Etablissez l'adresse de départ source au début de la liste de données (donc ici, 1D00) et posez l'adresse finale source, le dernier octet que vous voulez remplir (1DFF) moins la longueur de la liste avec laquelle vous voulez remplir la mémoire (1DFE).
Nous poursuivons cet exemple, donc, avec la commande :

```
*1D01<1D00.1DFEM
```

Cette commande remplit les positions 1D01 à 1DFF de zéros (ou plus précisément, avec le contenu de la position 1D00). Cette procédure ne fonctionne que pour autant que la liste d'origine existe au début du bloc à remplir. Voici ce qui se passe alors. Puisque l'adresse de départ destinataire vient un octet après l'adresse source, le moniteur déplace la donnée de 1D00 à 1D01 d'abord, chargeant 00 à cette dernière adresse. Lorsque le second octet est déplacé, le moniteur prélève le contenu de 1D01 et le place en 1D02. Ce processus se poursuit jusqu'à ce que le contenu de 1DFE (établi à zéro lors du transfert précédent) passe dans 1DFF. En examinant ces positions, vous vérifierez que les adresses 1D00 à 1DFF ont été remplies avec des zéros.
Vous pouvez remplir la mémoire avec des structures de données de plus d'un octet de longueur. Par exemple, pour déplacer 00 5E 7F FF dans les positions 1D00 à 1DFF, vous devrez placer ces quatre octets en mémoire à partir de 1D00 :

```
*1D00:00 5E 7F FF
```

Cette structure est désormais en place. Pour remplir la mémoire jusqu'à 1DFF, entrez la commande :

```
*1D04<1D00.1DFBM
```

Remarquez que l'adresse de début, destinataire, est de un octet supérieure à l'adresse finale source où a été logée la liste des quatre octets ; l'adresse de début source pointe le début de la zone mémoire et l'adresse source pointe la dernière adresse à remplir, moins la longueur des données :

```
  1DFF
- 04
------
  1DFB
```

(arithmétique hexadécimale)

A nouveau, si vous expérimentez cet exemple, examinez la mémoire de 1D00 à 1DFF pour vérifier que la duplication s'est bien faite dans tout le bloc mémoire.

La commande de vérification mémoire

Cette commande moniteur compare deux blocs de mémoire, relevant les différences qui pourraient se manifester entre eux. Vous pouvez employer cette commande en conjonction avec les commandes de lecture et écriture mémoire supportées par le moniteur et le DOS. Si vous sauvegardez la mémoire sur un périphérique et désirez vous assurer que l'enregistrement s'est fait correctement, vous pourrez utiliser cette commande de comparaison.

Son format est à peu près le même que celui d'une commande de déplacement. Entrez l'adresse de début, destinataire (à laquelle commence la comparaison), immédiatement suivie par le symbole « plus petit que » (<), puis l'adresse de début, source (à laquelle commence le bloc référence de la comparaison), un point, puis l'adresse finale source (où se trouve le dernier octet de la comparaison). Suit enfin le lettre V, pour vérification. En voici un exemple :

```
*32D0<0.CV
```

Cette instruction ordonne au moniteur de comparer les données commençant à l'adresse 32D0 à celles se trouvant en 0, et de continuer jusqu'à ce que l'adresse 32DC soit comparée à 000C. Notez que les zéros de tête ne sont pas nécessaires dans les adresses de commandes moniteur.

Si le moniteur rencontre une paire d'octets non identiques (source et destinataire), il affiche l'adresse source avec la valeur qu'elle contient, ainsi que la valeur trouvée dans l'adresse destinataire correspondante.

Par exemple, si vous avez déplacé la mémoire de 0000 à 000C vers les adresses 32D0 à 32DC :

```
*32D0<0.CM
```

puis affiché les blocs source et destinataire :

```
*0.C

0000- 4C 3C D4 4C 3A DB 8C 8C
0008- FF FF 4C 99 E1
*32D0.32DC

32D0- 4C 3C D4 4C 3A DB 8C 8C
32D8- FF FF 4C 99 E1
*
```

vous pourriez vérifier le transfert *de visu*. Si vous modifiez le contenu de 32D8, en posant 5A au lieu de FF :

```
*32D8:5A
```

puis introduisez la commande de vérification :

```
*32D0<0.CV
```

le moniteur comparera le bloc source au bloc destinataire, octet par octet, jusqu'à ce qu'il arrive aux adresses 0008 et 32D8, dont les contenus sont comparés. Puisque vous venez de modifier l'adresse 32D8, le moniteur détectera un désaccord :


```
0008-FF (5A)
*
```

ce qui signifie que la valeur, à l'adresse 0008 dans le bloc source, n'est pas la même que celle de la valeur à l'adresse correspondante 32D8 du bloc destinataire. Le moniteur affichera d'abord la valeur trouvée à l'adresse source (0008, de contenu FF), puis et entre parenthèses, la valeur trouvée à l'adresse destinataire (5A).

L'adresse destinataire n'apparaît pas ; le moniteur suppose que vous saurez ajouter des nombres en base 16 pour trouver l'adresse de non-correspondance. Une façon d'éviter ce calcul d'adresse consiste à inverser les blocs source et destinataire :

```
*0<32D0.32DCV
```

ce qui donnera :

```
32D8-5A (FF)
*
```

Ce message indique que l'adresse source 32D8 contient 5A en hexadécimal, alors que l'adresse correspondante (0008 : voir ci-dessus) contient FF. Cette méthode économise un calcul, mais en crée un autre ; vous devez calculer la nouvelle adresse finale source (32DC).

Vérification des mémoires sauvegardées sur les périphériques d'Apple II

La commande de vérification est particulièrement utile lorsque vous sauvegardez les contenus de la mémoire sur cassette ou disquette. En sauvegardant une partie de la mémoire, puis en la rechargeant en un emplacement différent, vous pouvez vérifier que la sauvegarde a été correcte et sans faute. L'exemple suivant montre comment exécuter cette vérification pour des programmes en langage assembleur, des fichiers de formes et d'autres informations rangées dans des périphériques d'Apple II.

Si vous employez une unité à cassettes pour sauvegarder la mémoire, la première étape consistera à entrer une commande d'écriture mémoire comme suit :

```
*2000.20FFW
```

Cette commande recopie les données en mémoire, à partir de l'adresse 2000 et jusqu'à 20FF, sur la bande magnétique. N'oubliez pas de lancer l'unité à cassettes en mode enregistrement *avant* de presser RETURN. Lorsqu'Appel II émet un « bip » vous indiquant que l'opération est terminée, arrêtez la cassette et rembobinez-la jusqu'au début du son de référence. Si la mémoire, de 2100 à 21FF, est disponible, utilisez la commande de lecture pour recharger les données de la cassette dans ces adresses :

```
*2100.21FFR
```

N'oubliez pas de démarrer l'unité à cassettes en mode lecture. Lorsqu'Apple II fait entendre son « bip », cette opération est terminée. Dès lors, vous pouvez vérifier la mémoire originale en la comparant à ce qui a été sauvegardé sur cassette :

```
*2000<2100.21FFV
```

La commande de vérification compare la mémoire, de 2000 à 20FF, à ce qui a été enregis-

tré sur cassette puis rechargé à l'adresse 2100. Si aucune inégalité ne se manifeste, vous êtes certain que l'opération d'écriture mémoire était réussie.
Pour vérifier la mémoire sauvegardée sur disque, la même procédure générale s'applique. Avant d'essayer de sauvegarder ou recharger la mémoire sur disque, n'oubliez pas d'appeler le DOS. Il vous faudra sauvegarder le bloc mémoire sur disque avec un BSAVE :

```
DONNEESMEM, A$2000,L$FF
```

Une fois la sauvegarde effectuée, sous le nom de fichier DONNEESMEM, relisez les données avec BLOAD :

```
BLOAD DONNEES MEM, A$2100
```

Notez que le paramètre adresse de cet ordre est de 256 octets (décimal) supérieur à l'adresse mémoire du bloc original sauvegardé. Lorsque DONNEESMEM a été rechargé, la commande de vérification va comparer les deux blocs :

```
CALL -151

*2000<2100.21FFV
```

S'il y a accord parfait, vous êtes certain que la sauvegarde a été réussie ; sinon, le moniteur affichera les différences entre les blocs afin que vous puissiez les corriger.

LA COMMANDE GO

Le moniteur dispose d'une commande transférant le contrôle d'Apple II à un programme, à une adresse que vous spécifiez. Au début de ce chapitre, vous avez vu comment quitter le moniteur et revenir à l'Applesoft sur disque. La commande :

```
*3D0G
```

ordonne au moniteur de sauter à l'adresse 3D0 et exécuter l'instruction en langage machine qui s'y trouve. La lettre G, à la fin de cette ligne, signifie GO (soit, en français quelque chose comme aller à, ou en marche...). Si vous entrez la commande ci-dessus lorsque le DOS est en mémoire, l'adresse 3D0 contiendra la première partie d'une instruction en langage machine qui est :

```
JMP  $9DB9
```

Aussi, lorsque le moniteur transfère les commandes à 3D0, l'ordinateur se branche-t-il à l'adresse 9DB9, où commence le DOS.
Le format général pour la commande GO est l'adresse à laquelle transférer le contrôle, suivie par la lettre G. L'adresse est optionnelle ; si aucune adresse n'est introduite, le moniteur se sert de son pointeur d'adresses, supposé pointer l'adresse de saut.

UTILISATION D'UNE IMPRIMANTE

Si votre Apple II est connecté à une imprimante, via la carte d'interface série ou la carte de communications, vous pourrez vous en servir pour les sorties. Pour dérouter les sorties et les envoyer non à l'écran mais à l'imprimante, entrez le numéro de connecteur de votre

carte d'interface, qui commande l'imprimante, suivi par CTRL-D et RETURN. Dès lors, toutes les sorties vont vers l'imprimante. Pour sélectionner l'écran d'Apple II et en faire à nouveau la sortie, utilisez le numéro de connecteur zéro (0) suivi par la commande CTRL-P.
Pour cette commande, assurez-vous que le connecteur indiqué contient bien une carte d'interface. Sinon, Apple II va se verrouiller. La seule façon de se sortir de là consisterait ensuite à presser RESET.
La commande d'imprimante fonctionne exactement comme la commande PR # (numéro de connecteur) en Basic. Toutes deux positionnent le commutateur à deux octets CSW (« character output switch », soit commutateur de sortie caractère) à l'adresse 54 ($36). Ces deux octets contiennent une adresse pointant le sous-programme de sortie de caractère qui doit entrer en service. En modifiant le connecteur avec CTRL-P, vous changez le contenu des deux octets de CSW.

LA COMMANDE CLAVIER

Cette commande ordonne au moniteur d'accepter des entrées d'un périphérique autre que le clavier de l'Apple II. Tout comme pour l'imprimante, vous spécifiez le numéro de connecteur correspondant. Frappez ensuite CTRL-K, puis return. Pour revenir ensuite au clavier d'Apple II, frappez la commande clavier avec le numéro 0 pour le connecteur.
Cette commande positionne le commutateur KSW (« keyboard input switch » : commutateur d'entrée clavier) à l'adresse 56 ($38) sur une adresse à deux octets, selon le connecteur sélectionné.

ÉTABLISSEMENT DES MODES D'AFFICHAGE

Pour que les sorties moniteur sur l'écran soient en vidéo inversée, entrez la commande de vidéo inversée, abrégée en un I. Les données affichées par Apple II apparaîtront en noir sur fond blanc. Cependant toutes les commandes moniteur que vous entrerez seront toujours affichées en mode normal, blanc sur fond noir.
Pour terminer l'affichage en vidéo inversée, introduisez la commande de vidéo normale, abrégée en N. Aucune de ces deux commandes ne demande de paramètres additionnels, autres que I ou N.

ARITHMÉTIQUE BINAIRE UN OCTET AVEC LE MONITEUR

Le moniteur exécute des additions et des soustractions binaires sur un octet. Le résultat est également sur un octet. Pour faire une addition, entrez un cumulande en hexadécimal, puis le signe plus (+), et enfin un cumulateur en hexadécimal. Si le résultat est supérieur à FF, le moniteur tronque le digit de plus fort poids et n'affiche que les huit bits de faible poids du résultat, comme le montre cet exemple :

```
*7F+8A
=09
```

Pour exécuter une soustraction, entrez le diminuande, le signe moins (−), puis le diminuteur. Comme pour l'addition, ces nombres doivent être en hexadécimal. Si la différence est inférieure à zéro, le moniteur affiche le résultat en complément à un, comme le montre cet exemple :

```
*0A-2D
=DD
```

COMMANDES MONITEUR DÉFINIES PAR L'UTILISATEUR

En entrant CTRL-Y en réponse au caractère d'appel du moniteur, vous invoquez une commande définissable par l'utilisateur. Le moniteur saute automatiquement à l'adresse hexadécimale 3F8 lorsque CTRL-Y est introduite. Il existe ici suffisamment d'espace, en 3F8, pour loger une instruction de saut en langage machine. Si vous disposez d'un programme spécial en langage machine stocké quelque part en mémoire, CTRL-Y pourra initialiser un saut à ce programme via 3F8.

L'exemple ci-dessous montre comment établir CTRL-Y pour redémarrer le disque Applesoft sans avoir à frapper la commande familière 3D0G.

Tout d'abord, il vous faut connaître le format d'une instruction de saut en langage machine. Cette instruction occupe trois octets. Le premier est le code de l'instruction, 4C. Les deux suivantes sont l'adresse du saut, mais inversée. Vous devez spécifier *d'abord* le *dernier* octet de l'adresse.

Voici une commande de chargement en mémoire qui installe une instruction de saut à l'adresse 3D0 :

```
*3F8:4C D0 03
```

Maintenant, essayez CTRL-Y. La commande 3D0G est certainement battue !

Pour un autre exemple, examinons comment vous pourriez employer CTRL-Y pour sauter au mini-assembleur, couvert en détail par la section suivante. Si vous entrez la commande moniteur :

```
*F666G
```

le caractère d'appel du mini-assembleur apparaît :

```
!
```

L'adresse à laquelle commence le mini-assembleur, F666, peut être appelée par une instruction de saut (« JMP », pour « jump », sauter) à l'adresse 3F8 :

```
!3F8:JMP $F666

03F8-    4C 66 F6    JMP    $F666   <-- Le mini-assembleur affiche cette ligne
!
```

Bien que le mini-assembleur ne soit discuté que dans la section suivante, sachez que la ligne ci-dessus établit l'adresse de l'instruction à 3F8, ne hexadécimal, l'opérande ($F666) indiquant qu'il faut sauter au début du programme mini-assembleur. Pour revenir au moniteur, faites :

```
!$FF69G
*
```

Le caractère d'appel du moniteur revient à l'angle gauche inférieur de l'écran. Après quoi, si vous pressez CTRL-Y, c'est le caractère d'appel du mini-assembleur qui réapparaît. Etablir une commande définie par l'utilisateur branchant au mini-assembleur économise des frappes sur le clavier. Pour modifier l'adresse de branchement en 3F8, frappez-en une autre.

LE MINI-ASSEMBLEUR

Si vous possédez un Apple II standard (ou la carte Integer Basic avec un Apple II Plus), vous disposez d'un programme en ROM qui vous évite, si vous programmez en langage assembleur, la torture d'un assemblage manuel. Le mini-assembleur réside en ROM avec l'Integer Basic. Il est appelé mini car le programmeur doit employer des adresses littérales, plutôt que des labels mnémoniques en tant qu'opérandes dans les instructions en langage assembleur. De plus, chaque ligne de code que vous entrez est automatiquement et immédiatement assemblée en langage machine. Le principal problème réside dans le fait que vous ne pouvez insérer ou supprimer des instructions à volonté, comme vous le pourriez avec un assembleur courant ou un éditeur de texte.

Le principal avantage du mini-assembleur est qu'il vous permet l'introduction d'instructions machines directement, tout en conservant la facilité d'emploi des instructions mnémoniques du langage assembleur.

Ce chapitre décrit le mini-assembleur et vous montre comment l'utiliser. Cependant, il ne vous expliquera pas les concepts de la programmation en assembleur. Il ne couvrira pas, non plus, le jeu d'instructions du 6502 employé par Apple II.

Aussi, si cette discussion sur l'assemblage, les opérandes, les mnémoniques, etc., vous égare, n'allez pas plus avant. Etudiez au préalable les techniques de programmation en assembleur et le jeu d'instructions du 6502. Puis, revenez à cette lecture et terminez ce chapitre.

L'ACCÈS AU MINI-ASSEMBLEUR

L'adresse d'entrée dans le mini-assembleur est F666 en hexadécimal. A partir du moniteur, faites :

```
*F666G
```

Le moniteur sautera alors au mini-assembleur. A partir de l'Integer Basic ou de l'Applesoft (versions disque ou cassette), entrez la commande en mode immédiat :

```
CALL -2458
```

Lorsque vous appellerez le mini-assembleur, le haut-parleur d'Apple II émettra un « bip ». Le caractère d'appel du mini-assembleur est un point d'exclamation (!).

Erreurs d'entrées

Le mini-assembleur détecte des erreurs que vous pourriez commettre en entrant des instructions en langage assembleur. Il affiche une erreur en émettant un « bip » et en ré-affichant l'instruction avec accent circonflexe en-dessous du premier caractère incorrect. Le compteur de positions n'est pas incrémenté ; il reste inchangé, de façon que vous puissiez ré-entrer l'instruction correctement.

COMMANDES MONITEUR POUR LE MINI-ASSEMBLEUR

Chaque fois que vous êtes dans le mini-assembleur, vous pouvez exécuter des commandes moniteur. Aussitôt après le caractère d'appel du mini-assembleur (!), entrez le symbole dollars ($), suivi par la commande moniteur. L'exemple suivant montre comment examiner le contenu de la mémoire à partir du mini-assembleur :

```
!$1CFF

1CFF- E6
!
```

Cette possibilité fait économiser du temps, lors des allées et venues entre le mini-assembleur et le moniteur. Vous pouvez introduire des commandes moniteur lorsque vous êtes dans l'assembleur simplement en frappant le caractère dollars, en premier caractère d'entrée. En fait, c'est cette caractéristique qui est utilisée pour quitter le mini-assembleur.

POUR QUITTER LE MINI-ASSEMBLEUR

Pour quitter le mini-assembleur, utilisez une commande moniteur précédée du symbole dollars. Pour revenir au moniteur, branchez à l'adresse FF69 avec la commande $FF69G. $CTRL-B ou $CTRL-C vous ramèneront au Basic, sauf si vous utilisez l'Applesoft sur disque ou cassette. Pour l'Applesoft sur disque, employez $3DOG, et $OG s'il est sur cassette.

FORMATS DES INSTRUCTIONS

Bien que l'objet de cette section ne consiste pas à vous enseigner la programmation en langage assembleur, il existe quelques aspects du mini-assembleur que vous devriez connaître avant de chercher à l'utiliser. Tout d'abord, le mini-assembleur dispose d'un pointeur mémoire du moniteur. Il vous faut le positionner avant d'entrer chaque instruction.
Deuxièmement, le microprocesseur 6502 recourt à divers formats d'instructions. Ces formats dépendent beaucoup du mode d'adressage.
Le microprocesseur 6502 dispose de onze modes d'adressage, mais de six formats d'instructions seulement. Ils sont décrits plus loin.
Le premier, l'adressage absolu, ou direct, ne demande qu'une adresse sur un ou deux octets comme opérande. Par exemple :

```
AND $303A
```

Le mini-assembleur ne requiert pas de symbole dollars ($) avant une adresse hexadécimale ; il suppose que l'adresse est en base 16.
Le second format d'adressage se trouve avec le mode immédiat, par exemple :

```
LDA #$04
```

Remarquez le symbole dièze (#), comme premier caractère de l'opérande. C'est un indicateur explicite qui indique que la valeur 04 doit être chargée dans l'accumulateur. Sans ce symbole, le mini-assembleur interprèterait l'instruction ainsi : « prendre le contenu de la cellule mémoire d'adresse 04 et le charger dans l'accumulateur », ce qui ramène à un adressage absolu.
On remarquera aussi que le terme *immédiat* peut induire en erreur. Ne confondez pas l'adressage immédiat en langage assembleur avec l'exécution immédiate en Basic. Les actions sont totalement différentes. Ces termes sont cependant courants, aussi continuerons-nous à les employer en dépit de leur ambiguïté.
Le troisème mode d'adressage est l'adressage indexé, qui a l'allure suivante :

```
CMP $23,X
```


ou

```
AND $80,Y
```

Ces deux instructions se ressemblent, les registres X ou Y apparaissent comme un second opérande. Fondamentalement, ce format donne naissance à une instruction en langage machine où le contenu du registre X ou du registre Y a été additionné au premier opérande, la somme constituant l'adresse à laquelle l'instruction se réfère.
Puis, on trouve la pré-indexation indirecte :

```
AND ($F0,X)
```

Ici, la somme des contenus de l'adresse ($F0) et du registre d'index X pointent une adresse dans les 256 premiers octets de la mémoire laquelle, à son tour, pointe la donnée qui servira d'opérande à l'instruction.
La post-indexation indirecte est un mode d'adressage se traduisant par :

```
ORA ($22),Y
```

Dans ce format d'instruction, le premier opérande sert de pointeur pour une adresse sur deux octets, ici logée à la position $22. Cette instruction ajoute le contenu du registre d'index Y à l'adresse trouvée au $22 ; la donnée trouvée à cette nouvelle adresse sert dans l'instruction en langage machine. Le mini-assembleur reconnaît l'adressage post-indexé indirect lorsque le premier opérande est entre parenthèses, comme dans l'exemple ci-dessus.
L'adressage indirect peut être illustré ainsi :

```
JMP ($22FE)
```

Ici, l'instruction de saut JMP ne charge pas l'adresse $22FE dans le compteur ordinal. C'est l'adresse trouvée à la position $22FE, sur deux octets, qui ira dans le compteur ordinal. De ce fait, l'opérande, en format indirect, est réellement un pointeur plutôt qu'une adresse littérale.

UTILISATION DU MINI-ASSEMBLEUR

Ainsi que nous l'avons déjà indiqué, le mini-assembleur entretient un compteur de positions qui s'incrémente de la longueur de chaque instruction en assembleur que vous entrez. En d'autres termes, une fois que l'instruction que vous avez entrée a été assemblée en langage machine, le mini-assembleur calcule sa longueur (1, 2 ou 3 octets) et incrémente le compteur de positions qui passe à la ligne suivante.
La première chose à faire pour exploiter le mini-assembleur consiste alors à positionner ce compteur de positions. Faites-le en tant que partie de la première instruction en assembleur, par exemple :

```
!8DB0:LDA #$04
```

Immédiatement après le caractère d'appel de l'assembleur, entrez l'adresse de départ du programme, ici 8DB0, suivie par un double point (:) puis par la première instruction en assembleur.
Vous n'aurez plus à introduire une nouvelle adresse pour le compteur de positions avec l'instruction suivante. Le mini-assembleur la calculera, sauf si vous souhaitez appeler une adresse différente en ré-initialisant le compteur de positions comme ci-dessus.

Une fois le compteur de positions initialisé, entrez vos instructions en langage assembleur, une par ligne. Après la première ligne, introduisez un blanc suivi par l'instruction en assembleur, comme ici :

```
! JSR FB1E
```

Ceci indiquera au mini-assembleur qu'il doit calculer la nouvelle valeur du compteur de positions.

Développement d'un exemple

Dans cette section, on va expliquer le mode de fonctionnement du mini-assembleur pas à pas. Le but de cet exemple consiste à créer un petit programme utilisant les commandes d'entrées de jeux et le haut-parleur de l'Apple II pour créer des sons. La procédure consiste à lire les valeurs d'entrées produites par les manettes de jeux avec le sous-programe moniteur PREAD (à l'adresse FB1E). La valeur de la manette de jeux 0 est l'intervalle entre deux clics du haut-parleur (0= le plus court temps, et FF le plus long), et la valeur de la manette de jeux 1, l'inverse (0 est le temps le plus long, et FF le plus court).
Le programme commence en 1D00 et l'adresse 1CFF sert à ranger la lecture de l'entrée 0.
A chaque nouvelle ligne du programme en assembleur que vous entrez, le mini-assembleur ajoute la valeur du compteur de positions, le code opération et l'opérande en langage machine (appelé aussi *code objet).* Par exemple :

```
1D00-   A2 00       LDX   #$00
```

Le compteur de positions est affiché en début de la ligne assemblée, suivi par un tiret. Après ce champ vient le code opération, suivi par le dernier octet de l'instruction. Avec les instructions à trois octets (celles qui se réfèrent à des adresses sur deux octets), l'octet de faible poids apparaît avant celui de fort poids. Enfin vient l'instruction mnémonique.
Le déroulement de cet exemple est le suivant. Notez que chaque ligne produite par le mini-assembleur apparaît en-dessous de la ligne que vous avez introduite pour la créer :

```
!1D00:LDX #$00          ← La première instruction établit le compteur de positions

1D00-   A2 00           LDX   #$00
! JSR FB1E              ← Tous les nombres sont en hexadécimal (le préfixe $ est inutile)

1D02-   20 1E FB        JSR   $FB1E
! STY 1CFF

1D05-   8C FF 1C        STY   $1CFF
! INX
1D08-   E8              INX
! JSR FB1E

1D09-   20 1E FB        JSR   $FB1E
! LDA C030

1D0C-   AD 30 C0        LDA   $C030
! DEC 1CFF
```



```
1D0F-   CE FF 1C        DEC   $1CFF
! BNE 1D0C ← Le mini-assembleur calcule le saut relatif (F8)

1D12-   D0 F8           BNE   $1D0C
! LDA C030

1D14-   AD 30 C0        LDA   $C030
! INY

1D17-   C8              INY
! BNE 1D14

1D18-   D0 FA           BNE   $1D14
! JMP 1D00

1D1A-   4C 00 1D        JMP   $1D00
!
```

Après avoir introduit ce programme, vérifiez que vous n'avez commis aucune erreur. La meilleure façon de procéder consiste à lister le programme en mémoire, de préférence en format assembleur. Il vous faudra employer le moniteur, comme décrit ci-dessous.
Une autre mesure de sécurité consistera à sauvegarder ce programme sur cassette (avec la commande moniteur W) ou disque (avec l'ordre Basic BSAVE).
Pour lancer le programme, branchez-vous à la position 1D00. Employez la commande du moniteur G, ou faites un CALL 7424 à partir du Basic. Manipulez les commandes des jeux afin de constater comment elles affectent le haut-parleur. Pour terminer, pressez RESET.

DÉSASSEMBLAGE

Le moniteur contient une commande vous permettant de lister des instructions en assembleur même si votre Apple II ne dispose pas du mini-assembleur en ROM. La commande L, pour « lister », désassemble 20 instructions en langage machine et les rétablit en langage assembleur puis les affiche sur l'écran ou tout autre périphérique que vous avez choisi. La commande L fait appel au compteur de positions, servant de pointeur vers l'instruction suivante à désassembler. Aussi, si vous entrez L juste après le programme suivant, le désassemblage commencera à l'adresse 1D1D et ne listera pas le programme ci-dessus.
Avant d'employer la commande listage, une bonne habitude veut qu'on positionne le compteur de positions. Voici le listage désassemblé du programme ci-dessus :

```
!$1D00L

1D00-   A2 00       LDX   #$00
1D02-   20 1E FB    JSR   $FB1E
1D05-   8C FF 1C    STY   $1CFF
1D08-   E8          INX
1D09-   20 1E FB    JSR   $FB1E
1D0C-   AD 30 C0    LDA   $C030
1D0F-   CE FF 1C    DEC   $1CFF
1D12-   D0 F8       BNE   $1D0C
1D14-   AD 30 C0    LDA   $C030
1D17-   C8          INY
1D18-   D0 FA       BNE   $1D14
```

```
1D1A-   4C 00 1D    JMP   $1D00
1D1D-   9F          ???
1D1E-   4E A5 12    LSR   $12A5
1D21-   A4 96       LDY   $96
1D23-   A3          ???
1D24-   D0 A4       BNE   $1CCA
1D26-   EF          ???
1D27-   A4 62       LDY   $62
1D29-   A2 70       LDX   #$70
```

Dans ce cas, les huit dernières instructions désassemblées sont immatérielles, car le programme se termine à l'adresse 1D1A.
Notez que la commande L est une commodité du moniteur indépendante du mini-assembleur (bien que le symbole $ préfixe cette commande). En entrant L ($L dans le mini-assembleur) et pressant RETURN sans positionner le compteur de positions, vous engagerez le moniteur à désassembler les 20 instructions qu'il trouvera juste à la suite de celles qu'il vient de lister.

TEST ET DÉBUGAGE DES PROGRAMMES

En plus du mini-assembleur, le moniteur de l'Apple II standard offre des possibilités de débugage (mise au point) qui vous aideront lorsque vous programmerez en assembleur. Les programmes en langage bas niveau sont certainement parmi les plus difficiles à déboguer et à tester. Au lieu d'afficher simplement le contenu de variables, vous devez inspecter des positions mémoires. des registres et le programme lui-même. Deux commandes du moniteur interviennent alors, STEP et TRACE.
Ces commandes STEP et TRACE ne sont pas disponibles sur les versions d'Apple II disposant du moniteur autostart.

La commande STEP

Alors qu'il est assez facile d'essayer de tester un programme en assembleur en vérifiant le bon déroulement de ses opérations, cette méthode se révèle souvent peu efficace lorsqu'il faut isoler des erreurs. Si le programme est suffisamment court, on peut l'exécuter en pas à pas, afin de tester le résultat de chaque instruction en langage machine, après qu'Apple II l'ait exécutée. C'est ce que permet la commande STEP.
Lorsque vous exécutez la commande STEP (« step » = *pas*), le moniteur désassemble et affiche l'instruction pointée par le compteur de positions, l'exécute, et affiche les contenus des registres du microprocesseur, puis retourne le contrôle d'Apple II au moniteur.
Le format de la commande STEP comporte un paramètre adresse optionnel (pour positionner le compteur de positions), suivi par la lettre S.
La commande STEP ci-dessous exécute la première instruction du programme expérimental sonore donné ci-dessus. Le contenu affiché du registre X est 0. Les autres trois registres dont le contenu est affiché sont inchangés :

```
*1D00S

1D00-   A2 00       LDX   #$00
 A=FF X=00 Y=8C P=32 S=F8
*
```

Vous pouvez alterner les commandes et passer de STEP à d'autres commandes moniteur (pour examiner la mémoire, par exemple). Pour le vérifier, avancez en pas à pas dans ce programme sonore jusqu'à l'instruction STY à la position mémoire 1D05. Vous devrez

entrer la commande S neuf fois, car l'instruction JSR de la position 1D02 appelle un sous-programme en FB1E que vous devrez parcourir avant de revenir en 1D05. Une fois là, examinez la mémoire avec la commande moniteur pour examiner la position 1CFF. L'instruction en 1D05 range la lecture de la manette 0 à l'adresse 1CFF :

```
*S

1D05-   8C FF 1C    STY   $1CFF
 A=00 X=00 Y=00 P=32 S=F8
*1CFF

1CFF- 00
*
```

Avec la commande d'examen mémoire, vous constatez que le contenu du registre Y (3F) est maintenant rangé à l'adresse 1CFF. Vous pourrez faire appel à la plupart des commandes moniteur en faisant du pas à pas.

La commande TRACE

Un programme peut, parfois, se révéler trop long pour le vérifier en pas à pas. Vous préférerez alors suivre l'exécution de chaque pas mais n'intervenir que lorsque c'est nécessaire. La commande TRACE du moniteur vous servira à cela. Sa sortie est semblable à celle de STEP, mais elle vous dispense d'entrer un ordre STEP pour chaque instruction qu'Apple II doit exécuter. Pour stopper la commande TRACE, vous pouvez presser RESET ou inclure une instruction BRK en langage machine dans le programme ; lorsque le moniteur la rencontrera, il retournera les commandes à l'ordinateur (via le moniteur).
Le format de la commande TRACE comporte une adresse optionnelle suivie par la lettre T. Voici la première partie du programme sonore en mode TRACE :

```
*1D00T

1D00-   A2 00       LDX   #$00
 A=FF X=00 Y=8C P=32 S=F6
1D02-   20 1E FB    JSR   $FB1E
 A=FF X=00 Y=8C P=32 S=F6
FB1E-   AD 70 C0    LDA   $C070
 A=00 X=00 Y=8C P=32 S=F4
FB21-   A0 00       LDY   #$00
 A=00 X=00 Y=00 P=32 S=F4
FB23-   EA          NOP
 A=00 X=00 Y=00 P=32 S=F4
FB24-   EA          NOP
 A=00 X=00 Y=00 P=32 S=F4
FB25-   BD 64 C0    LDA   $C064,X
 A=00 X=00 Y=00 P=32 S=F4
FB28-   10 04       BPL   $FB2E
 A=00 X=00 Y=00 P=32 S=F4
FB2E-   60          RTS
 A=00 X=00 Y=00 P=32 S=F4
1D05-   8C FF 1C    STY   $1CFF
 A=00 X=00 Y=00 P=32 S=F6
1D08-   E8          INX
 A=00 X=01 Y=00 P=30 S=F6
```

```
1D09-    20 1E FB     JSR    $FB1E
 A=00 X=01 Y=00 P=30 S=F6
FB1E-    AD 70 C0     LDA    $C070
 A=27 X=01 Y=00 P=30 S=F4
FB21-    A0 00        LDY    #$00
 A=27 X=01 Y=00 P=32 S=F4
FB23-    EA           NOP
 A=27 X=01 Y=00 P=32 S=F4
FB24-    EA           NOP
 A=27 X=01 Y=00 P=32 S=F4
FB25-    BD 64 C0     LDA    $C064,X
 A=27 X=01 Y=00 P=30 S=F4
FB28-    10 04        BPL    $FB2E
 A=27 X=01 Y=00 P=30 S=F4
FB2E-    60           RTS
```

L'inconvénient de TRACE par rapport à STEP réside dans le fait que vous maîtrisez moins l'exécution de chaque pas du programme. Lorsque vous entrez initialement le programme, il vous faut introduire des BRK aux points clés. Ce sont les fonctions logiques de votre programme. Naturellement, vous pouvez remplacer BRK par des NOP (pas d'opération) lorsque vous avez terminé le débugage, mais cette méthode n'est pas réellement satisfaisante.

D'autre part, la commande TRACE s'exécute à une fraction de la vitesse du programme en langage assembleur. Par exemple, substituez une instruction BRK à l'instruction d'adresse 1D1A dans le programme sonore. En entrant la commande 1D00G, ce programme ne demande qu'une fraction de seconde pour s'exécuter. Par contre, en entrant 1D00T, le programme prend de 60 à 70 secondes. Aussi, si le programme à tester est long, agissez avec précaution en mode TRACE si vous souhaitez qu'elle vous soit de quelque utilité.

Retour sur le compteur de positions

Ainsi que nous l'avons mentionné antérieurement dans la section consacrée à la commande STEP, vous pouvez employer des commandes du moniteur en alternance avec STEP et TRACE. Il existe cependant quelques exceptions à cette règle. En effet, les commandes LIST(L), GO, et CTRL-Y (définie par l'utilisateur) modifient toutes le contenu du compteur de positions lorsque vous les introduisez. Il y aura rupture dans le déroulement du programme exécuté en STEP ou TRACE, et vous devrez repositionner le compteur de positions avant de repartir en STEP ou TRACE. L'exemple suivant montre comment le compteur de positions est modifié par l'une de ces commandes :

```
*1D00S
 1D00-    A2 00        LDX    #$00
  A=62 X=00 Y=00 P=32 S=F8
 *S
 1D02-    20 1E FB     JSR    $FB1E
  A=62 X=00 Y=00 P=32 S=F8
 *L                                  ← Ici, commande de listage

 FB1E-    AD 70 C0     LDA    $C070
 FB21-    A0 00        LDY    #$00
 FB23-    EA           NOP
```



```
FB24-    EA           NOP
FB25-    BD 64 C0     LDA    $C064,X
FB28-    10 04        BPL    $FB2E
FB2A-    C8           INY
FB2B-    D0 F8        BNE    $FB25
FB2D-    88           DEY
FB2E-    60           RTS
FB2F-    A9 00        LDA    #$00
FB31-    85 48        STA    $48
FB33-    AD 56 C0     LDA    $C056
FB36-    AD 54 C0     LDA    $C054
FB39-    AD 51 C0     LDA    $C051
FB3C-    A9 00        LDA    #$00
FB3E-    F0 0B        BEQ    $FB4B
FB40-    AD 50 C0     LDA    $C050
FB43-    AD 53 C0     LDA    $C053
FB46-    20 36 F8     JSR    $F836
*S                                  ← Modifier le compteur de positions
                                      afin que le prochain pas soit
                                      $FB49 au lieu de $FB1E
FB49-    A9 14        LDA    #$14
 A=14 X=00 Y=00 P=30 S=F6
*S

FB4B-    85 22        STA    $22
 A=14 X=00 Y=00 P=30 S=F6
*
```

Sous-programmes utiles du moniteur

Parfois, le Basic n'est pas assez puissant pour exécuter toutes les fonctions de votre programme. C'est là, naturellement, l'une des raisons pour lesquelles les programmeurs reviennent aux sous-programmes en assembleur. Cette section montre comment les référencer à partir d'un programme en Basic.

En mêlant des programmes en assembleur à votre programme Basic, vous pouvez créer autant de problèmes que vous souhaitez en résoudre. Où allez-vous loger, en mémoire, le programme en assembleur ? Rappelez-vous que la mémoire d'Apple II contient quatre grandes zones réservées (texte, pages graphiques basse résolution, et deux pages graphiques haute résolution). Le DOS et l'interpréteur Basic peuvent également occuper de la mémoire. Loger en mémoire un programme là où il ne créera pas de problème dépend du volume mémoire dont vous disposez et de la version d'Apple II que vous utilisez.

Incorporation des sous-programmes

Si vous décidez d'employer un sous-programme moniteur dans votre programme Basic, assurez-vous d'abord qu'il n'en existe pas d'équivalent en Basic. Cela vous épargnera la peine de compliquer ce qui peut rester simple. Puis, testez si le sous-programme en assembleur demande qu'on lui passe des paramètres à partir du programme Basic. S'il vous faut charger des valeurs dans les registres du microprocesseur avant d'exécuter ce sous-programme, ou si ses résultats résideront dans ces registres après exécution, vous devrez utiliser des instructions supplémentaires en langage assembleur pour interfacer le Basic. La plupart des sous-programmes du moniteur ne demandent pas de paramètres au Basic ; ceux qui en demandent disposent fréquemment d'une contrepartie en Basic.

Une fois défini le sous-programme à employer, il vous incombera de le documenter de façon que sa signification soit claire. Par exemple, CALL −936 efface l'écran et place le

curseur à l'angle supérieur gauche de l'écran. Une façon de rendre cette instruction d'appel plus explicite consiste à établir une variable au début du programme comme suit :

```
10 EFFACEMENT = -936
```

puis à s'y référer ensuite :

```
1510 CALL EFFACEMENT
```

Dans ce cas, l'instruction CALL est évidente, mais au prix d'une instruction supplémentaire dans le programme. Votre programme devient alors plus facile à lire et à débuguer.

Problèmes à éviter

Si votre Apple II dispose d'un éditeur-assembleur, il est facile de translater des programmes en ré-initialisant le point d'origine et en ré-assemblant. Cependant, si vous avez rédigé un sous-programme en assembleur à l'aide du mini-assembleur, et si ce sous-programme a été prévu pour travailler avec le Basic, vous rencontrerez peut-être des difficultés qui vous obligeront à ré-écrire ce sous-programme pour des Apple II avec divers volumes de mémoire. Ceci se produira si vous employez des positions mémoires occupées par le DOS, les pages graphiques, ou l'Applesoft en disque ou cassette. Essayez alors d'utiliser les sous-programmes du moniteur chaque fois que ce sera possible.
Si vous programmez en Applesoft, employez toujours la fonction USR pour passer des paramètres au sous-programme, ou retour, à la place de l'instruction CALL. Les adresses 9D à A3 stockent les valeurs des paramètres transmis par USR ; vous pourrez vous servir de cette zone pour repasser des paramètres au Basic. Employez l'ordre POKE pour placer une instruction de saut JMP aux positions 10 à 12 (0A à 0C en hexadécimal). Ces positions doivent contenir une instruction JMP au début du sous-programme en langage machine appelé par USR.

INTÉGRATION DU PROGRAMME AU BASIC

En Basic, les ordres LOMEM: et HIMEM: protègent votre programme en assembleur et interdisent toute surcharge par le Basic. L'établissement d'un programme en assembleur a aussi d'autres exigences si vous voulez disposer en mémoire, simultanément, de l'Applesoft en disquette ou cassette ou du DOS. La procédure générale pour employer des sous-programmes en assembleur ou des programmes en Basic avec le DOS est la suivante :

1. Appelez le DOS.
2. Chargez l'interpréteur Basic du disque ou de la cassette, si besoin est.
3. Etablissez LOMEM : ET HIMEM :
4. Chargez le programme en langage assembleur.
5. Chargez le programme Basic du disque ou de la bande (ou frappez-le au clavier).

Le DOS repositionne HIMEM : après que vous l'ayez chargé, du disque vers la mémoire. L'interpréteur Applesoft repositionne LOMEM: après son chargement. Vous repositionnez LOMEM: et HIMEM; pour réserver de la place à vos programmes en assembleur, puis les charger dans cet espace protégé. Les charges ultérieures de programmes Basic et leurs lancements n'affecteront que l'espace mémoire restant entre LOMEM: et HIMEM: Si vous voulez plus de détails sur la façon de réserver de la place à vos programmes en assembleur, examinez la carte de la mémoire donnée dans l'appendice G. Revoyez aussi la discussion sur LOMEM: et HIMEM: dans le chapitre 8.

CHAPITRE 8

PRÉCIS DE BASIC, INSTRUCTIONS ET FONCTIONS

Ce chapitre décrit la syntaxe de toutes les instructions et fonctions Basic d'Apple II. Les instructions seront tout d'abord présentées, par ordre alphabétique ; puis, ce sera le tour des fonctions, également par ordre alphabétique.
Ce chapitre sert de référence pour toutes les instructions et fonctions. Les chapitres 3 à 7 décrivaient les concepts de la programmation. Ils donnaient également des exemples d'instructions et de fonctions intervenant dans le programme.

MODES IMMÉDIAT ET PROGRAMMÉ

La plupart des instructions peuvent être exécutées en mode immédiat ou programmé. Sauf mention contraire, vous pouvez supposer qu'elles peuvent être employées dans ces deux modes. Les exceptions seront indiquées. Quelques instructions n'interviennent que dans l'un de ces modes et non dans l'autre ; d'autres serviront dans les deux modes mais seront plus pratiques avec l'un d'entre eux.
Certains ordres sont des ordres du système d'exploitation (DOS). Ils peuvent intervenir comme on l'a vu en mode immédiat. En mode programmé, cependant, ils devront être inclus dans un ordre PRINT, avec CTRL-D (code ASCII = 4) en premier caractère. Ainsi, les deux ordres suivants sont équivalents :

```
]CATALOG
]PRINT CHR$(4); "CATALOG"
```

Notez qu'au lieu d'employer CHR$(4) comme ci-dessus, vous pouvez frapper des guillemets en y insérant CTRL-D. Sur l'écran, vous aurez l'impression d'avoir pressé les guillemets deux fois. Le caractère CTRL-D est présent, même s'il n'est pas visible.

VERSIONS DU BASIC

Toutes les instructions et fonctions ci-après sont disponibles à la fois en Integer Basic et en Applesoft, sauf mention contraire. Lorsqu'une instruction ou une fonction agit différemment dans ces deux versions du Basic, les différences seront notées.

NOMENCLATURE ET CONVENTIONS DE FORMATS

Il nous faut normaliser la forme générale des instructions et fonctions. Voici les conventions que nous utiliserons (ponctuation, caractères, etc.) :

{}	Les accolades indiquent un choix d'items. L'un des items inclus dans les accolades doit être présent ; ces accolades n'interviennent plus dans l'instruction réelle.
[]	Les crochets indiquent que le paramètre inclus est optionnel ; ces crochets n'interviennent pas dans l'instruction réelle.
...	Les points de suspension indiquent que le précédent item peut être répété ; ces points de suspension n'apparaissent plus dans l'instruction réelle.
numéro de ligne	Un numéro de ligne est impliqué dans chaque instruction en mode programme.
autres signes de ponctuation	Tous les autres symboles de ponctuation — virgule, double point, guillemets et parenthèses — doivent apparaître comme indiqué.
MAJUSCULES	Les mots et lettres en majuscules doivent apparaître exactement comme indiqué.
italique	Les termes génériques sont en italique. Le programmeur fournira le mot exact ou la valeur, selon les termes génériques listés ci-dessous.

Les termes génériques italiques ci-dessous sont utilisés dans les définitions d'instructions et de fonctions. Tous ceux qui ne figureraient pas dans cette liste sont spécifiques de l'instruction dans laquelle ils apparaissent ; ils seront définis dans le texte décrivant l'instruction.

col	Numéro de colonne en graphique basse résolution ; une expression numérique dont la valeur est comprise entre 0 et 39.
colh	Numéro de colonne en graphique haute résolution ; une expression numérique dont la valeur est comprise entre 0 et 279.
const	Toute constante numérique ou chaîne.
D*n*	Numéro d'unité à disques, devant être spécifié sous forme D0 ou D1.
expr	Toute constante, variable, ou expression, chaîne numérique, relationnelle ou booléenne (en Applesoft seulement) ; toute combinaison valide de ces éléments.
expr$	Tout chaîne, constante, variable ou expression.
exprnm	Toute constante numérique, variable ou expression.
nomdefichier	Tout nom de fichier.
ligne	Tout numéro de ligne d'un programme Basic.
ligne'	L'une des lignes d'un programme Basic.
memadr	Expression numérique, variable ou constante qui évalue une adresse mémoire. Les adresses mémoires vont de −65535 à 65535 (en décimal), où −65535 est équivalent à −1, −65534 à −2, etc.
mempos	Toute position mémoire spécifiée par une constante entière, de 0 à 65535 (décimal) ou $0 à $FFFF (hexadécimal). Les constantes hexadécimales sont identifiées par le préfixe dollars ($).

message	Toute chaîne de texte incluse dans des guillemets.
rang	Numéro de rangée en graphique basse résolution ; une expression numérique de valeur comprise entre 0 et 47.
rangh	Numéro de rangée en graphique haute résolution ; une expression numérique de valeur comprise entre 0 et 191.
S*n*	Numéro de connecteur d'entrées-sorties ; doit être S0, S1, S2, S3, S4, S5, S6 ou S7.
var	En Integer Basic, toute variable numérique ou chaîne. En Applesoft, toute variable numérique, entière ou chaîne.
varnm	Tout nom de variable numérique.
var(ind)	En Integer Basic, toute variable numérique indicée. En Applesoft, toute variable indicée entière, numérique ou chaîne.
V*n*	Identification pour numéro de volume d'un disque (entre V0 et V255).

INSTRUCTIONS (par ordre alphabétique)

APPEND

Ouvre un fichier (voir OPEN) et positionne le pointeur de fichier à la fin du fichier.

Format :

APPEND *nomdufichier [,Dn]* [,S*n*] [,V*n*]

Tableau 8.1. — APPEND en langage machine.

LANGAGE MACHINE		EN ASSEMBLEUR 6502	
Décimal	**Hexadécimal**	**Instruction**	**Commentaire**
169 0 76 237 253	A9 0 4C ED FD	LDA $0 JMP $FDED	Le programme moniteur à $FDED extrait le caractère du registre A ($0, dans ce cas) et l'envoie sur dispositif de sortie sélectionné, le disque. Voir appendice D.

Un buffer mémoire de 595 octets est attribué au fichier de texte spécifié. Le fichier doit être séquentiel. La commande WRITE peut ensuite servir à ranger des informations dans le fichier, à partir du premier octet libre.

Ce dernier suit immédiatement le dernier caractère du fichier, sauf s'il se trouve des octets inutilisés au milieu.

Parfois, APPEND ne commencera pas au premier octet libre du fichier (souvent, à la fin du fichier). Au contraire, il commencera au début du fichier (horreur !). Pour être sûr que ça n'arrivera pas, votre programme devra toujours écrire un marqueur de fin de fichier avant de clore le fichier dans lequel il vient d'écrire. Le court sous-programme en langage machine du tableau 8.1 assure cette fonction. Avec POKE, introduisez-le en mémoire, n'importe où, là où vous disposerez de cinq octets libres (les positions 768 à 772 conviennent, sauf si vous ne les occupez pas avec autre chose). Appelez alors ce sous-programme (avec un CALL) tout juste avant de clore le fichier.

Si le fichier n'existe pas sur l'unité à disquettes D*n* du connecteur S*n*, le message d'erreur FILE NOT FOUND est affiché. Si le disque présent dans D*n* via S*n* n'est pas le volume V*n*, l'erreur VOLUME MISMATCH est affichée. V0 s'accordera avec tous disques. Si le fichier est déjà ouvert, APPEND le ferme et le ré-ouvre (voir CLOSE).
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, la dernière référence d'unité à disquettes et de connecteur est prise en compte. V0 est pris si V*n* est absent. De même, *n* peut être absent ; D0, S0 ou V0 seront alors utilisés.
APPEND est une commande du DOS, exigeant un PRINT et CTRL-D en mode programmé.
Il ne peut être employé en mode commande.

AUTO

Etablit la numérotation automatique des lignes en Integer Basic.

Format :

AUTO *ligne* [, *incrément*]

Les numéros de lignes sont affichés automatiquement chaque fois que vous frappez un RETURN, à partir de *lignes*, par incréments définis par *incrément* dont la valeur, par défaut, est 10 (s'il n'est pas spécifié). Frappez CTRL-X pour effacer un numéro automatique de ligne ; la numérotation automatique reprend sauf si MAN est entré sur la ligne suivante (voir MAN).
Peut être employé en mode immédiat.
Non disponible en Applesoft.

BLOAD

Retrouve un fichier binaire sur disque et le recharge dans la section spécifiée de la mémoire.

Format :

BLOAD *nomdufichier* [,A mempos] [,D*n*] [,S*n*] [,V*n*]

Si le paramètre A est absent, le fichier spécifié est chargé en mémoire, en commençant à la position à partir de laquelle il avait été sauvegardé (voir BSAVE). Si le paramètre est présent, le fichier va en mémoire à *mempos*.
BLOAD doit être employé avec soin. Tout ce qui se trouvait en mémoire (votre programme, l'Applesoft, le DOS, etc.) sera surchargé par le fichier.
Si le fichier n'existe pas sur l'unité D*n* du connecteur S*n*, le message d'erreur FILE NOT FOUND est affiché. Si le disque à D*n* et S*n* n'est pas le volume V*n*, il en résulte une erreur VOLUME MISMATCH.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, la dernière référence d'unité à disques ou de connecteur est prise en compte. Si V*n* est absent, V0 est utilisé.
C'est une commande du DOS, qui exige un PRINT et un CTRL-D en mode programme.

BRUN

Retrouve un fichier binaire (qui devrait être un programme en langage machine), le charge dans la section mémoire spécifiée, puis exécute un saut en langage machine (JMP dans le langage assembleur du 6502) à la position mémoire de départ.

Format :

BRUN *nomdufichier* [,A *mempos*] [,D*n*] [,S*n*] [,V*n*]

Si le paramètre A est absent, le fichier spécifié est chargé en mémoire à partir de la position qu'il occupait lorsqu'il a été sauvegardé (voir BSAVE). Si le paramètre A est présent, le fichier est chargé à l'adresse *mempos*.
Un programme en langage machine pourra ne tourner correctement qu'à une seule position mémoire. Vérifiez soigneusement les instructions qui dépendent des adresses avant de charger le programme en un nouvel emplacement. BRUN surcharge tout ce qui se trouvait dans la section mémoire qui va être occupée par le fichier ; ce pourra être désagréable si le DOS ou l'Applesoft sont alors détruits.
Si le fichier n'existe pas sur l'unité D*n* du connecteur S*n*, le message d'erreur FILE NOT FOUND est affiché. Si la disquette en D*n* de S*n* n'est pas le volume V*n*, l'erreur VOLUME MISMATCH est affichée.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, les dernières références d'unité à disques et de connecteur sont prises en compte. Si V*n* est absent, V0 est adopté.
Cette commande du DOS exige un PRINT et un CTRL-D en mode programme.

BSAVE

Crée un fichier sur disque et sauvegarde une section de la mémoire d'Apple II, en binaire, dans ce fichier.

Format :

BSAVE *nomdufichier,* A*mempos,* L*longueur* [,D*n*] [,S*n*] [,Vn]

Le paramètre A spécifie l'adresse de début de la section mémoire à sauvegarder. Le paramètre L spécifie le nombre d'octets à sauvegarder ; *longueur* doit être un entier dans la gamme 0 à 32767 (décimal), et peut être une constante décimale ou hexadécimale. Une constante hexadécimale est identifiée par le préfixe dollars ($).
Si le fichier n'existe pas sur l'unité à disquettes D*n*, du connecteur S*n*, le message d'erreur FILE NOT FOUND est affiché. Si la disquette en D*n* et S*n* n'est pas le volume V*n*, il en résulte une erreur VOLUME MISMATCH.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, les dernières références de D*n* et S*n* sont prises en compte. V0 est pris en compte si V*n* est absent. De même, *n* peut être absent ; dans ce cas, D0, S0 et V0 sont utilisés.
C'est une commande du DOS, qui exige un PRINT et un CTRL-D en mode programme.

CALL

Branche à un sous-programme en langage machine à la position spécifiée.

Format :

CALL *memadr*

CALL peut être employé avec des sous-programmes que vous avez rédigés, aussi bien qu'avec divers sous-programmes intrinsèques listés dans l'appendice D.

CATALOG

Affiche la liste de tous les fichiers se trouvant sur le disque spécifié.

Format :

CATALOG [,D*n*] [,S*n*]

CATALOG affiche d'abord DISK VOLUME, suivi par le numéro de volume du disque. Si la paramètre volume V*n* est inclus dans cette commande, il est ignoré.
La liste des fichiers se trouvant sur le disque est affichée en-dessous du numéro de volume. Pour chaque fichier, CATALOG donne une lettre code précisant le type du fichier, puis le nombre de secteurs requis pour ranger le fichier, et le nom du fichier. Si le fichier est verrouillé, un astérisque apparaît à la gauche du type de fichier (voir LOCK).
Les codes de types de fichiers sont :

I	Programme en Integer Basic
A	Programme en Applesoft
T	Fichier de texte
B	Fichier binaire (en langage machine)

Si la longueur du fichier excède 255 secteurs, la longueur du fichier est affichée modulo 255, c'est-à-dire que si le fichier est de 256, un 0 est affiché ; s'il est d'une longueur 257, un 1 est affiché, etc.
D*n* et S*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont absents, les dernières références d'unité et de connecteur sont prises en compte.
Cette commande du DOS exige un PRINT et un CTRL-D en mode programmé.

CHAIN

Charge et lance un programme en Integer Basic à partir du disque, sans effacer les valeurs des variables ou des tableaux.

Format :

CHAIN *nomdufichier* [,D*n*] [,S*n*] [,V*n*]

La commande CHAIN ne peut être employée qu'en Integer Basic, et ne peut servir qu'à charger un programme en Integer Basic.
Si le fichier n'existe pas sur l'unité D*n* du connecteur S*n*, l'erreur FILE NOT FOUND est affichée. Si, le disque en D*n* de S*n* n'est pas le volume V*n*, il en résulte une erreur VOLUME MISMATCH.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, leur dernière référence sera prise en compte. V0 sera pris en compte si V*n* est absent. Si *n* est absent, D0, S0 et V0 seront utilisés.
Cette commande du DOS exige un PRINT et un CTRL-D en mode programme.

CLEAR

Cette commande Applesoft attribue des 0 à toutes les variables numériques et à tous les éléments des tableaux numériques. Elle attribue aussi une valeur nulle à toutes les variables chaînes et à tous les éléments des tableaux de chaînes.

Format :

CLEAR

L'exécution de cet ordre donne un résultat équivalent à la mise hors tension, puis sous tension d'Apple II, et à la recharge du programme en mémoire. Un programme continuera à tourner après un CLEAR pour autant que les effets de CLEAR ne contrariront pas la logique du programme.
Pour l'Integer Basic, employez CLR.

CLR

Cette commande en Integer Basic assigne un 0 à toutes les variables numériques et à tous les éléments des tableaux, et attribue une valeur nulle aux chaînes.

Format :

CLR

Elle supprime aussi les dimensions des tableaux et chaînes. Vous pourrez cependant affecter les valeurs des tableaux après un ordre CLR, aussi longtemps que les variables ne se seront pas vu attribuer des valeurs nouvelles.
CLR peut être employé en mode immédiat.
En Applesoft, utilisez CLEAR.

CLOSE

Désattribue le buffer utilisé par le fichier spécifique sur disque et, si la dernière opération sur le fichier était un WRITE, sauvegarde tout ce qui était resté dans le buffer en le rangeant dans le fichier.

Format :

CLOSE [*nomdufichier*]

Vous *devez* clore un fichier après avoir utilisé l'ordre WRITE, afin d'éviter de perdre des données. Si le *nomdufichier* est présent, seul le fichier est clos. S'il est absent, tous les fichiers sont alors (sauf un fichier EXCEL).
Parfois, un fichier séquentiel occupera exactement un secteur lorsqu'il sera clos. Dans ces conditions, un APPEND interviendra au début du fichier plutôt qu'à sa fin. Pour éviter cela, appelez le court sous-programme en langage machine donné dans le tableau 8.1 juste avant l'ordre CLOSE. Vous pourrez vous servir d'un POKE n'importe où se trouvent cinq octets libres (et par exemple, les positions 768 à 772 si vous ne les utilisez pas).
Cette commande du DOS exige un PRINT et un CTRL-D en mode programmé.

COLOR=

Etablit une couleur en mode graphique basse résolution.

Format :

COLOR= *exprnm*

Tableau 8.2. — Codes des couleurs en basse résolution.

Code	Couleur	Code	Couleur	Code	Couleur	Code	Couleur
0	Noir	4	Vert foncé	8	Brun	12	Vert
1	Magenta	5	Gris	9	Orange	13	Jaune
2	Bleu foncé	6	Bleu moyen	10	Gris	14	Vert-bleu
3	Pourpre	7	Bleu clair	11	Rose	15	Blanc

Jusqu'à l'ordre suivant COLOR, tous les ordres PLOT, VLIN et HLIN resteront dans la couleur spécifiée. Les codes des couleurs sont donnés dans le tableau 8.2. Le *exprnm* doit avoir une valeur comprise dans la gamme 0 à 255 ; les valeurs réelles sont converties en entiers. Les valeurs supérieures à 15 répètent les couleurs ci-dessus (0, 16, 32, etc. sont des noirs, etc.) . En l'absence de spécification, COLOR = 0.
COLOR n'a pas d'effet en mode graphique haute résolution. Utilisé en mode texte, COLOR est un facteur qui détermine quel caractère est placé sur l'écran par une instruction PLOT. Pour une description détaillée de cette caractéristique, voyez PLOT.

CON

Cette commande en Integer Basic fait reprendre l'exécution du programme à l'instruction suivante après une halte.

Format :

CON

Cette instruction intervient après que l'exécution d'un programme ait été stoppée par un CTRL-C, et parfois un RESET. S'il n'y a pas eu de programme interrompu, CON verrouille simplement le système. On ne peut poursuivre un programme après une interruption par CTRL-C pendant un ordre INPUT.
Si une ligne de programme a été ajoutée ou modifiée, ou si un message d'erreur a été généré après la halte, CON fonctionnera parfois, mais pourra aussi produire un message d'erreur ou verrouiller le système.
CON ne s'emploie qu'en mode immédiat.
Pour Applesoft, voir CONT.

CONT

Cet ordre Applesoft fait reprendre l'exécution du programme, après une halte, à l'instruction suivante.

Format :

CONT

CONT intervient après que l'exécution ait été stoppée par STOP, END ou CTRL-C. Si une instruction INPUT est interrompue par CTRL-C, le programme ne peut être repris. S'il n'y a pas de programme interrompu, ou si une ligne du programme a été changée ou ajoutée, ou si un message d'erreur a été généré depuis que l'exécution a été stoppée, CONT produire le message ?CAN'T CONTINUE ERROR (« Je ne peux pas continuer, erreur »).
Pour l'Integer Basic, voir CON.

DATA

Crée une liste de valeurs qui seront attribuées par des instructions READ en Applesoft.

Format :

DATA *const* [,*const*...]

L'instruction DATA peut apparaître n'importe où dans un programme ; elle spécifie soit des valeurs numériques, soit des chaînes. Les constantes chaînes sont généralement incluses dans des guillemets ; ceux-ci ne sont pas nécessaires, sauf si la chaîne contient des blancs (des espaces), des virgules ou des doubles points. Des guillemets ne peuvent être représentés dans *const ;* ils doivent être spécifiés par une fonction CHR$(34).
Un, ou plusieurs paramètres *const* peuvent être nuls, c'est-à-dire être constitués par des blancs. Une *const* nulle attribue des zéros à la variable numérique, ou une chaîne nulle (" ") à une variable chaîne.
Vous ne recevrez pas de message d'erreur si vous entrez une instruction DATA en mode immédiat, mais ses éléments ne seront pas accessibles à une commande READ.

DEF FN

L'instruction DEF FN sert à définir des fonctions spéciales, définies et utilisées dans des programmes en Applesoft.

Format :

DEF FN*nvar (arg)* = *exprnm*

La variable réelle *nvar* identifie la fonction, qui doit être appelée par son nom FN*nvar*.
La fonction est définie par *exprnm ; arg* est un nom de variable fictive qui peut apparaître dans *exprnm* (et qui apparaît généralement. Son emploi dans une instruction DEF FN n'a pas d'effet sur d'autres variables dotées du même nom dans le programme.
Lorsque FN*nvar* est appelée, la valeur de la variable fictive *arg* est spécifiée par une expression numérique, une variable ou une constante. Les valeurs de toutes les autres variables dans *exprnm* doivent être définies avant que FN*nvar* soit invoquée. Voyez aussi FN dans la section « Fonctions » du présent chapitre.
L'instruction complète DEF FN doit apparaître sur une unique ligne de programme. Cependant, une fonction définie antérieurement peut être introduite dans *exprnm,* et de ce fait, des fonctions définies par l'utilisateur, de toutes complexités, peuvent être développées. Une fonction définie par l'utilisateur ne peut s'invoquer elle-même, directement ou indirectement (en appelant une fonction qui l'appelle à son tour).
Le nom de fonction *nvar* peut être ré-utilisé, et ainsi redéfinie par une autre instruction DEF FN apparaissant ultérieurement dans le programme.
Cette instruction n'est pas disponible en Integer Basic.
Elle est illégale en mode immédiat. Cependant, une fonction définie par l'utilisateur qui aurait été définie avec une instruction DEF FN depuis le dernier NEW, CLR ou LOAD peut être référencée dans une instruction en mode immédiat.

DEL

Elimine les lignes spécifiées du programme.

Format :

DEL $ligne_1$, $ligne_2$

Toutes les lignes du programme supérieures ou égales à $ligne_1$, et inférieures ou égales à $ligne_2$, sont éliminées du programme courant en mémoire. Si $ligne_1$ n'existe pas, la suppression commande à la ligne suivante de numéro supérieur. Si $ligne_2$ n'existe pas, la suppression se termine à la ligne du numéro immédiatement inférieur.
DEL doit être suivi par deux numéros de lignes séparés par une virgule. Aucun numéro ne peut être négatif, le second devrait être supérieur ou égal au premier. Si les deux numéros sont identiques, une ligne (au plus) est supprimée.
DEL ne peut intervenir qu'en mode immédiat en Integer Basic.
Si DEL est employé en mode programme (ce qui n'est possible qu'en Applesoft), les suppressions indiquées prennent place et le programme stoppe. CONT ne le fera pas reprendre dans ce cas.

DELETE

Efface un fichier sur disque.

Format :

DELETE *nomdufichier* [,D*n*] [,S*n*] [,V*n*]

Le fichier portant le nom spécifié est supprimé sur le disque.
Si le fichier n'existe pas sur l'unité à disques D*n*, reliée au connecteur S*n*, le message d'erreur FILE NOT FOUND est affiché. Si le disque en D*n* de S*n* n'est pas le volume V*n*, il en résulte une erreur VOLUME MISMATCH.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, la dernière référence d'unité ou de connecteur est employée. V0 intervient si V*n* est absent. De même, *n* peut être absent ; dans ce cas, D0, S0 et V0 seront utilisés.
C'est une commande du DOS, exigeant un PRINT et un CTRL-D en mode programmé.

DIM

Réserve de l'espace en mémoire pour un tableau ou une chaîne.
En raison des grandes différences en Integer Basic et Applesoft, DIM sera discuté dans ces deux langages.

Format en Integer Basic :

DIM *var (ind)* [,*var (ind)*...]

Seuls, des tableaux numériques monodimensions et des variables simples chaînes peuvent être dimensionnés en Integer Basic.
Lorsqu'un tableau est dimensionné, un espace est réservé en mémoire pour un nombre d'éléments égal à *ind* plus 1. Ils sont numérotés de 0 à *ind*. L'élément 0 d'un tableau est identique à la variable simple de même nom (par exemple, A(0) = A).

L'instruction DIM déclare les longueurs maximales des variables chaînes. Dans ce cas, *ind* est la longueur de la chaîne.
Chaque indice *ind* doit être compris entre 1 et 255 dans une instruction DIM. En outre, la dimension maximale est limitée par la mémoire disponible.
Si vous vous référez à un tableau avec un indice supérieur au plus grand indice déclaré dans une instruction DIM dimensionnant ce tableau, un message d'erreur ***RANGE ERR se produit. Si vous tentez d'utiliser plus de caractères qu'il n'en a été dimensionnés pour une chaîne, un message ***STRING ERR est créé.
DIM n'attribue aux éléments d'un tableau en Integer Basic aucune valeur particulière lorsqu'il est exécuté. De ce fait, vous devez initialiser chaque tableau (par exemple, à zéro) après l'avoir dimensionné. Par ailleurs, les variables chaînes disposent toujours d'une valeur nulle après avoir été dimensionnées pour la première fois.

Format en Applesoft :

DIM *var (ind [,ind...]) [,var (ind [,ind...])...]*

L'instruction DIM identifie un tableau à une ou plusieurs dimensions comme suit :

var (ind_i)	Tableau à une dimension
var (ind_i, ind_j)	Tableau à deux dimensions
var (ind_i, ind_j, ind_k...)	Tableau à trois dimensions

Applesoft supporte trois types de tableaux : entiers, réels et chaînes. Chaque élément d'un tableau est du type spécifié par le nom de variable du tableau. Le nombre de dimensions du tableau est déterminé par le nombre d'indices, dans l'instruction DIM. Lorsqu'un tableau est référencé, chaque indice doit être compris dans la gamme 0 à *ind*, où *ind* est l'indice correspondant à la même variable dans l'instruction DIM.
Le nombre de dimensions d'un tableau est limité par le volume mémoire disponible. Le nombre maximum de dimensions qu'un tableau peut posséder est 88, et ce n'est possible que si la plupart des indices sont 0. Une instruction DIM avec 89 indices ou davantage, ou une instruction qui excède les possibilités de la mémqoire produira le message ?OUT OF MEMORY ERROR.
Si vous tentez d'employer un tableau avec un indice hors de la gamme, ou avec un mauvais numéro d'indice, le message ?BAD SUBSCRIPT ERROR apparaîtra.
Si un tableau est référencé avant qu'une instruction ne l'ait dimensionné, Applesoft attribuera par défaut la valeur 10 à chaque indice. Le tableau sera ensuite traité comme si une instruction DIM avec des indices à 10 pour chaque dimension avait été exécutée.
Un tableau ne peut jamais être dimensionné deux fois, même s'il a été dimensionné par défaut. Si vous tentez de redimensionner un tableau, vous obtiendrez le message ?REDIM'D ARRAY ERROR.

DRAW

Cet ordre Applesoft trace une forme graphique haute résolution sur l'écran.

Format :

DRAW *exprnm* [AT *colh, rangh*]

La forme identifiée par la valeur entière de *exprnm* est tracée dans la couleur déterminée

par la dernière instruction HCOLOR. L'échelle et la rotation de la forme doivent être établies par des commandes SCALE et ROT avant l'exécution de DRAW.
La forme est tracée à partir de la position donnée par les valeurs entières des expressions numériques *colh* et *rangh*. Si vous ne spécifiez pas la position de départ, la forme commence au dernier point tracé par la dernière exécution d'un DRAW, XDRAW et HPLOT.
Le numéro de forme spécifié *(exprnm)* doit être compris entre 0 et le nombre de formes dans la table des formes (255 au maximum), inclusivement.
Evitez l'usage de DRAW s'il n'y a pas de formes en mémoire. Le système pourrait se bloquer, ou alors vous hériteriez de formes aléatoires sur l'écran. Si votre programme s'étend dans la portion mémoire graphique haute résolution, une partie de celle-ci peut être détruite.
N'est pas disponible en Integer Basic.

DSP

Affiche les valeurs modifiées d'une variable spécifiée au fur et à mesure de la progression d'un programme en Integer Basic.

Format :

DSP *var*

La valeur de la variable *var* et le numéro de ligne courant sont affichés chaque fois que la valeur de la variable change. L'affichage peut se mêler aux sorties de votre programme, les rendant tous deux illisibles (ou un seul). RUN annule les instructions DSP. Utilisez CON ou GOTO lorsque vous débuguez avec un DSP en mode immédiat.
Pour faire cesser DSP, utilisez NO DISP.
N'est pas disponible en Applesoft.

END

Provoque l'arrêt d'un programme.

Format :

END

Aucun message n'est affiché. En Integer Basic, END doit être la dernière instruction exécutée, faute de quoi l'avertissement ***NO END ERR (« erreur d'absence de fin ») est affichée. END est totalement optionnel en Applesoft.
Ne peut être employé en mode immédiat en Integer Basic.

EXEC

Exécute un fichier de texte sur disque comme si chaque caractère du texte du fichier était introduit par le clavier.

Format :

EXEC *nomdufichier* [,R*n*] [,D*n*] [,S*n*] [,V*n*]

Le fichier de texte employé avec EXEC consiste en une combinaison de commandes

Basic, DOS et lignes de programme. Lorsqu'EXEC est exécutée, la première ligne du fichier notifiée est lue sur le disque. Si c'est une commande, elle est immédiatement exécutée ; s'il s'agit d'une ligne de programme, elle est ajoutée aux lignes de programme déjà en mémoire, tout comme si vous veniez de l'entrer au clavier.
Un fichier EXEC peut servir à introduire un programme entier, le lister, le lancer, le sauvegarder sur disque, ou à toute autre action que vous pourriez commander au clavier. Vous pouvez même utiliser un fichier EXEC pour créer et exécuter un second fichier EXEC.
Le paramètre R, s'il est présent, spécifie le champ du fichier à exécuter en premier. Lorsqu'il est présent, le paramètre R compte toujours à partir du début du fichier ; le premier champ du fichier est 0, le second est 1. etc. Le nombre accompagnant R doit être une constante entière dans la gamme 0 à 32767. Si R*n* spécifie le premier champ suivant la fin du fichier, rien ne se passe. S'il spécifie deux champs, ou davantage, après la fin du fichier, un message END OF DATA (« fin des données ») survient.
Si un ordre INPUT est exécuté alors qu'un fichier EXEC est ouvet, la réponse est prise dans le fichier EXEC.
Lorsque la dernière ligne de fichier a été exécutée, le fichier EXEC se clot lui-même (voir CLOSE). Lorsqu'une commande EXEC est rencontrée dans un fichier EXEC, ce dernier fichier original est fermé et ses commandes ultérieures ignorées ; le nouveau fichier EXEC est alors ouvert et exécuté normalement.
Si le fichier n'existe pas sur la disquette de l'unité D*n* de S*n*, le message FILE NOT FOUND est affiché. Si la disquette en D*n* de S*n* n'est pas le volume V*n*, une erreur VOLUME MISMATCH en résulte.
D*n*, S*n*, V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, c'est l'unité ou le connecteur référencés en dernier lieu qui sont repris. V0 est pris si V*n* est absent. De plus, *n* peut être absent ; dans ce cas, D0, S0 et V0 sont utilisés.
C'est une commande du DOS, qui exige un PRINT et un CTRL-D en mode programmé.

FLASH

Cette commande Applesoft met en service le mode clignotant vidéo.

Format :

FLASH

Toutes les sorties émanant d'instructions PRINT exécutées apparaissent alternativement en blanc sur fond noir, puis l'inverse. Les messages d'erreur aussi. Cependant, les caractères renvoyés en écho à l'écran par les ordres INPUT ne sont pas affectés ; de même, ne sont pas affectés les caractères affichés antérieurement à FLASH.
FLASH fonctionne en altérant légèrement les codes ASCII standards. Aussi, les caractères envoyés sur le disque en mode flash disposeront-ils de codes incorrects ; à la lecture, on obtiendra des caractères erronés.
N'est pas disponible en Integer Basic.

FN

Est présentée dans la section « Fonctions » de ce chapitre. Voir aussi DEF FN.

FOR

Débute une boucle répétant une séquence d'instructions jusqu'à ce qu'une variable, automatiquement incrémentée, atteigne une valeur donnée.

Format :

FOR *varnm* = *exprnm*$_1$ TO *exprnm*$_2$ [STEP *exprnm*$_3$]

A la première exécution de FOR, *varnm* se voit assigner la valeur de *exprnm*$_1$. Les ordres suivants FOR sont exécutés jusqu'à la rencontre d'une instruction NEXT. *Varnm* est alors incrémentée de *exprnm*$_3$ (ou de 1 si la clause STEP est absente). Après quoi, la nouvelle valeur de *varnm* est comparée à la valeur de *exprnm*$_2$. Le sens de la comparaison dépend du signe de *exprnm*$_3$. Si le signe est positif et si la nouvelle valeur de *varnm* est inférieure ou égale à *exprnm*$_2$, l'exécution de la boucle reprend à l'instruction qui suit le FOR. Le même processus intervient lorsque le signe de *exprnm*$_3$ est négatif et que la nouvelle valeur de *varnm* est plus grande ou égale à *exprnm*$_2$. Par ailleurs, l'exécution se poursuit par l'instruction suivant l'instruction NEXT si *varnm* est plus grande que *exprnm*$_2$ (avec *exprnm*$_3$ positive) ou inférieure à *exprnm*$_2$ (avec *exprnm*$_3$ négative). Parce que la comparaison intervient *après* l'incrémentation de *varnm,* les instructions entre FOR et NEXT sont toujours exécutées au moins une fois.
En Integer Basic *varnm* doit être une variable entière. En Applesoft, *varnm* doit être une variable réelle. Elle ne peut jamais être une variable chaîne.
Les valeurs de début, fin, et incrémentation sont déterminées par *exprnm*$_1$, *exprnm*$_2$ et *exprnm*$_3$ une seule fois, à la première exécution de l'instruction FOR. Si vous modifiez ces valeurs dans le corps de la boucle, il n'en résultera aucun effet sur la boucle. Vous pouvez changer la valeur de *varnm* dans la boucle. Cela vous permettra de terminer la boucle FOR-NEXT avant que la valeur finale soit atteinte ; positionnez *varnm* à la valeur finale *(exprnm*$_2$*),* et à la passe suivante, la boucle se terminera. Ne démarrez pas une boucle en dehors d'un sous-programme pour la terminer dans un sous-programme.
Les boucles FOR-NEXT peuvent être emboîtées. Chaque boucle emboîtée doit disposer d'un nom de variable *varnm* différent. Chaque boucle interne doit être totalement comprise dans la boucle voisine externe ; au mieux, les boucles peuvent s'achever au même point. L'Integer Basic autorise 16 niveaux d'emboîtage FOR-NEXT, Applesoft 10 seulement.
FOR peut être employé en mode immédiat mais seulement en Applesoft. La boucle complète doit être entrée en une seule ligne. Si NEXT est absent, la boucle n'est exécutée qu'une fois.

FP

Place Apple II en Applesoft.

Format :

FP [,D*n*] [,S*n*] [,V*n*]

La source d'Applesoft dépend du type d'Apple II que vous possédez, et des options qui sont installées :

1. Avec l'Apple II Plus, le langage est en mémoire morte (ROM), quelles que soient les options présentes.
2. Si la carte « Applesoft firmware » est installée, c'est elle qui fournit le langage à FP, quelle que soit la position du commutateur sur la carte.
3. Avec le « Apple Language System » installé, c'est de ce système que FP appelle Applesoft.

4. Sur les autres Apple II, FP recherche Applesoft sur le disque spécifié (ou courant). S'il n'y est pas, le message LANGUAGE NOT AVAILABLE (« langage non disponible ») est affiché.

FP efface les programmes en Basic se trouvant alors en mémoire.
Si le fichier n'existe pas sur l'unité à disquette D*n* du connecteur S*n*, le message d'erreur FILE NOT FOUND est affiché. Si le disque correspondant à D*n*, et S*n*, n'est pas le volume V*n*, une erreur de VOLUME MISMATCH se manifeste.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* n'existent pas, les dernières références à une unité ou un connecteur sont reprises. V0 est pris si V*n* est absent. Si *n* est absent, D0, S0 et V0 sont utilisés.
Utilisable uniquement en mode immédiat.

GET

Cette instruction Applesoft accepte un seul caractère du clavier sans l'afficher en écho sur l'écran.

Format :

GET *var*

L'exécution du programme cesse jusqu'à ce qu'une touche soit pressée. Lorsque *var* est une variable chaîne, le caractère entré se voit attribué à cette variable. Si CTRL-@ est entré, une chaîne nulle est attribuée à la variable.
On emploie rarement GET avec une variable numérique pour *var*. Mais dans ce cas, l'introduction de l'un des digits de 0 à 9 attribue cette valeur à la variable. L'entrée de signes plus, moins, d'une virgule, d'un double point, de CTRL-@, d'un espace, de E ou d'un point attribue à la variable la valeur zéro. L'emploi d'un caractère autre que ceux cités déclenche le message ?SYNTAX ERROR et le programme stoppe.
GET ne peut être employé en mode direct.
N'est pas disponible en Integer Basic.

GOSUB

Ordonne au programme de se brancher à la ligne indiquée. Lorsqu'une instruction RETURN est exécutée, le programme revient à l'instruction qui suit immédiatement le GOSUB.

Format général :

GOSUB *ligne*

L'instruction GOSUB appelle un sous-programme. Le point d'entrée dans le sous-programme doit être la ligne dont le numéro est *ligne*. Un point d'entrée logique dans un sous-programme est le début de ce sous-programme, c'est-à-dire la ligne contenant la première instruction, ou les premières instructions à exécuter. Le point d'entrée n'est pas nécessairement la ligne du sous-programme avec le plus petit numéro de ligne.
Après exécution, le sous-programme revient au programme appelant et se branche à la ligne qui suit l'instruction GOSUB. Pour cela, le sous-programme utilise une instruction RETURN.

Une instruction GOSUB peut intervenir n'importe où dans un programme ; en conséquence, un sous-programme peut être appelé de n'importe quel endroit.
Les sous-programmes peuvent être emboîtés, ce qui signifie qu'un sous-programme peut être appelé par un autre sous-programme. Applesoft autorise jusqu'à vingt-cinq niveaux d'emboîtage ; cela signifie qu'on pourra exécuter 24 instructions GOSUB avant de trouver le premier RETURN. En Integer Basic, la limite est de 16 GOSUB.
Normalement, vous devez sortir d'un sous-programme par une instruction RETURN et non pas un GOTO. Mais vous pouvez employer GOTO pour vous brancher ailleurs, hors du sous-programme, si vous exécutez d'abord une instruction POP.
Ne peut être utilisé en mode immédiat en Integer Basic.

Format additionnel en Integer Basic :

GOSUB *exprnm*

L'Integer Basic permet à une expression numérique de remplacer le numéro de ligne. Si *exprnm* n'évalue pas un numéro de ligne constant, le message ***BAD BRANCH ERR (« erreur de mauvais branchement ») est affiché. Cette forme de GOSUB vous permet de simuler l'instruction ON GOSUB, qui n'est pas disponible en Integer Basic.

GOTO

Provoque un branchement inconditionnel à la ligne indiquée.

Format général :

GOTO *ligne*

L'exécution du programme se poursuit immédiatement à l'instruction dont le numéro de ligne est donné. Si le numéro de ligne n'existe pas, le message ?UNDEF'D STATEMENT ERROR est affiché par Applesoft, et le message ***BAD BRANCH ERR par l'Integer Basic.

Format additionnel en Integer Basic :

GOTO *exprnm*

En Integer Basic, une expression numérique peut remplacer le numéro de ligne. Si *exprnm* n'évalue pas un numéro de ligne existant, le message ***BAD BRANCH ERR est affiché. Cette forme du GOTO calculé sert à simuler l'instruction ON GOTO, non disponible en Integer Basic.

GR

Convertit l'écran en mode graphique basse résolution (40 × 40) en laissant quatre lignes de texte en bas de l'écran.

Format :

GR

La portion graphique de l'écran est effacée et le curseur déplacé dans la fenêtre de texte ; COLOR est mis à 0 (noir).

Si GR est exécutée alors que HGR est en service, GR se comporte normalement. Mais si HGR2 est en service, vous serez placé en position d'examen de la page 2, graphique basse résolution et texte. C'est une source de confusion ; l'écran se remplira le plus souvent avec n'importe quoi, et rien de ce que vous frapperez n'apparaîtra. Pour revenir au mode normal, frappez TEXT. Assurez-vous que vous utilisez TEXT dans vos programmes avant de commuter de HGR2 à GR.
Vous pouvez passer à un graphique basse résolution sur le plein écran (40 × 48) grâce à l'instruction POKE −16302,0 juste après l'exécution de GR. Tout ce que vous frapperez ensuite en mode immédiat se traduira par des points en couleur sur les quatre dernières lignes de l'affichage, mais sera exécuté correctement. POKE −16301,0 restaure la fenêtre de texte.

HCOLOR=

Cet ordre Applesoft établit la couleur pour le tracé en mode graphique haute résolution.

Format :

HCOLOR = *exprnm*

Jusqu'à l'instruction suivante HCOLOR, tous les ordres HPLOT et DRAW sont exécutés dans la couleur spécifiée. Le code des couleurs est listé dans le tableau 8.3. La valeur de *exprnm* doit être comprise entre 0 et 7. Des valeurs externes à cette gamme produiront un message d'erreur ?ILLEGAL QUANTITY ERROR. Un tracé en graphique haute résolution exécuté avant le premier ordre HCOLOR se fera dans une couleur indéterminée.
HCOLOR n'affecte pas le graphique en basse résolution. Une instruction HCOLOR exécutée lorsqu'Apple II n'est pas en mode haute résolution n'affecte pas la couleur du tracé suivant en graphique haute résolution.
Non disponible en Integer Basic.

Tableau 8.3. — Corde des couleurs en haute résolution.

Code	Couleur
0	Noir
1	Vert
2	Violet (*)
3	Blanc

Code	Couleur
4	Noir
5	Orange (*)
6	Bleu (*)
7	Blanc

(*) Dépend du réglage de votre téléviseur.

HGR

Cette instruction Applesoft convertit l'écran en mode graphique haute résolution (280 × 160), avec quatre lignes de fenêtre de texte en bas de l'écran.

Format :

HGR

La page 1 de la mémoire d'écran haute résolution est affichée. La mémoire d'écran basse

résolution (texte) n'est pas affectée, mais seules les quatre lignes les plus basses sont visibles. Le curseur ne vient pas dans cette fenêtre. Vous pourrez ne pas la voir tant que vous n'avez pas frappé quelques lignes après l'exécution HGR. La portion graphique de l'écran est remise au noir. HCOLOR n'est pas affectée par cette commande.
Vous pouvez passer au graphique haute résolution sur plein écran (280 × 192) en exécutant l'instruction POKE −16302,0 après un HGR. Les commandes en mode immédiat que vous entrerez ensuite ne seront plus visibles, mais seront exécutées correctement. POKE −16301,0 restaure la fenêtre de texte.
Avec les systèmes Apple II disposant de moins de 32K octets de mémoire, vous ne pouvez utiliser HGR et le DOS simultanément car ils tenteront tous deux d'occuper la même zone de mémoire.
De plus, l'interpréteur Applesoft sur disque ou cassette occupe une partie de la page 1 de la mémoire graphique haute résolution. Aussi ne pouvez-vous employer HGR avec l'Applesoft sur disque ou cassette.
Même avec l'Applesoft en firmware, si votre programme est très long, il risquera de déborder dans la page 1 haute résolution. Vous pouvez vous en protéger avec la commande HIMEM: 8192 qui maintient votre programme hors de la page 1.
N'est pas disponible en Integer Basic.

HGR2

Convertit l'écran mode graphique haute résolution (280 × 192). La page 2 de la mémoire graphique haute résolution est affichée.

Format :

HGR2

La mémoire d'écran basse résolution (texte) n'est pas affichée. Bien que vous ne puissiez voir ce que vous frappez, toute commande que vous entrerez sera exécutée. L'écran est effacé et devient noir. HCOLOR n'est pas affectée.
La page 2 de la mémoire graphique n'est pas disponible si votre système dispose de moins de 24K. Avec les systèmes 24K, établissez HIMEM: à 16384 avant d'user de HGR2, afin de protéger vos programmes et vos variable, ou vos graphismes.
Vous ne pouvez employer HGR2 et le DOS simultanément que si votre système dispose de 36K octets de mémoire au moins. Cela signifie qu'il vous faut au moins 36K pour utiliser HGR2 en Applesoft sur disque.
N'essayez pas d'établir une fenêtre de texte avec POKE −16301,0. Vous afficheriez alors la page 2 en graphique basse résolution alors que vos commandes iraient en page 1 et resteraient invisibles (mais seraient toutefois exécutées correctement).
Non disponible en Integer Basic.

HIMEM:

Etablit une frontière supérieure en mémoire pour les programmes en Basic et les variables.

Format :

HIMEM : *exprnm*

HIMEM: fixe la plus haute position, en mémoire vive (RAM), disponible pour vos programmes en Basic. Le DOS réside toujours au-dessus de HIMEM: s'il est présent. Avec l'ordre HIMEM:, vous pouvez réserver des espaces additionnels pour vos sous-programmes en langage machine et vos fichiers de formes graphiques haute résolution. Vous pouvez également protéger la zone de la mémoire graphique haute résolution, en RAM.
HIMEM: est d'abord positionné sur la plus haute position mémoire de votre Apple II (par exemple, 49151 pour un 48K). Le DOS résidera au-dessus, et ajustera HIMEM: plus bas d'environ 10800 octets lors de son appel. Chaque buffer supplémentaire de fichier que vous réservez via MAXFILES baisse encore HIMEM: de 595 octets. Si votre programme Applesoft emploie des chaînes, leurs valeurs sont rangées à partir de la position résultante de HIMEM:, toujours en allant vers le bas. Voyez la carte de la mémoire dans l'appendice G.
La valeur de l'expression *exprnm* doit être comprise dans la gamme −65535 à 65535 (ou −32767 à 32767 en Integer Basic) faute de quoi un message d'erreur intervient.
Vous ne devez pas positionner HIMEM: plus haut que la position supérieure de la mémoire disponible ; sinon, vos données pourront se perdre dans une mémoire inexistante.
Vous pouvez afficher la valeur courante de HIMEM: grâce à l'instruction appropriée, comme ci-dessous :

```
PRINT PEEK(116)*256+PEEK(115)      pour l'Applesoft
PRINT PEEK(77)*256+PEEK(76)        pour l'Integer Basic
```

HIMEM : n'est pas affectée par NEW, RUN ou CLEAR.
Si vous positionnez HIMEM: plus bas que LOMEM:, ou si vous ne disposez pas d'assez de mémoire pour lancer votre programme, un message d'erreur surviendra.
Ne peut être employé en mode immédiat qu'en Integer Basic.

HLIN

Trace une ligne horizontale sur l'écran en mode graphique basse résolution.

Format :

HLIN col_1, col_2 AT *rang*

La ligne est tracée de col_1 à col_2 dans la rangée spécifiée. La couleur est déterminée par la dernière instruction COLOR exécutée. Si l'écran est en mode texte, ou si la fenêtre de texte est présente lorsque *rang* est supérieure à 39, HLIN trace une ligne de caractères sur l'écran, dans la fenêtre de texte, là où les points graphiques auraient dû apparaître. Les caractères sont déterminés par l'instruction COLOR précédente ; voyez le tableau 8.5 (avec l'instruction PLOT, dans ce chapitre) pour les cas particuliers.
En Integer Basic, col_1 doit être inférieure ou égale à col_2, faute de quoi le message ***RANGE ERR sera affiché.

HOME

Cet ordre Applesoft efface l'écran et positionne le curseur dans l'angle supérieur gauche de l'écran (rangée 1 et colonne 1).

Format :

HOME

En Integer Basic, employez CALL-936.

HPLOT

Cette instruction Applesoft place un point ou trace un trait de couleur sur l'écran en mode graphique haute résolution.

Formats :

HPLOT *colh, rangh*
HPLOT TO *colh, rangh*
HPLOT $colh_1$, $rangh_1$ TO $colh_2$, $rangh_2$ [TO $colh_3$, $rangh_3$...]

Le premier format de cette commande place un point de couleur à la position spécifiée. La couleur est déterminée par la dernière instruction HCOLOR.
Le second format trace une ligne de couleur à partir du dernier point tracé jusqu'aux coordonnées *colh* et *rangh*. S'il n'y a pas eu de point tracé depuis les dernières commandes HGR ou HGR2, rien ne sera tracé. La couleur est déterminée par la dernière instruction HCOLOR.
Le troisième format trace aussi une ligne de couleur. Avec Applesoft en firmware, la ligne peut disposer de plus d'un segment. Elle est d'abord tracée de $colh_1$ et $rangh_1$ jusqu'à $colh_2$ et $rangh_2$. La couleur est déterminée (pour tous les segments) par la plus récente instruction HCOLOR.
Des coordonnées additionnelles ne sont pas permises avec Applesoft en disque ou cassette. Un nouveau segment de ligne, s'il est présent dans un programme avec l'Applesoft en firmware, est alors tracé de $colh_2$ et $rangh_2$ jusqu'à $colh_3$ et $rangh_3$, et ainsi de suite. On pourra placer autant de paires de coordonnées qu'il peut en tenir sur une ligne de programme.
Toute portion de ligne ou tout point situé dans la fenêtre de texte ne sera pas visible. Cependant, si vous passez au graphique plein écran avec la commande POKE −16302,0 les lignes et points dans la fenêtre de texte apparaîtront.
Vous devez toujours exécuter un HGR ou un HGR2 avant un PLOT, faute de quoi vous pourriez détruire vos programmes et variables.
N'est pas disponible en Integer Basic.

HTAB

Cet ordre Applesoft positionne le curseur sur la colonne spécifiée, dans la ligne affichée courante.

Format :

HTAB *col*

Le curseur se déplace à droite ou à gauche sur la colonne spécifiée par la valeur de *col*, sans effacer les caractères affichés. Les colonnes sont numérotées de 1 à 40 (de la gauche vers la droite).
En Integer Basic, employez l'ordre TAB.

IF-THEN

Commande conditionnelle ordonnant au programme d'exécuter l'instruction indiquée, ou les instructions suivantes. Les règles en Integer Basic et Applesoft sont présentées séparément.

Formats en Integer Basic :

IF *expr* THEN *instruction*
IF *expr* THEN [GOTO] *ligne*

Dans la première forme, l'expression spécifie une condition qui, si elle est vraie, provoque l'exécution de l'*instruction* suivant THEN. Si la condition est fausse, l'instruction qui suit immédiatement le IF-THEN est exécutée, celle accompagnant le THEN étant alors ignorée.
Dans le second format (le format de branchement conditionnel, l'expression spécifie une condition qui, si elle est vraie, provoque un branchement du programme au numéro de ligne indiqué.
Les expressions relationnelles sont le plus fréquemment utilisées avec IF-THEN. Les valeurs chaînes peuvent être comparées pour recherche de l'égalité ou de l'inégalité en Integer Basic.
expr peut être aussi une expression numérique. Dans ce cas, *expr* est considérée vraie si sa valeur n'est pas nulle.
L'expression de IF-THEN ne peut être une expression chaîne (par exemple, tout ce qui peut être évalué sous forme de chaîne) en Integer Basic.

Formats en Applesoft :

IF *expr* THEN *instruction [:instruction...]*

IF *expr* { THEN / GOTO / THEN GOTO } *ligne*

Dans le premier format, l'expression spécifie une condition qui, si elle est vraie, provoque l'exécution de chaque instruction suivant THEN sur la même ligne de programme. Si la condition est fausse, le contrôle passe à l'instruction de la ligne suivante du programme, et aucune des instructions suivant THEN n'est exécutée.
Dans le second format (branchement conditionnel), le programme se branche à la ligne dont le numéro est *ligne* si la condition est vraie. Sinon, l'exécution se poursuit avec l'instruction de la ligne de programme qui suit le IF-THEN.
Si un branchement inconditionnel se trouve dans l'une des instructions suivant le THEN, il doit être la dernière instruction de la ligne, et il doit être au format GOTO *ligne*. S'il n'est pas la dernière instruction de la ligne, les instructions qui le suivent ne seront jamais exécutées.
Les expressions les plus couramment employées avec IF-THEN sont les relationnelles. Si des expressions chaînes sont comparées avec des opérateurs relationnels, c'est le code ASCII (liste dans l'appendice I) des caractères qui déterminera les valeurs relatives des chaînes. Les chaînes sont comparées, caractère par caractère, jusqu'à ce qu'une non-concordance survienne. La chaîne disposant alors du caractère de code ASCII le plus élevé dans cette position de non-correspondance est considérée comme plus grande. S'il n'y a pas de non-concordance, la chaîne la plus longue est la plus grande.
L'expression peut être numérique. Si sa valeur n'est pas nulle, elle est réputée vraie. Si elle est zéro (fausse), l'exécution continue par l'instruction de la ligne suivante du programme.
En Applesoft, *expr* peut être aussi une expression chaîne. Cependant, l'exécution de plus de deux ou trois de tels IF-THEN au cours d'un programme génère le message ?FORMULA TOO COMPLEX ERROR (« erreur de formule trop complexe »).

Un problème surgira en Applesoft si le dernier caractère autre qu'un espace, précédant THEN, est la lettre A. En effet, le A combiné au T forme le mot AT. Vous pourrez éviter ce problème en incluant tout ou partie de l'expression (avec le A coupable) dans des parenthèses.
Si une boucle FOR-NEXT suit le THEN, elle doit être totalement contenue dans la ligne IF-THEN. Des instructions supplémentaires IF-THEN peuvent suivre le THEN pour autant qu'elles soient complètement contenues dans la ligne IF-THEN d'origine. Cependant, des expressions booléennes seront plus claires que des IF-THEN emboîtés. Par exemple, les deux instructions suivantes sont équivalentes, mais la seconde est plus facile à lire :

```
10 IF A$=" X " THEN IF B=2 THEN IF C>D THEN 50
10 IF A$=" X " AND B=2 AND C>D THEN 50
```

IN

Sélectionne le connecteur du périphérique dont l'entrée va être acceptée.

Format :

IN # *connecteur*

Les instructions INPUT vont attendre des données en provenance du périphérique correspondant au connecteur indiqué (connecteur se dit ici, en anglais, *slot*). Le *connecteur* doit être une constante entière entre 0 et 7. Notez que le connecteur 0 n'est pas un périphérique ; IN # 0 spécifie le clavier. S'il n'y a pas de périphérique relié au connecteur indiqué, le système se bloque jusqu'à ce que vous pressiez RESET.
Lorsque le DOS est présent dans la mémoire d'Apple II, IN # est considérée comme une commande du DOS et exige un PRINT et un CTRL-D en mode programmé.

INIT

Initialise une disquette.

Format :

INIT *nomdufichier* [,D*n*] [,S*n*] [,V*n*]

Le programme courant en mémoire est sauvegardé sur le disque sous le nom de fichier donné. Il devient le programme de salutations et est lancé automatiquement chaque fois que le disque est appelé. Le disque se voit affecter le numéro de volume avec lequel il a été initialisé ; en l'absence de numéro de volume, le numéro 254 lui est attribué par défaut. Si le fichier n'existe pas sur l'unité à disquette D*n* du connecteur S*n*, le message d'erreur FILE NOT FOUND est affiché.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, les dernières références d'unité et connecteur sont prises en compte. De même, *n* peut être absent et dans ce cas, D0 et S0 sont utilisés.
INIT ne peut être employé qu'en mode immédiat.

INPUT

Accepte des entrées de caractères à partir du clavier ou d'autres périphériques, les évalue, et attribue ces valeurs aux variables spécifiées. Il peut n'y avoir qu'une entrée et qu'une variable.

Format en Integer Basic :

INPUT [" *appel* ",] *var* [,*var*...]

INPUT est la demande de valeurs, en Integer Basic, pour n'importe quelles combinaisons d'entiers et de variables chaînes. Si la première variable est un entier, un point d'interrogation est affiché à la position courante du curseur, et invite à faire l'entrée. L'Integer Basic supprime ce point d'interrogation si la première variable est une chaîne.
L'*appel*, optionnel, est une constante chaîne. S'il est présent, il est affiché juste avant que la première variable soit entrée ; il n'est pas répété pour les autres variables de la série. Un point d'interrogation apparaît après cet appel si une variable entière est attendue en entrée. L'*appel* seul est affiché si c'est une variable chaîne qui doit être entrée. Remarquez que l'*appel* est suivi par une virgule dans l'instruction INPUT. L'*appel* ne peut être une variable chaîne ou une expression chaîne.
Lorsqu'un seul ordre INPUT demande plusieurs valeurs entières à la suite, vous pouvez les introduire sur des lignes successives ; terminez chaque valeur par un RETURN. L'Integer Basic affichera un double point d'interrogation (??) sur chaque nouvelle ligne pour vous inviter à poursuivre les entrées. En option, vous pouvez introduire plus d'une valeur entière sur une seule ligne ; séparez alors les valeurs avec des virgules.
Les entrées numériques doivent consister uniquement en caractères numériques valides, donc en digits de 0 à 9, espaces, et signes plus ou moins. Vous obtiendrez un message d'erreur si vous pressez simplement RETURN alors qu'une valeur numérique est attendue.
Vous devez introduire chaque valeur chaîne sur une ligne séparée. Tous les caractères (sauf CTRL-C et CTRL-X) entrés avant le RETURN sont acceptés et attribués à des variables chaînes. Une chaîne nulle ("") est attribuée à la variable si vous pressez simplement RETURN alors qu'une valeur chaîne est attendue.
Si vous entrez des caractères inacceptables (par exemple, des lettres pour une valeur numérique), le message d'alerte ***SYNTAX ERR et RETYPE LINE (« refrappez la ligne ») apparaît. Vous devez alors ré-entrer la ligne incriminée.
Ne peut être employé en mode immédiat.

Format Applesoft :

INPUT [" *appel* ";] *var* [,*var*...]

INPUT peut demander n'importe quelle combinaison de valeurs numériques ou variables chaînes. Un point d'interrogation est normalement affiché à la position courante du curseur, invitation à effectuer l'entrée. Applesoft le supprime si l'*appel* optionnel est présent.
L'*appel* optionnel est une constante chaîne. S'il est présent, il est affiché juste avant que la première variable soit introduite ; il n'est pas répété pour les autres variables de la liste. Il n'y a pas de point d'interrogation après un *appel*. Remarquez que l'*appel* est suivi par un point-virgule dans l'instruction INPUT.
Très généralement, lorsqu'un INPUT demande plus d'une valeur, vous pouvez les introduire sur des lignes séparées ; terminez chaque entrée par RETURN. En option, vous pouvez entrer dans plus d'une valeur sur une unique ligne ; séparez alors les valeurs avec des virgules.
Si vous entrez des caractères inacceptables (par exemple, des lettres pour une valeur numérique), un message d'alerte apparaît et vous devrez refaire totalement l'entrée. Applesoft affiche REENTER et ré-exécute l'ordre INPUT à son début. L'invitation

(point d'interrogation ou *appel*) est alors ré-affichée et vous devez ré-entrer toutes les valeurs pour l'instruction INPUT.
Les entrées numériques consistent exclusivement en caractères numériques valides. Si vous pressez simplement RETURN alors qu'une variable numérique doit être entrée, vous recevrez un message d'erreur et devrez ré-entrer la ligne. Les digits 0 à 9, les espaces, les signes plus et moins sont acceptés comme entrée numérique. Applesoft accepte aussi un point décimal, un signe plus ou moins additionnel, et la lettre E pour des valeurs réelles et en notation scientifique.
En Applesoft, si le premier caractère autre qu'un espace dans une entrée de chaîne est une ouverture de guillemets, tous les caractères (y compris les virgules et doubles points) jusqu'à la fermeture des guillemets ou un RETURN se verront attribués à la variable chaîne. Si l'entrée ne commence pas par des guillemets, tous les caractères (y compris des guillemets) jusqu'à la virgule suivante, un double point ou un RETURN sont attribués à la variable. Si deux chaînes, ou davantage, sont demandées par le même INPUT, elles doivent être incluses dans des guillemets et séparées par des virgules.
Si vous pressez simplement RETURN lorsqu'une variable chaîne est attendue, une chaîne nulle (““) sera attribuée à la variable.
En Applesoft, tous les caractères placés après un double point dans une réponse à INPUT sont ignorés, sauf si l'entrée a débuté avec une ouverture de guillemets.
INPUT ne peut être employé en mode direct.

INT

Place l'Apple II en Integer Basic.

Format :

INT

Tous les programmes en mémoire sont effacés, Si l'Integer Basic n'est pas présent (par exemple, sur un Apple II Plus sans le « Language System »), le message LANGUAGE NOT AVAILABLE (« langage non disponible ») est affiché.
En mode immédiat uniquement.

INVERSE

Cet ordre Applesoft provoque le passage en mode vidéo inversé.

Format :

INVERSE

Toutes les sorties exécutées avec des ordres PRINT apparaissent en noir sur fond blanc. Les messages d'erreur sont affectés de la même façon. Cependant, les caractères émis en écho sur l'écran par des instructions INPUT ne sont pas affectés, non plus d'ailleurs que les caractères pré-existants sur l'écran.
INVERSE fonctionne en modifiant légèrement les codes standards ASCII. Aussi, des caractères envoyés au disque en mode inverse pour leur sauvegarde disposeront-ils de codes incorrects. A la lecture, on obtiendra des caractères erronés.
N'est pas disponible en Integer Basic.

LET =

Instruction d'affectation, LET = , ou plus simplement = , attribue une valeur à une variable spécifiée.

Format :

[LET] *var* = *expr*

La variable *var* se voit attribuer la valeur calculée en évaluant *expr*.

LIST

Affiche tout ou partie du programme courant en mémoire. La commande LIST dispose de deux formats. L'un est reconnu à la fois par l'Integer Basic et l'Applesoft, le second uniquement par Applesoft. Ils sont décrits séparément.

Format général :

LIST $ligne_1$ [,$ligne_2$]

Toute portion de programme peut être listée. Si aucun numéro de ligne ne suit LIST, le programme entier est affiché. Si $ligne_1$ seule est présente, cette seule ligne est affichée, si elle existe. Si les deux numéros de lignes sont présents, le programme sera affiché à partir de $ligne_1$ et jusqu'à $ligne_2$.
Si $ligne_1$ n'existe pas, le listage commence à la ligne supérieure suivante.
Si $ligne_2$ n'existe pas, le listage se termine à la ligne inférieure précédente.
LIST ne peut être utilisée avec des variables ou des expressions à la place des numéros de lignes.
Lorsque LIST affiche votre programme, elle ajoute des espaces additionnels autour des variables et des mots réservés pour rendre le listage plus lisible. Si cela vous gêne, vous pouvez éliminer les espaces en réduisant la fenêtre de texte à une longueur de 33 (ou moins) avec la commande POKE 33,33. (La commande POKE 33,40 rétablira la fenêtre de texte dans sa pleine largeur.)
Les longueurs des lignes de programme sont limitées, mais ces limites sont calculées avant que LIST ajoute des espaces additionnels. Vous pouvez donc étendre la longueur apparente de vos lignes de programme en émettant les espaces lorsque vous frappez ces lignes. Cependant, de telles lignes seront trop longues à éditer ou à copier après qu'elles aient été listées avec les espaces introduits par LIST.

Format étendu (Applesoft) :

$$\text{LIST} \begin{cases} ligne_1 \left[\left\{ \begin{matrix} - \\ , \end{matrix} \right\} \right] \\ \left[ligne_1\right] \left\{ \begin{matrix} - \\ , \end{matrix} \right\} ligne_2 \end{cases}$$

En Applesoft, une virgule ou un tiret (−) peuvent séparer deux numéros de ligne.
En Applesoft, vous pouvez lister votre programme du début jusqu'à un numéro spécifique de ligne en plaçant une virgule ou un tiret avant $ligne_2$ (et en mettant $ligne_1$). Vous pouvez aussi lister à partir d'un numéro de ligne spécifique et jusqu'à la fin en plaçant une

virgule ou un tiret après *ligne*$_1$ (et en omettant *ligne*$_2$).

LOAD

Charge un programme à partir d'une cassette ou d'une disquette.

Format pour cassette :

LOAD

Le prochain programme de la séquence, en cassette, est chargé et remplace tout programme courant en mémoire. Votre unité à cassettes doit être en mode lecture (PLAY) lorsque LOAD est exécuté ; Apple II ne vous rappellera pas à l'ordre. Au début du chargement, Apple II émet un premier « bip », puis un second à la fin du chargement. A ce moment, arrêtez votre unité à cassettes.
Vous ne pouvez employer LOAD qu'en mode immédiat en Integer Basic.

Format pour disquette :

LOAD *nomdufichier* [,D*n*] [,S*n*] [,V*n*]

Le programme nommé *nomdufichier* est chargé à partir du disque. Si le chargement est réussi, les programmes antérieurement en mémoire sont effacés.
Si le programme à charger est en Applesoft et si Apple II est alors en Integer Basic, ou vice-versa, Apple II passera sur le bon langage. Il faudra alors peut-être que vous l'ayez chargé à partir du disque spécifique. Si le langage n'est pas disponible, le message LANGUAGE NOT AVAILABLE est affiché.
Si le fichier n'existe pas sur l'unité D*n* du connecteur S*n*, le message FILE NOT FOUND est affiché. Si le disque de D*n* et S*n* n'est pas le volume V*n*, il en résulte une erreur VOLUME MISMATCH.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, les dernières références d'unité ou de connecteur sont reprises. V0 est employé à la place d'un V*n* absent. De plus, *n* peut être absent et dans ce cas, D0, S0 et V0 seront utilisés.
C'est une commande du DOS ; elle exige un PRINT et un CTRL-D en mode programme.

LOCK

Protège un fichier d'un effacement accidentel.

Format :

LOCK *nomdufichier* [,D*n*] [,S*n*] [,V*n*]

Une fois verrouillé, un fichier ne peut plus être effacé ou renommé tant qu'il reste verrouillé (voyez UNLOCK). Aucun programme portant le nom d'un fichier verrouillé ne peut être sauvegardé. Un fichier verrouillé est repéré par un astérisque à la gauche du type de fichier dans le catalogue de la disquette.
Si le fichier n'existe pas sur l'unité D*n* du connecteur S*n*, le message d'erreur FILE NOT FOUND est affiché. Si le disque de D*n* et S*n* n'est pas le volume V*n*, il en résulte une erreur VOLUME MISMATCH.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, les dernières références d'unité et de connecteur sont reprises en compte. V0 servira si V*n* est

absent. De plus, *n* peut être absent et dans ce cas, D0, S0 et V0 seront employés.
C'est une commande du DOS et elle exige un PRINT et un CTRL-D en mode programme.

LOMEM:

Etablit une frontière basse dans la mémoire disponible pour les programmes en Basic et pour les données.

Format :

LOMEM: *exprnm*

LOMEM: établit la plus basse position en mémoire vive (RAM) disponible pour vos programmes en Basic et vos variables. Le moniteur et l'interpréteur Basic utilisent la RAM en-dessous de LOMEM: pour les pointeurs, le graphique à basse résolution et la mémoire d'écran texte, etc. Lorsque l'interpréteur Applesoft est chargé d'une cassette ou d'un disque, il réside toujours en RAM au-dessous de LOMEM:. Vous pouvez réserver des espaces additionnels pour vos sous-programmes en langage machine et vos fichiers de formes graphiques haute résolution avec l'instruction LOMEM:.
En Integer Basic ou en Applesoft firmware, LOMEM: démarre à la position 2048, juste au-dessus de la zone système. Le chargement de l'interpréteur Applesoft d'un disque ou d'une cassette porte LOMEM: à 12291. Chaque fois que vous ajoutez une ligne de programme en Applesoft ou que vous modifiez une ligne existante, LOMEM: est ajusté vers le haut ou vers le bas. L'effacement d'un programme Applesoft (avec NEW ou CTRL-B) change aussi LOMEM:. Aussi, si vous voulez réserver de l'espace en-dessous de vos programmes, vous devez le faire après avoir effacé les programmes antérieurs, mais avant de charger ou frapper le nouveau programme.
La valeur de *exprnm* doit se situer dans la gamme de – 65535 à 65535 (– 32767 à 32767 en Integer Basic) faute de quoi vous recevrez un message d'erreur.
Vous pouvez afficher la valeur courante de LOMEM: avec l'instruction

PRINT PEEK(106) *256 + PEEK(105).

En Applesoft, si LOMEM: est établi supérieur à HIMEM:, ou inférieur à la valeur existante de LOMEM:, ou inférieur à la plus haute position mémoire utilisée par le système d'exploitation courant ou par le programme, le message ?OUT OF MEMORY ERROR surgit (« erreur : hors de la mémoire »).
Ne peut être employé qu'en mode immédiat en Integer Basic.

MAN

Termine la numérotation automatique des lignes en Integer Basic.

Format :

MAN

La numérotation automatique des lignes est instaurée par AUTO.
Faites CTRL-X pour stopper temporairement la génération des numéros de lignes, puis entrez MAN.
Non disponible en Applesoft.

MAXFILES

Spécifie le nombre maximum de fichiers pouvant être actifs simultanément.

Format :

MAXFILES *limite*

Le système d'exploitation (DOS) supporte un maximum de 16 fichiers ouverts simultanément. Lorsque l'instruction MAXFILES est exécutée, elle réserve 595 octets de mémoire (une pile tampon) pour chaque fichier. MAXFILES est automatiquement établi à 3 lorsque vous appelez le DOS.
Toutes les commandes du DOS, sauf MAXFILES, emploient une pile tampon lorsqu'elles sont exécutées. Ainsi, le nombre pratique maximal de fichiers que vous pouvez ouvrir en même temps est égal à la limite MAXFILES moins un. Si vous tentez d'exécuter une commande DOS alors qu'il ne reste plus aucun buffer libre, le message d'erreur NO BUFFERS AVAILABLE (« pas de buffers disponibles ») sera affiché.
MAXFILES repositionne HIMEM: lorsqu'il est exécuté, ce qui peut effacer une partie de votre programme ou de vos variables, etc. Si possible, exécutez MAXFILES avant de charger ou lancer votre programme.
Si vous utilisez MAXFILES dans un programme en Applesoft, placez-le à la première ligne. Pour employer MAXFILES en Integer Basic, vous devez créer un fichier EXEC comme on l'a discuté dans le chapitre 5 (voyez aussi EXEC dans ce chapitre).
limit doit être une constante entière entre 1 et 16, sinon, vous obtiendrez un message d'erreur RANGE ERROR.
C'est une commande du DOS ; elle exige un PRINT et un CTRL-D en mode programme.

MON

Affiche les commandes du disque et/ou le flot des données sur l'écran.

Format :

MON [C] [,I] [,O]

Les trois paramètres imposent ce qui doit être affiché. Si C est spécifié, toutes les commandes disque sont affichées sur l'écran. Si I est spécifié, toutes les entrées vers Apple II, provenant du disque, sont affichées. Si O est spécifié, toutes les sorties d'Apple II vers le disque sont affichées. Ces trois paramètres peuvent intervenir dans n'importe quelle combinaison et dans n'importe quel ordre. Si aucun d'eux n'est présent, MON n'a pas d'effet. MON reste actif jusqu'à ce qu'un NOMON, FP ou INT soit exécuté, que le système soit ré-appelé, ou, sur certaines machines, que RESET soit frappé.
C'est une commande du DOS ; elle exige un PRINT et un CTRL-D en mode programmé.

NEW

Efface le programme courant et les variables de la mémoire.

Format :

NEW

NEW repositionne aussi LOMEM : mais n'affecte pas HIMEM :, COLOR ou HCOLOR.

NEW ne peut être employé qu'en mode immédiat en Integer Basic.

NEXT

Termine une boucle commencée avec une instruction FOR.

Format général :

NEXT *varnm [,varnm...]*

Lorsque NEXT est exécuté, la variable index de la boucle *varnm* est incrémentée de la quantité spécifiée dans l'ordre correspondant FOR. Le programme se poursuit alors avec l'instruction qui suit NEXT, ou alors revient au FOR correspondant, selon les paramètres établis avec FOR. Voyez la discussion de FOR qui précède.
S'il n'y a pas de FOR actif pour la boucle, qui réponde à *varnm*, une erreur ?NEXT WITHOUT FOR ERROR (« erreur de NEXT sans FOR ») est affichée par Applesoft, alors que l'Integer Basic affiche ***BAD NEXT ERR (« erreur de mauvais NEXT »).
Des variables multiples qui suivent NEXT doivent se présenter dans le bon ordre (la dernière boucle initialisée doit se terminer la première), sinon, une erreur survient.
NEXT ne peut être employé en mode immédiat en Integer Basic. En Applesoft, un NEXT en mode immédiat pourra provoquer un branchement à un FOR exécuté en mode programme s'il est encore actif.

Format Applesoft additionnel :

NEXT

En Applesoft, vous pouvez employer NEXT sans nom de variable identificateur. La variable sera, par défaut, celle de la plus récente boucle FOR en activité. NEXT sans variable s'exécute plus vite qu'un NEXT avec variable.

NO DSP

Annule le mode affichage pour la variable spécifiée, en Integer Basic. (Voyez DSP.)

Format :

NO DSP *var*

Non disponible en Applesoft.

NOMON

Termine l'affichage des commandes disque ou du flot de données initialisé par MON.

Format :

NOMON [C] [,I] [,O]

Chaque paramètre spécifié annule une partie de l'affichage commandé par MON. Si C est spécifié, les commandes disques ne sont plus affichées. Si I est spécifié, les données entrées dans Apple II et provenant du disque ne sont plus affichées. Si O est spécifié, les

données qui sortent d'Apple II, vers le disque, ne sont plus affichées. Ces paramètres peuvent intervenir selon toutes combinaisons et dans n'importe quel ordre. Si MON n'est pas en activité pour le, ou les paramètres spécifiés, ou si aucun paramètre n'est spécifié, NOMON n'a pas d'effet.
C'est une commande du DOS ; elle exige un PRINT et un CTRL-D en mode programme.

NORMAL

Cet ordre Applesoft termine les modes FLASH et INVERSE.

Format :

NORMAL

Voyez FLASH et INVERSE.
Non disponible en Integer Basic.

NO TRACE

Termine le tracé de l'exécution d'un programme initialisé par TRACE.

Format :

NO TRACE

Si TRACE n'est pas en service, NO TRACE n'a pas d'effet.

ONERR GOTO

Branche à la ligne dont le numéro est indiqué lorsqu'une erreur survient dans un programme Applesoft.

Format :

ONERR GOTO *ligne*

Cette commande positionne un indicateur qui provoque le branchement à la ligne dont le numéro est indiqué lorsqu'une erreur survient. ONERR GOTO doit être exécuté avant que l'erreur ne se manifeste.
Chaque type d'erreur dispose d'un numéro de code. Le code de l'erreur la plus récente est stocké en mémoire à la position 222. L'ordre PEEK(222) retrouve ce code d'erreur. Les codes d'erreurs et leurs messages sont listés dans le tableau C.1 (appendice C). Lorsqu'une erreur se produit dans une boucle FOR-NEXT ou dans un sous-programme, les pointeurs et piles sont modifiés. Votre programme de traitement de l'erreur doit retourner aux ordres FOR ou GOSUB afin de redémarrer la boucle ou le sous-programme. Si le traitement de l'erreur renvoie à NEXT ou RETURN, une nouvelle erreur se manifestera.
ONERR GOTO pourra ne pas fonctionner correctement dans certaines circonstances. L'Apple II pourra se bloquer s'il y a deux erreurs GET dans une ligne et si le programme de traitement de l'erreur se termine par RESUME, et non par GOTO. Dans les programmes employant des instructions PRINT (ou encore, si TRACE est actif), la 43e erreur qui ne provient pas d'une instruction INPUT provoque un saut au moniteur. Dans cette situation, si c'est un GOTO qui achève le programme de traitement d'erreur (au lieu d'un

RESUME), la 87e erreur INPUT déclenche un saut au moniteur.
Vous pouvez éviter tous ces problèmes si votre programme de traitement d'erreur inclut un appel au programme en langage machine, comme le montre le tableau 8.4.

Tableau 8.4. — Programme en langage machine pour ONERR GOTO.

LANGAGE MACHINE		LANGAGE ASSEMBLEUR DU 6502	
Décimal	**Hexadécimal**	**Instruction**	**Commentaires**
104	68	PLA	Range l'octet du sommet de la pile dans l'accumulateur
168	A8	TAY	et le sauvegarde dans le registre d'index Y
104	68	PLA	Place l'octet suivant de la pile dans l'accumulateur
166	A6	LXD $DF	Utilisation du pointeur ONERR
223	DF		
154	9A	TXS	comme adresse de pile
72	48	PHA	Charge les contenus sauvegardés de la pile
152	98	TYA	dans la pile ONERR (deux octets –
72	48	PHA	de l'accumulateur et du registre Y)
96	60	RTS	Retour à Applesoft

Utilisez l'ordre POKE pour ranger les nombres décimaux aux positions mémoires 768 à 777 (ou toutes autres positions disponibles). Puis appelez 768 (CALL 768) à partir de votre programme de traitement d'erreur.
Non disponible en Integer Basic.
Ne peut être employé en mode immédiat.

ON-GOSUB

Fournit un appel conditionnel à un ou plusieurs sous-programmes, dans un programme Applesoft, selon la valeur de l'expression.

Format :

ON *exprnm* GOSUB *ligne [,ligne...]*

Le programme branche au premier numéro de ligne si la valeur entière de l'expression est 1, au second si c'est 2. Le prochain ordre RETURN renvoie à l'instruction qui suit immédiatement ON-GOSUB.
L'expression doit avoir une valeur comprise entre 0 et 255 ; sinon, le message ?ILLEGAL QUANTITY ERROR apparaît. Si l'expression donne un zéro ou une valeur supérieure au nombre de numéros de lignes listés, l'exécution se poursuit à l'instruction suivant le ON-GOSUB.
Non disponible en Integer Basic. (Mais voyez GOSUB pour la forme du GOSUB calculé en Integer Basic).

ON-GOTO

Provoque un branchement conditionnel à l'un des points d'un programme Applesoft, selon la valeur courante de l'expression.

Format :

ON *exprnm* GOTO *ligne [,ligne...]*

Le programme branche au premier numéro de ligne si la valeur entière de l'expression est 1, au second si cette valeur est 2, etc.
L'expression doit avoir une valeur numérique comprise entre 0 et 255 ; sinon, le message ?ILLEGAL QUANTITY ERROR survient. Si l'évaluation de l'expression donne un zéro ou une valeur supérieure au nombre de numéros de lignes listés, l'exécution se poursuit avec l'instruction qui suit le ON-GOTO.
Non disponible en Integer Basic. (Mais voyez GOTO pour la forme calculée en Integer Basic.)

OPEN

Prépare l'accès à un fichier séquentiel ou aléatoire de texte sur disque.

Format :

OPEN *nomdufichier* [,L*n*] [,D*n*] [,S*n*] [,V*n*]

Un buffer mémoire de 595 octets est attribué au fichier de texte spécifié. Les commandes READ et WRITE peuvent alors être employées pour rechercher ou ranger les informations. Si le fichier spécifié n'est pas sur le disque, il en est créé un. Si le fichier est déjà sur le disque, il est fermé puis ré-ouvert.
Si le paramètre L n'est pas spécifié, c'est un fichier séquentiel qui est ouvert.
Si le paramètre L est spécifié, le fichier est du type aléatoire. Le nombre *n* suivant L est la longueur de l'enregistrement, en octets, et doit être une constante entière comprise entre 1 et 32767. Il peut être absent : L1 sera utilisé. Chaque fois qu'un fichier aléatoire particulier est ouvert, la longueur de l'enregistrement doit rester la même.
Si la disquette dans l'unité D*n* du connecteur S*n* n'est pas le volume V*n*, une erreur VOLUME MISMATCH intervient.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, les dernières références à l'unité de disquettes et au connecteur sont reprises en compte. V0 est pris pour un V*n* absent. De plus, *n* peut être absent et ce seront D0, S0 et V0 qui seront alors utilisés.
Cette commande est du DOS ; elle exige un PRINT et un CTRL-D en mode programme. OPEN ne peut intervenir qu'en mode immédiat.

PDL

Listé dans la section « Fonctions » de ce chapitre.

PEEK

Listé dans la section « Fonctions » de ce chapitre.

PLOT

Affiche un point sur l'écran graphique basse résolution.

Format :

PLOT *col, rang*

En mode graphique basse résolution, PLOT place un point de couleur sur l'écran. La couleur est déterminée par la dernière instruction COLOR exécutée. La gamme des numéros de colonnes va de 0 à 39, le 0 étant à gauche de l'écran et le 39 à droite. Les rangées vont de 0 à 47, avec 0 au sommet de l'écran et 47 en bas. Un point tracé dans les rangées 40 à 47 se trouvera dans la fenêtre de texte de quatre lignes, sauf si un POKE – 16302,0 a d'abord été exécuté pour supprimer cette fenêtre.
En mode texte ou dans la fenêtre de texte, PLOT place un caractère en couleur, et un point, sur l'écran. Parce qu'un caractère occupe la place de deux points graphiques verticaux, il existe deux jeux différents de coordonnées qui, avec PLOT, provoqueront l'apparition d'un caractère en un emplacement donné. Pour placer un caractère particulier sur l'écran, vous devez commander deux PLOT pour chaque moitié de la position du caractère. Sa couleur dépendra du dernier ordre COLOR exécuté pour chaque moitié.
Le tableau 8,5 montre les caractères produits par chaque combinaison de couleurs dans les points supérieur et inférieur. Pour créer un caractère particulier, cherchez-le dans le tableau 8.5 ; lisez le haut du tableau et sa gauche pour trouver la couleur à commander par PLOT à chaque moitié du caractère. La moitié supérieure est spécifiée par un PLOT dans une rangée paire, la moitiée inférieure par un PLOT dans une rangée impaire. Un PLOT pour une moitié seulement produit un caractère basé sur la couleur du PLOT et la couleur déjà présente dans les autres points graphiques.

Tableau 8.5. — Caractères produits par les commandes graphiques en mode texte.

	MODE VIDEO				COULEUR DU POINT GRAPHIQUE SUPERIEUR															
	Pour vidéo inversée	Pour caractère clignotant	Pour caractère normal		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
COULEUR DU POINT GRAPHIQUE INFERIEUR	0	4	8	12	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	1	5	9	13	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	2	6	10	14	sp	!	"	#	$	%	&	'	(	)	*	+	,	–	.	/
	3	7	11	15	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?

Remarquez que chaque caractère peut être produit par quatre couleurs différentes pour le point graphique inférieur. Ils ne sont cependant pas identiques. Le caractère sera en vidéo inversée si le numéro de couleur est inférieur à 4, il flashera si le numéro va de 4 à 7, et il sera normal de 8 à 15.

POKE

L'ordre POKE range un octet de donnée dans la position mémoire spécifiée.

Format :

POKE *memadr, octet*

Une valeur comprise entre 0 et 255, fournie par *octet*, est écrite dans la mémoire à la position *memadr*. Si la position mémoire spécifiée excède la position maximale en mémoire (par exemple, 16383 pour 16K de mémoire), ou adresse un périphérique qui n'est pas en réception, POKE n'aura pas d'effet.
Utilisez POKE avec précaution. Certaines positions mémoires contiennent des informations essentielles pour le fonctionnement continu de Apple II. En changeant au hasard le contenu de positions mémoires, vous pourrez détruire votre programme, bloquer votre système, ou détériorer votre Basic.

POP

Fait oublier à Apple II l'adresse de retour pour l'instruction GOSUB la plus récente.

Format :

POP

POP modifie effectivement l'instruction GOSUB exécutée la plus récente et en fait un GOTO (après coup). Le prochain ordre RETURN exécuté branchera à l'instruction suivant immédiatement le second GOSUB exécuté le plus récemment. Si le total de POP et de RETURN exécutés dans un programme excède le nombre de GOSUB exécutés, un message d'erreur surviendra.

POSITION

Déplace le pointeur du fichier disque du nombre spécifié de champs à partir de sa position courante.

Format :

POSITION *nomdufichier* [,R*n*]

Si le fichier n'a pas été ouvert avant que POSITION soit exécuté, il sera ouvert (voir OPEN). Le paramètre R spécifie de combien de champs le pointeur du fichier doit avancer à partir de sa position courante. Le nombre suivant R doit être une constante entière, de 0 à 32767. S'il est absent, il est considéré égal à zéro, ce qui signifie que le pointeur ne bouge pas. Si le fichier est ouvert par POSITION, les champs sont comptés à partir du début du fichier. Celui-ci doit être séquentiel.
Un champ consiste en une séquence de caractères se terminant par un retour chariot. POSITION parcourt le fichier, caractère par caractère, comptant les retours chariot. Le nombre de retours chariot rencontrés est le nombre de champs à sauter. Si un espace inemployé est rencontré dans le fichier avant que le nombre de champs spécifié ait été atteint, c'est un message END OF DATA (« fin des données ») qui apparaît.
C'est une commande du DOS, qui exige un PRINT et un CTRL-D en mode programme. POSITION ne peut servir en mode immédiat.

PR

Sélectionne le connecteur du périphérique qui recevra la sortie.

Format :

PR # *connecteur*

Les instructions PRINT suivantes vont émettre des données vers le périphérique relié au connecteur indiqué. (Dans Apple II, ces connecteurs sont marqués « slot ») ; *connecteur* doit être une constante entière entre 0 et 7. Notez que le connecteur 0 ne correspond pas à un périphérique. PR # 0 spécifie l'écran d'affichage standard 40 colonnes comme organe de sortie. S'il n'y pas de périphérique relié au connecteur indiqué, le système se verrouillera jusqu'à ce que vous pressiez RESET.
Lorsque le DOS est présent dans la mémoire d'Apple II, PR # est considéré comme étant une commande du DOS, exigeant un PRINT et un CTRL-D en mode programmé.

PRINT

Sort des caractères sur l'écran ou tout autre périphérique.

Format :

PRINT *[expr] [{;,}... [expr] ...]*

Il existe un certain nombre de variantes acceptables pour PRINT. Par lui-même, PRINT sort un retour chariot et un passage à la ligne suivante. Lorsque PRINT est suivi par une ou plusieurs expressions, leurs valeurs sont affichées. La façon dont elles apparaissent dépend de leur nature et de l'usage de points-virgules ou virgules entre ces valeurs.
Toutes les valeurs numériques en Integer Basic, et beaucoup en Applesoft, sont affichées en représentation numérique standard. Les valeurs négatives sont précédées par le signe moins ; les valeurs positives ne sont pas précédées par un signe plus ou un espace. La notation scientifique est utilisée par Applesoft pour des valeurs plus proches du zéro que ±.01 et pour toutes valeurs de plus de neuf digits en tête du point décimal.
Les valeurs chaînes sont affichées telles quelles.
Les virgules et les points-virgules déterminent l'espacement entre les valeurs affichées. Un point-virgule commande l'affichage de la valeur suivante immédiatement après la valeur précédente ; ces valeurs sont concaténées sans espace intermédiaire. Une virgule commande l'affichage de la valeur suivante à la position suivante de tabulation, donc à plusieurs espaces de la valeur précédente.
En Integer Basic, les marques de tabulation viennent tous les huit espaces, aux colonnes 1, 9, 17, etc. Si un caractère autre qu'un espace est affiché juste avant la marque de tabulation (par exemple, dans la colonne 16), il désactive cette marque de tabulation.
Applesoft sépare les marques de tabulation de 16 caractères, aux colonnes 1, 17, 33, etc. Les marques de tabulation sur l'écran sont désactivées en accord avec le schéma donné dans la figure 4.1 (chapitre 4). Pour d'autres périphériques, les marques de tabulation seront désactivées si un caractère non espace est imprimé tout juste avant la marque (par exemple, dans la colonne 32).
Si la liste des expressions ne se termine pas par une virgule ou un point-virgule, un retour chariot et un changement de ligne sont exécutés après le dernier item de la liste. Si la liste se termine par un point-virgule, le premier caractère affiché par le PRINT suivant prendra place immédiatement après le dernier caractère affiché par le PRINT courant, sans espace. Si la ligne se termine par une virgule, la prochaine sortie se fera à la position de tabulation suivante.
En Integer Basic, tous les items doivent être séparés par une virgule ou un point-virgule. En Applesoft, les items peuvent être listés sans virgule ni point-virgule. Les sorties sont alors concaténées, comme si ces items avaient été séparés par des points-virgules.
Applesoft reconnaît dans le point d'interrogation, une abréviation pour PRINT. Le mot PRINT sera néanmoins écrit dans le listage du programme.

READ (COMMANDE DISQUE)

Spécifie un fichier disque à partir duquel des ordres ultérieurs INPUT et GET obtiendront des données.

Format :

READ *nomdufichier* [,R*n*] [,B*n*]

Si le fichier spécifié n'a pas été ouvert au préalable, il sera ouvert (voir OPEN). Tous les ordres INPUT et GET reçoivent leurs caractères du disque jusqu'à ce qu'une instruction disque (ou un caractère CTRL-D seul — code ASCII 4) soit affiché, que RESET soit frappé, ou qu'une erreur survienne. Si le fichier n'est pas sur le disque, le message FILE NOT FOUND apparaît.
Le fichier sera lu en séquentiel si le paramètre R est absent. Dans un fichier séquentiel, le paramètre B spécifie l'octet (le caractère) à partir duquel READ commence. Si B est absent, READ débute par le premier octet du fichier (octet 0). Si l'octet à lire n'a jamais été rangé sur le disque par une commande WRITE, un message END OF DATA (« fin de données ») est affiché lorsque l'un des ordres INPUT ou GET est exécuté.
Si le paramètre R est présent, le fichier est considéré comme étant aléatoire. Le paramètre R spécifie l'enregistrement du fichier qui sera lu par READ.
Dans un fichier aléatoire, le paramètre B précise à quel octet de l'enregistrement spécifié la lecture doit commencer. En l'absence du paramètre B, READ débute avec le premier octet de l'enregistrement (octet 0). Si l'octet à lire n'a jamais été enregistré sur le disque par une commande WRITE, le message END OF DATA est affiché lorsque INPUT ou GET sont exécutés.
Les nombres accompagnant B et R doivent être des constantes entières entre 0 et 32767. En leur absence, zéro est pris par défaut.
N'employez pas CTRL-C pour stopper un ordre READ en Applesoft. Vous obtiendriez une série de messages ?REENTER. N'utilisez que RESET pour stopper le programme.
C'est une commande du DOS, exigeant un PRINT et un CTRL-D en mode programme. Ne peut être employée en mode immédiat.

READ (ORDRE D'AFFECTATION)

Affecte des valeurs DATA à des variables, en Applesoft.

Format :

READ *var [,var...]*

C'est un pointeur sur les données (DATA) qui détermine quelle valeur assigner à la première variable de l'instruction READ. Au démarrage d'un programme et après un ordre RESTORE, le pointeur pointe la première donnée de la liste des DATA. Après chaque READ, chaque variable se voit assigner une valeur, le pointeur progressant d'une valeur à la suivante.
Les variables peuvent être de tous types, mais leur type doit correspondre à celui de la valeur dans la liste DATA. Une valeur numérique attribuée à une variable chaîne ne créera pas de problème. Une chaîne attribuée à une variable numérique donnera un message ?SYNTAX ERROR. Le numéro de ligne de l'ordre DATA fautif sera annoncé avec le message d'erreur.
Si READ tente d'affecter plus de variables qu'il n'y en a dans DATA, le message ?OUT OF DATA ERROR (« erreur de fin de données ») apparaît, accompagné du numéro de la ligne READ fautive.
READ peut être exécuté en mode immédiat tant que le programme en mémoire contient assez de valeurs DATA. Sinon, le message ?OUT OF DATA ERROR apparaît. Si le DOS est présent, un READ en mode immédiat est interprété comme une commande du DOS et le message NOT DIRECT COMMAND est affiché.
Non disponible en Integer Basic.

RECALL

Retrouve un tableau numérique sur cassette, en Applesoft.

Format :

RECALL *varnm*

Applesoft attend indéfiniment que le tableau soit trouvé sur la bande magnétique ; aucune autre instruction ne peut être exécutée entretemps. RECALL ne commande pas le mouvement de la bande ni ne prévient qu'il faut mettre la cassette en mode lecture. On devrait donc placer des ordres PRINT avant et après le RECALL pour rappeler ces actions à mener. Apple II émet un « bip » lorsqu'il commence à recevoir les valeurs du tableau, et un second lorsque l'opération est terminée. Le tableau doit être dimensionné avec l'instruction RECALL, faute de quoi un message ?OUT OF DATA ERROR survient (voir DIM).
Vous n'êtes pas obligé d'employer le même nom de variable tableau dans l'ordre RECALL que dans l'ordre STORE qui l'avait sauvegardé. Vous devez utiliser un tableau avec les mêmes dimensions, cependant. Si le tableau sauvegardé contient davantage d'éléments que celui rappelé (RECALL), vous recevrez le message ?OUT OF DATA ERROR.
Si le tableau rappelé contient au moins autant de données que celui sauvegardé, mais n'a pas exactement les mêmes dimensions, un message ERR est généré mais l'exécution du programme se poursuit.
Si le tableau rappelé a plus d'éléments que le tableau sauvegardé, les valeurs du tableau rappelé seront généralement brouillées. Il y a des exceptions. Vous pouvez rappeler un tableau disposant du même nombre de dimensions que le tableau sauvegardé, dans lequel chaque dimension, excepté la dernière, est la même que la dimension correspondante du tableau sauvegardé. La dernière dimension peut être supérieure à celle du tableau rappelé.
Vous pouvez aussi rappeler un tableau avec davantage de dimensions que le tableau sauvegardé si les dimensions du tableau correspondent aux dimensions équivalentes du tableau rappelé (ou les excèdent, dans le cas de la dernière dimension du tableau sauvegardé).
Les tableaux de chaînes ne peuvent être utilisés avec RECALL. Mais les valeurs numériques rappelées peuvent être converties en valeurs chaînes avec la fonction CHR$.
Non disponible en Integer Basic.

REM

L'ordre « remarque » (REM) sert à placer des commentaires dans le programme afin de le documenter.

Format :

REM *comment*

comment est une séquence quelconque de caractères tenant sur la ligne de programme courante.
Les remarques sont reproduites dans les listages du programme, mais sont autrement ignorées. Une REM peut être placée sur une ligne propre, ou à la fin d'une ligne multi-instruction.
REM ne peut être placée en tête d'autres instructions sur une ligne multi-instruction car le texte qui suit REM est traité comme un commentaire.

RENAME

Change le nom d'un fichier sur disque sans modifier son contenu.

Format :

RENAME *nomdufichier*$_1$, *nomdufichier*$_2$ *[,Dn] [,Sn] [,Vn]*

Le fichier de nom *nomdufichier*$_1$ est trouvé sur la disquette et son nom devient *nomdufichier*$_2$. S'il était ouvert, il sera fermé (voir CLOSE). Le fichier n'est pas affecté par ailleurs.
RENAME attribuera directement un nom déjà existant sur le disque ; il pourra le faire plusieurs fois. Assurez-vous donc qu'il n'existe pas déjà de fichier appelé *nomdufichier*$_2$ avant d'exécuter RENAME.
Si *nomdufichier*$_1$ n'existe pas sur l'unité à disques D*n* reliée au connecteur S*n*, le message FILE NOT FOUND est affiché. Si le disque en D*n* et S*n* n'est pas le volume V*n*, l'erreur VOLUME MISMATCH survient.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, les dernières références d'unité ou de connecteur sont reprises. Si *n* est absent, D0, S0 et V0 sont utilisés.
C'est une commande du DOS ; elle exige un PRINT et un CTRL-D en mode programmé.

RESTORE

Repositionne le pointeur de liste de données (DATA), en Applesoft, au début de la liste.

Format :

RESTORE

Les ordres READ suivants reprennent à la première valeur des DATA.
Non disponible en Integer Basic.

RESUME

Fait reprendre l'exécution d'un programme Applesoft au début de l'instruction où s'est produite une erreur.

Format :

RESUME

RESUME ne peut être employé qu'après qu'un ONERR GOTO ait été déclenché par une erreur. Si RESUME est exécuté alors qu'aucune erreur ne s'est produite, le résultat sera imprévisible, mais généralement tragique.

Non disponible en Integer Basic.
Ne peut être employé en mode immédiat.

RETURN

Branche le programme à l'instruction qui suit le plus récent GOSUB exécuté.

Format :

RETURN

Une instruction POP supprimera la connaissance du plus récent GOSUB et, de ce fait, le RETURN après un POP branchera à l'instruction qui suit l'avant-dernier GOSUB.
Si davantage de RETURN (et de POP) que de GOSUB sont exécutés dans un programme, un message d'erreur sera émis.

ROT

Cet ordre Applesoft établit l'orientation des dessins haute résolution tracés par XDRAW ou DRAW.

Format :

ROT = *exprnm*

ROT = 0 trace la forme selon l'orientation avec laquelle elle a été définie. La forme pivote de 90 degrés dans le sens des aiguilles d'une montre pour chaque incrément de 16 de la valeur de *exprnm*. Ainsi, ROT = 32 renverse la forme et ROT = 64 la rétablit dans son orientation d'origine. Les valeurs de *exprnm* supérieures à 64 sont évaluées modulo 64.
Lorsque SCALE a été établi à 1, seules quatre valeurs sont reconnues pour ROT = . Ce sont : 0, 16, 32 et 48 (ainsi que des valeurs supérieures à 63 équivalentes). Lorsque SCALE = 2, huit valeurs sont reconnues, et 16 pour SCALE = 3, etc., et ce jusqu'à un maximum de 64. Une valeur non reconnue de ROT = sera traitée comme s'il s'agissait de la valeur reconnue inférieure la plus proche.
L'expression *exprnm* doit avoir une valeur comprise entre 0 et 255, sinon le message ?ILLEGAL QUANTITY ERROR sera affiché lors de l'exécution de ROT = .
ROT = n'est pas reconnu comme un mot réservé, sauf si le caractère « = » est le premier suivant un caractère autre qu'un espace.
Non disponible en Integer Basic.

RUN (COMMANDE DISQUE)

Charge un programme d'une disquette et lance son exécution.

Format :

RUN *nomdufichier* [,D*n*] [,S*n*] [,V*n*]

Le programme appelé *nomdufichier* est chargé de la disquette et exécuté. Si le chargement est réussi, tout programme antérieurement en mémoire est effacé.
Si le programme à charger et exécuter est en Integer Basic et Apple II en Applesoft, ou vice-versa, l'ordinateur commutera dans le bon langage. Si nécessaire, il chargera l'inter-

préteur Applesoft à partir du disque spécifié. Si le langage n'est pas disponible, le message LANGUAGE NOT AVAILABLE est affiché.
Si le fichier n'existe pas sur l'unité D*n* du connecteur S*n*, le message FILE NOT FOUND est affiché. Si la disquette n'est pas le volume V*n*, l'erreur VOLUME MISMATCH survient.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, les dernières références d'unité ou de connecteur sont reprises. Si V*n* est absent, V0 est utilisé. Si *n* est absent, D0, S0 et V0 sont utilisés.
C'est une commande du DOS ; elle exige un PRINT et un CTRL-D en mode programme.

RUN (ORDRE GÉNÉRAL)

Exécute le programme courant en mémoire, à partir du numéro de ligne spécifié, s'il est présent ; sinon, à partir de la ligne de plus faible numéro dans le programme.

Format général :

RUN *[ligne]*

Si ce numéro de *ligne* n'existe pas, vous recevrez le message ***BAD BRANCH ERR de l'Integer Basic, ou ?UNDEF'D STATEMENT ERROR d'Applesoft.

Format additionnel en Integer Basic :

RUN *exprnm*

En Integer Basic, le numéro de ligne peut être donné par une expression numérique.
RUN ne peut être employé qu'en mode immédiat en Integer Basic.

SAVE

Sauvegarde le programme courant en mémoire sur cassette ou disque.

Format cassette :

SAVE

Le programme en mémoire est sauvegardé sur la bande magnétique. Il faut que l'unité à cassette soit en mode enregistrement (RECORD) lorsque SAVE est exécuté, Apple II ne vous le rappellera pas. Il émet un « bip » au démarrage de la sauvegarde, et un second « bip » à son achèvement. A ce moment, stoppez l'unité à cassettes.
Ne peut être employé qu'en mode immédiat en Integer Basic.

Format disquette :

SAVE *nomdufichier* [,D*n*] [,S*n*] [,V*n*]

Si le *nomdufichier* n'existe pas sur le disque, un fichier est créé sous ce nom et dans le langage du programme. Le programme est sauvegardé. S'il se trouve déjà un fichier de ce nom et dans le même langage, son contenu est effacé et le programme courant est sauvegardé à sa place. Si le programme *nomdufichier* existe mais dans un langage différent, ou avec un type de fichier différent, le message FILE TYPE MISMATCH apparaît.

Si le disque de l'unité D*n*, du connecteur S*n*, n'est pas le volume V*n*, il en résulte une erreur de VOLUME MISMATCH.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, leurs dernières références sont reprises en compte. Si V*n* est absent, V0 est utilisé. Si *n* est absent, D0, S0 et V0 sont utilisés.
C'est une commande du DOS ; elle exige un PRINT et un CTRL-D en mode programmé.

SCALE

Cet ordre Applesoft établit la taille des formes graphiques en haute résolution tracées par DRAW ou XDRAW.

Format :

SCALE = *exprnm*

La taille de la forme, d'après le fichier des formes, est multipliée par la valeur entière de *exprnm*. Si SCALE = 1, la forme est tracée telle qu'elle a été définie ; si SCALE = 2, sa taille est double, etc. Si SCALE = 0, elle est tracée 255 fois sa taille d'origine.
La valeur d'*exprnm* doit être comprise entre 0 et 255 ; sinon, le message ?ILLEGAL QUANTITY ERROR survient à l'exécution de SCALE.
SCALE n'est pas reconnu comme un mot réservé, sauf si le caractère « = » est le premier suivant un caractère autre qu'un espace.
Non disponible en Integer Basic.

SHLOAD

Cet ordre Applesoft charge un fichier de formes graphiques haute résolution à partir d'une cassette.

Format :

SHLOAD

La construction et la sauvegarde des fichiers de formes ont été discutés dans le chapitre 6.
Le fichier de formes est chargé en mémoire juste en-dessous de HIMEM: et HIMEM: est positionné juste en-dessous de ce fichier. La position de début du fichier est rangée en mémoire, en 22 et 23.
Avant d'exécuter un SHLOAD, vérifiez que vous avez établi HIMEM: de façon que le fichier de formes ne vienne pas sur la fin de votre programme ou de vos variables, qui seraient alors effacés pour le graphique. Reportez-vous à la discussion de HIMEM: dans ce chapitre, à la carte mémoire dans l'appendice G, et au chapitre 6.
Non disponible en Integer Basic.

SPEED

Cet ordre Applesoft modifie la vitesse de sortie des caractères.

Format :

SPEED *exprnm*

La valeur d'*exprnm* établit le rythme d'apparition des caractères sur l'écran ou tout autre

périphérique. Les vitesses vont de 0 (la plus faible) à 255 (la plus haute).
Non disponible en Integer Basic.

STOP
Arrête l'exécution d'un programme en Applesoft.

Format :

STOP

Apple II revient en mode immédiat. Le message BREAK IN *ligne* est affiché, où *ligne* est le numéro de ligne à laquelle se trouvait le STOP exécuté.
Non disponible en Integer Basic.

STORE
Sauvegarde le tableau Applesoft spécifié sur bande magnétique.

Format :

STORE *varnm*

STORE ne commande pas le mouvement de la bande et ne prévient pas l'opérateur qu'il faut mettre l'unité à cassettes en mode enregistrement. Cette unité doit être en marche (enregistrement) lorsque STORE est exécuté. Votre programme devrait vous fournir ces conseils (via des PRINT). Apple II émet un « bip » en début et en fin de sauvegarde.
Vous ne pouvez sauvegarder que des tableaux numériques ; les tableaux de chaînes doivent au préalable se convertir en valeurs entières avec la fonction ASC (voir aussi RECALL).
Non disponible en Integer Basic.

TAB
Cet ordre en Integer Basic positionne le curseur sur la colonne spécifiée de la ligne courant.

Format :

TAB *col*

Le curseur se déplace vers la droite ou la gauche et va sur la colonne spécifiée par la valeur *col,* sans effacer les caractères affichés. Les colonnes sont numérotées de 1 à 40 (de gauche à droite).
En Applesoft, employez l'ordre HTAB. Voyez aussi la fonction TAB listée dans la section « Fonctions » de ce chapitre.

TEXT
Replace l'écran en mode usuel plein texte, après un mode graphique.

Format :

TEXT

Le caractère d'appel et le curseur sont déplacés sur la dernière ligne de l'écran ; en mode texte, ce sera le seul résultat. Si la fenêtre de texte a été établie dans un format autre que l'écran plein texte, TEXT rétablit l'écran plein texte.
TEXT n'efface pas l'écran, ou plus précisément, n'efface pas la page 1 de la mémoire d'écran basse résolution. Parce que le mode texte normal utilise la même mémoire d'écran que le graphique haute résolution, l'exécution de TEXT alors qu'on se trouve en mode graphique basse résolution laissera les 20 lignes supérieures de l'écran remplies de caractères bizarres.

TRACE
Affiche le numéro de ligne de chaque instruction exécutée.

Format :

TRACE

Cette aide à la mise au point provoque l'affichage des numéros de ligne avec l'affichage des sorties du programme, rendant l'un ou l'autre illisible (ou tous deux). TRACE est supprimée par NO TRACE.

UNLOCK
Déverrouille un fichier verrouillé sur disque, permettant de le supprimer ou de le modifier.

Format :

UNLOCK *nomdufichier* [,D*n*] [,S*n*] [,V*n*]

Si le fichier spécifié est verrouillé, le verrou saute. S'il n'est pas verrouillé, rien ne se passe (voir LOCK).
Si le fichier n'existe pas sur l'unité D*n* du connecteur S*n*, le message d'erreur FILE NOT FOUND est affiché. Si le disque de D*n* et S*n* n'est pas le volume V*n*, l'erreur VOLUME MISMATCH se produit.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont omis, les dernières références d'unité ou de connecteur sont reprises en compte. Si V*n* est absent, V0 est utilisé. Si *n* est absent, D0, S0 et V0 sont utilisés.
C'est une commande du DOS ; elle exige un PRINT et un CTRL-D en mode programme.

USR
Listé dans la section « Fonctions » de ce chapitre.

VERIFY
Teste un fichier spécifié sur disque pour vérifier s'il est correct.

Format :

VERIFY *nomdufichier* [,D*n*] [,S*n*] [,V*n*]

Lorsqu'un fichier est sauvegardé sur disque avec un SAVE, un BSAVE ou un PRINT, une somme de test est calculée pour chaque secteur et rangée sur le disque. VERIFY recal-

cule cette somme de test et la compare à celle se trouvant sur le disque. Si ce sont les mêmes, le fichier est intact et aucun message n'est émis. S'il y a désaccord, le message I/O ERROR est produit. Tout type de fichier peut être vérifié.
Si le fichier n'existe pas sur l'unité D*n*, connecteur S*n*, le message d'erreur FILE NOT FOUND est affiché. Si la disquette de D*n* et S*n* n'est pas le volume V*n*, l'erreur VOLUME MISMATCH se produit.
D*n*, S*n* et V*n* peuvent être spécifiés dans n'importe quel ordre. Si D*n* ou S*n* sont absents, les plus récentes références à l'unité ou au connecteur sont reprises en compte. Si V*n* est absent, V0 est utilisé. Si *n* est absent, D0, S0 et V0 sont utilisés.
C'est une commande du DOS ; elle exige un PRINT et un CTRL-D en mode programme.

VLIN

Trace une ligne verticale sur l'écran, en graphique basse résolution.

Format :

VLIN $rang_1$, $rang_2$ AT *col*

La ligne est tracée de $rang_1$ à $rang_2$ dans la colonne spécifiée par *col*. La couleur est déterminée par la plus récente instruction COLOR exécutée. Si l'écran est en mode texte, ou si la fenêtre de texte est présente et si la rangée est supérieure à 39, quelques lignes, ou toutes, apparaîtront comme des caractères et non comme des points graphiques. Les caractères sont déterminés par les ordres COLOR exécutés précédemment ; voyez le tableau 8.5 (avec l'ordre PLOT, dans ce chapitre) pour plus de détails.
En Integer Basic, $rang_1$ doit être inférieur ou égal à $rang_2$, faute de quoi le message ***RANGE ERR sera affiché.

VTAB

Positionne le curseur sur la ligne spécifiée de la colonne courante affichée.

Format :

VTAB *rang*

Le curseur est déplace vers le haut ou le bas sur la ligne spécifiée par la valeur de *rang*, sans effacer les caractères affichés. Les rangées sont numérotées de 1 à 24 (du haut vers le bas).

WAIT

Stoppe un programme Applesoft jusqu'à ce qu'une position mémoire particulière atteigne une condition spécifique.

Format :

WAIT *memadr*, $exprnm_1$ [,$exprnm_2$]

WAIT teste tout ou partie du mot de huit bits se trouvant en mémoire à *memadr* et recherche la structure de zéros et de uns spécifiée par la valeur binaire de $exprnm_2$. La valeur binaire de $exprnm_1$ détermine quels bits sont à prendre en considération et quels bits doivent être ignorés. Si un bit particulier de $exprnm_1$ est à 1, le bit correspondant dans

memadr est testé ; s'il est à zéro, WAIT ignore les bits correspondants dans *memadr*. Tant que les bits significatifs (déterminés par $exprnm_1$) de *memadr* sont différents de leurs correspondants dans $exprnm_2$, l'attente se poursuit. Lorsque toutes les paires significatives sont identiques (soit des 0, soit des 1), l'attente est terminée et le programme Applesoft se poursuit.
Si $exprnm_2$ est absent, un zéro est utilisé.
WAIT ne peut être interrompu que par RESET (ou une mise hors tension). Les valeurs des expressions numériques vont de 0 à 255 ; sinon, le message ?ILLEGAL QUANTITY ERROR est affiché. Si la position mémoire spécifiée est supérieure au maximum en mémoire (par exemple, 32767 si vous ne disposez que de 32K de mémoire), ou si vous adressez un périphérique de sortie qui n'est pas en réception, WAIT bloquera le système jusqu'à ce que vous pressiez RESET.
Non disponible en Integer Basic.

WRITE

Spécifie un fichier sur disque vers lequel les ordres PRINT suivants seront sortis.

Format :

WRITE *nomdufichier* [,R*n*] [,B*n*]

Si le fichier spécifié n'est pas déjà ouvert, il sera ouvert (voir OPEN). Les ordres ultérieurs PRINT sauvegardent les données sur le disque jusqu'à ce qu'un ordre disque (ou jusquà ce qu'un caractère CTRL-D seul — code 4 en ASCII) soit affiché. Si le fichier n'est pas sur le disque, le message FILE NOT FOUND apparaît.
Le fichier est écrit en séquentiel si le paramètre R est absent. Dans un fichier séquentiel, le paramètre B spécifie à quel octet (caractère) l'écriture (WRITE) commence. Si B est absent, WRITE commence au premier octet du fichier (octet 0).
Si le paramètre R est présent, le fichier est à accès aléatoire. Le WRITE s'adresse à l'enregistrement spécifié par le paramètre R.
Dans un fichier aléatoire, B indique à quel octet dans l'enregistrement spécifié, l'écriture commence. En l'absence de B, WRITE commence au premier octet de l'enregistrement (octet 0).
Le paramètre B peut servir à écrire à partir d'un point au-delà du dernier caractère déjà dans le fichier (ou l'enregistrement). Cette donnée peut être lue (READ), mais toute tentative de lecture intervenant sur des octets inutilisés provoquera l'affichage du message OUT OF DATA (« plus de données »).
Les nombres accompagnant R et B doivent être des entiers entre 0 et 32767. En leur absence, on supposera qu'il s'agit d'un zéro.
Lorsque WRITE est actif, chaque caractère sorti par Apple II et qui aurait normalement été envoyé à l'écran est dirigé vers le disque, y compris les points d'interrogation produits par INPUT et les éventuels messages d'erreur.
C'est une commande du DOS ; elle exige un PRINT et un CTRL-D en mode programme.
Ne peut être employé en mode immédiat.

XDRAW

Cet ordre Applesoft trace une forme graphique haute résolution sur l'écran et, si elle est employée une seconde fois avec les mêmes paramètres, l'efface.

Format :

XDRAW *exprnm* [AT *colh, rangh*]

La forme numéro *exprnm*, du fichier des formes, est tracée, chaque point dans la couleur complémentaire de la couleur de ce point sur l'écran. Les couleurs 0 et 3 forment une paire complémentaire, de même que 1 et 2, puis 4 et 7, et 5 et 6 (voir le tableau 8.3). L'échelle et la rotation de la forme doivent être établies par SCALE et ROT avant exécution de XDRAW.
Utilisez XDRAW à la place de DRAW pour effacer commodément un tracé existant sur l'écran. Parce que XDRAW est la couleur complémentaire, si vous exécutez deux (ou quatre, six, etc.) XDRAW avec les mêmes paramètres, ce qui est sur l'écran restera inchangé.
Si vous n'avez pas spécifié où commence XDRAW, la forme est tracée à partir du point tracé par la dernière instruction DRAW, XDRAW ou HPLOT. Si vous spécifiez une position, le tracé commence là (*colh, rangh*).
Le numéro de forme, *exprnm,* doit être entre 0 et le nombre de formes dans le fichier (255 au maximum) inclusivement.
Non disponible en Integer Basic.

FONCTIONS (par ordre alphabétique)

Les fonctions d'Apple II, en Basic, sont décrites ci-après par ordre alphabétique. La nomenclature et les abréviations sont données au début de ce chapitre.
La plupart de ces fonctions n'existent qu'en Applesoft. On le précisera.

ABS

Retourne la valeur absolue d'un nombre. C'est la valeur du nombre sans tenir compte du signe.

Format :

ABS *exprnm*

ASC

Retourne le code ASCII d'un caractère spécifié.

Format :

ASC *(expr$)*

Si la chaîne dépasse un caractère, ASC retourne le code ASCII pour le premier caractère uniquement. Le code retourné ne sera pas nécessairement le plus bas code ASCII (dans la gamme 0 à 95) pour ce caractère. Les caractères produits par les codes ASCII entre 96 et 255 dupliquent ceux de la gamme basse sur l'écran.
Cependant, il ne sont pas évalués comme étant le même caractère par les opérateurs relationnels tels que <, > et =. Ils peuvent être traités différemment par les imprimantes et autres dispositifs de sortie. Si le premier caractère de *expr$* est CTRL-@ (code 0 en

ASCII). Le message ?SYNTAX ERROR est généré. Si *expr$* est une chaîne nulle, le message ?ILLEGAL QUANTITY ERROR est produit.
Les codes ASCII sont listés dans l'appendice I.

ATN

Retourne l'arc tangente de l'argument.

Format :

ATN *(exprnm)*

Calcule l'arc tangente, en radians, d'*exprnm.* L'angle retourné est dans la gamme de $-\pi/2$ à $\pi/2$.
Non disponible en Integer Basic.

CHR$

Retourne la valeur chaîne du code ASCII spécifié.

Format :

CHR$ *(exprnm)*

Retourne le caractère représenté par la valeur entière de *exprnm,* interprétée comme un code ASCII. Vous trouverez le tableau des codes ASCII dans l'appendice I. Utilisez cette fonction pour créer des caractères non disponibles sur le clavier, pour commander des périphériques, etc. La valeur de *exprnm* doit se trouver dans la gamme 0 à 255 faute de quoi le message ?ILLEGAL QUANTITY ERROR apparaîtra.
Non disponible en Integer Basic.

COS

Retourne le cosinus d'un angle.

Format :

COS *(exprnm)*

Calcule le cosinus de *exprnm* radians.
Non disponible en Integer Basic.

EXP

Retourne *e* élevé à une puissance.

Format :

EXP *(exprnm)*

Calcule *e* (base des logarithmes népériens, 2.71828283) élevée à la puissance *exprnm.*
Non disponible en Integer Basic.

FN

Appelle une fonction définie préalablement par l'utilisateur.

Format :

FN *varnm (exprnm)*

varnm est le nom de la fonction. La valeur de *exprnm* est assignée partout où la variable factice intervient dans la définition de la fonction, et l'expression résultante est évaluée. Voyez DEF FN dans la partie instructions de ce chapitre.
Une fonction peut ne pas être récursive ; par exemple, *exprnm* peut ne pas se référer à FN *varnm* ni à aucune autre fonction que se réfère à FN *varnm*.
Si vous tentez d'utiliser FN *varnm* avant l'instruction DEF FN *varnm,* vous recevrez le message ?UNDEF'D FUNCTION ERROR (« erreur de fonction non définie »).
Non disponible en Integer Basic.

FRE

Retourne le nombre d'octets en mémoire encore disponible pour le programme Applesoft.

Format :

FRE *(exprnm)*

La mémoire dont vous pouvez disposer se trouve en-dessous de la zone réservée aux chaînes et au-dessus de la zone des tableaux. S'il reste plus de 32767 octets disponibles, FRE retourne un nombre négatif. Ajoutez 65536 à ce nombre pour trouver le volume mémoire disponible.
FRE remet aussi à zéro les chaînes anciennes dans la zone chaînes. Lorsqu'une chaîne change de valeur au cours d'un programme, l'ancienne valeur reste en mémoire et la nouvelle s'ajoute au contenu de la zone réservée aux chaînes. Cela pourra empiéter sur la mémoire que vous utilisez pour autre chose. Pour éviter ce problème, exécutez périodiquement un ordre tel que A = FRE(0) dans un programme qui emploie intensivement des chaînes.
La valeur de *exprnm* n'est pas utilisée par FRE, mais elle pourra déclencher une erreur si elle est illégale.
Non disponible en Integer Basic.

INT

Retourne la partie entière d'un nombre.

Format :

INT *(exprnm)*

Retourne le plus grand entier inférieur ou égal à la valeur de *exprnm.*
Non disponible en Integer Basic.

LEFT$

Retourne les caractères les plus à gauche d'une chaîne.

Format :

LEFT$ *(expr$, exprnm)*

Retourne les *exprnm* caractères les plus à gauche de *expr$*. L'expression *exprnm* doit être comprise entre 1 et 255, et *expr$* ne peut posséder plus de 255 caractères. Si *exprnm* est plus grand que la longueur de *expr$,* la chaîne entière est retournée.
Non disponible en Integer Basic.

LEN

Retourne la longueur d'une chaîne.

Format :

LEN *(expr$)*

Compte le nombre de caractère de *expr$,* y inclus les espaces et les caractères sans impression. Si *expr$* est supérieur à 255 caractères (ce qui est possible uniquement si *expr$* est une expression chaîne incluant des concaténations), le message ?STRING TOO LONG ERROR (« erreur de chaîne trop longue ») est créé.

LOG

Retourne le logarithme naturel d'un nombre.

Format :

LOG *(exprnm)*

Calcule le logarithme naturel de *exprnm.* Retourne ?ILLEGAL QUANTITY ERROR si *exprnm* est zéro ou négatif.
Non disponible en Integer Basic.

MID$

Retourne la partie centrale spécifiée d'une chaîne.

Format :

MID$ *(expr$, exprnm$_1$ [,exprnm$_2$])*

Retourne *exprnm$_2$* caractères de *expr$,* à partir du caractère *exprnm$_1$*. Si *exprnm$_2$* est absent, MID$ retourne la partie d'*expr$* à partir du caractère *exprnm$_1$* et jusqu'au dernier caractère. Si la longueur d'*expr$* est inférieure à *exprnm$_1$*, une chaîne nulle est retournée. S'il y a moins de *exprnm$_2$* caractères dans *expr$* après *exprnm$_1$*, le résultat est le même que si *exprnm$_2$* était absente. *expr$* ne doit pas excéder 255 caractères, et *exprnm$_1$* tout comme *exprnm$_2$* doivent être compris entre 1 et 255.
Non disponible en Integer Basic.

PDL

Retourne la valeur courante de la manette de jeu spécifiée (« paddle »).

Format :

PDL *(exprnm)*

La valeur retournée est un entier entre 0 et 255, basé sur la rotation de la manette de jeu numéro *exprnm,* ou la résistance d'un élément connecté à la prise *exprnm* de commande de jeux. Les commandes de jeux sont numérotées de 0 à 3. Si le numéro de manette est inférieur à 0 ou supérieur à 255, le message ?ILLEGAL QUANTITY ERROR est émis. Si le numéro est compris entre 4 et 255, PDL retourne un nombre quelque peu imprévisible entre 0 et 255 et pourra provoquer des effets variés, tels qu'un « clic » dans le haut-parleur, ou un décalage en mode graphique.
Si deux instructions PDL sont exécutées consécutivement, ou presque consécutivement, la seconde valeur pourra être affectée par la première. Assurez-vous que quelques instructions séparent deux PDL (une boucle vide FOR-NEXT fera l'affaire).

PEEK

Retourne le contenu d'une position mémoire.

Format :

PEEK *(memadr)*

La valeur retournée est l'équivalent décimal des huit bits se trouvant dans la position mémoire *memadr.* L'appendice E donne la liste de quelques positions importantes.

POS

Retourne la colonne sur laquelle se trouve le curseur.

Format :

POS *(exprnm)*

L'expression est factice ; elle n'est pas utilisée et de ce fait, peut prendre n'importe quelle valeur légale.
POS retourne une valeur entre 0 et 39. Les positions commencent à gauche (0).
Non disponible en Integer Basic.

RIGHT$

Retourne les caractères les plus à droite d'une chaîne.

Format :

RIGHT$ *(expr$, exprnm)*

Retourne les *exprnm* caractères les plus à droite de *expr$.* La valeur de *exprnm* doit être comprise entre 1 et 255, et *expr$* ne doit pas dépasser 255 caractères. Si *exprnm* est plus grande que la longueur de *expr$,* la chaîne entière est retournée.
Non disponible en Integer Basic.

RND

Retourne un nombre aléatoire.

Format :

RND *(exprnm)*

Retourne un nombre aléatoire dans une gamme dépendant de la valeur de *exprnm* et de la version du Basic.
En Integer Basic, RND retourne un nombre aléatoire entier entre 0 et la valeur de *exprnm,* exclusif de *exprnm* mais avec 0 inclusif. Ainsi, RND(1) retourne toujours 0, et RND(−2) produit un mélange moitié-moitié de 0 et −1. Essayer d'employer RND(0) provoque l'affichage du message ***>32767 ERR.
En Applesoft, RND retourne toujours un nombre réel plus grand que, ou égal à 0, et inférieur à 1. La valeur retournée peut être de l'un des trois types, selon le signe de *exprnm.*
Si *exprnm* est positive, RND retourne une valeur différente à chaque usage, sauf si une séquence répétitive a commencé.
Une séquence répétitive commence lorsque RND est utilisé avec une *exprnm* négative. Chaque valeur négative particulière démarre toujours la même séquence ; des arguments positifs ultérieurs retourneront une séquence répétitive de nombres aléatoires. Une séquence répétitive différente démarre pour chaque valeur négative différente de *exprnm.* Cette caractéristique est utile pour tester et débuguer des programmes faisant intervenir RND.
Si *exprnm* est 0 en Applesoft, RND retourne le nombre positif le plus récemment créé (il n'est pas affecté par CLEAR ou NEW).

SCRN

Retourne le code de couleur du point graphique basse résolution dont les coordonnées ont été spécifiées.

Format :

SCRN *(col, rang)*

Si l'écran est en mode texte, ou si la fenêtre de texte est présente et que le point spécifié se trouve à l'intérieur, SCRN retourne le code de couleur de la moitié du caractère. Le code couleur de la moitié supérieure du caractère est retourné si la rangée *rang* est paire, celle de la moitié inférieure si *rang* est impaire. Le code ASCII du caractère à la position *(a,b)* (avec *a* dans la gamme 0 à 39 et *b* de 0 à 23) est retourné par l'expression SCRN (*a*,2∗*b*) +16∗ SCRN (*a*,2∗*b*+1). Ainsi, le caractère lui-même est retourné par CHR$(*n*), où *n* est la valeur fournie par l'expression ci-dessus.
Si *col* est dans la gamme de 0 à 39, SCRN retourne le code de couleur du point graphique *(col, rang).* Si *col* est entre 40 et 47 et *rang* entre 0 et 31, SCRN retourne le code de couleur du point graphique (*col*−40, *rang*+16). Si *col* est entre 40 et 43 et *rang* entre 32 et 47, SCRN retourne un nombre sans relation aucune avec ce qui se trouve sur l'écran.
Si SCRN est employé alors que l'écran est en mode haute résolution, le nombre retourné se rapporte à la zone graphique basse résolution de la mémoire plutôt qu'à l'affichage haute résolution.
SCRN n'est reconnu comme un mot réservé que si le caractère suivant, autre qu'un espace, est laissé entre parenthèses.

SGN
Détermine si un nombre est positif, négatif ou nul.

Format :

SGN *(exprnm)*

La fonction SGN retourne +1 si *exprnm* est positive, −1 si elle est négative et 0 si elle est nulle.

SIN
Retourne le sinus de l'angle.

Format :

SIN *(exprnm)*

Calcule le sinus de *exprnm* radians.
Non disponible en Integer Basic.

SPC
Déplace le curseur à droite du nombre spécifié de positions.

Format :

SPC *(exprnm)*

La fonction SPC intervient dans les instructions PRINT pour afficher *exprnm* espaces. De ce fait, tous les caractères que croise le curseur sont effacés.
La fonction SPC déplace le curseur à droite, quelle que soit sa position (colonne) de départ. C'est la différence avec TAB, qui déplace le curseur d'un certain nombre de colonnes mesuré à partir de la colonne la plus à gauche.
Non disponible en Integer Basic.

SQR
Retourne la racine carrée d'un nombre positif.

Format :

SQR *(exprnm)*

Une valeur négative de *exprnm* déclenche un message ?ILLEGAL QUANTITY ERROR.
SQR *(exprnm)* est plus rapide que *(exprnm)* ∧ (.5).
Non disponible en Integer Basic.

STR$
Convertit une valeur numérique en chaîne.

Format :

STR$ *(exprnm)*

La valeur de *exprnm* est convertie en chaîne. Les caractères de la chaîne sont les mêmes que ceux qui auraient été affichés par une instruction PRINT *exprnm*. Ainsi, STR$(2/3) = " .666666667 " et STR$ (2468013579) = " 2.46801358E+09 ". Si *exprnm* excède les limites des nombres réels, le message ?OVERFLOW ERROR apparaît.
Non disponible en Integer Basic.

TAB
Déplace le curseur à droite sur la colonne de position spécifiée.

Format :

TAB *(exprnm)*

L'utilisation de TAB dans une instruction PRINT déplace le curseur sur la colonne *exprnm*, si celle-ci est à la droite de la position courante du curseur. Le curseur ne bouge pas si *exprnm* ne vise pas une colonne à sa droite. TAB affiche des espaces en déplaçant le curseur à droite, effaçant ainsi ce qui se trouvait sur son passage.
Pour TAB, les colonnes sont numérotées de 1 à 255. Si *exprnm* est supérieur à la largeur du dispositif de sortie (40, pour l'écran), il ramène le curseur à la ligne inférieure et reprend le comptage à partir de la marge gauche. Si la valeur de TAB est 0, le curseur va à la colonne 256. Une valeur de TAB hors de la gamme 0 à 255 déclenche le message ?ILLEGAL QUANTITY ERROR.
Voyez également HTAB (Applesoft) et TAB (Integer Basic) dans la section « instructions » de ce chapitre.
Non disponible en Integer Basic.

TAN
Retourne la tangente d'un angle.

Format :

TAN *(exprnm)*

Calcule la tangente de *exprnm* radians.
Non disponible en Integer Basic.

USR
Branche sur un sous-programme en langage machine, en passant les valeurs par l'accumulateur.

Format :

USR *exprnm*

Le sous-programme part de la position mémoire 10 (0A en hexadécimal). Les positions 10 à 12 (0A à 0C en hexadécimal) doivent contenir un ordre de saut (JMP) en langage assembleur qui se branche au début du sous-programme. Puisque USR est une fonction, il retourne une valeur numérique réelle. Ce qui se trouve dans l'accumulateur lorsque le sous-programme exécute un RTS (retour au programme Applesoft) est la valeur renvoyée.

Le moniteur d'Apple II contient plusieurs sous-programmes utiles en langage machine. Voyez leur liste dans l'appendice D.
Voyez aussi CALL, décrit dans les « Instructions », disponible également en Integer Basic.
Non disponible en Integer Basic.

VAL

Convertit une chaîne en une valeur numérique.

Format :

VAL *(expr$)*

Retourne la valeur numérique représentée par *expr$*. Si le premier caractère de *expr$* n'est pas un caractère numérique, un zéro est renvoyé. Sinon, *expr$* est examiné caractère par caractère jusqu'à ce qu'un caractère inacceptable soit rencontré. Les caractères acceptables sont les digits de 0 à 9, les espaces, le point décimal, un signe plus ou moins en tête, et dans le contexte de la notation scientifique, un signe plus ou moins additionnel, un point décimal additionnel, et la lettre E.
Si *expr$* est une chaîne impliquant une concaténation menant à plus de 255 caractères, le message ?STRING TOO LONG ERROR (« erreur de chaîne trop longue ») survient. Si la valeur numérique de *expr$* excède les limites des nombres réels, le message ?OVERFLOW ERROR (« erreur de dépassement ») est affiché.
Non disponible en Integer Basic.

APPENDICES

APPENDICE A

FONCTIONS NUMÉRIQUES COMPLÉMENTAIRES

Bien que la liste suivante de fonctions numériques complémentaires ne soit pas complète, elle propose les plus fréquentes. Certaines valeurs de *x* invalideront des fonctions (par exemple, si COS*(x)* = 0, la sécante SEC*(x)* n'est pas réelle), aussi votre programme devra-t-il s'en assurer.
Aucune de ces fonctions ne s'applique à l'Integer Basic.

ARCCOS*(x)* = − ARN*(x*/SQR(− *x***x* + 1) + 1.5707633
Retourne le cosinus inverse de *x*(ABS*(x)* < 1).

ARCCOT*(x)* = − ATN*(x)* + 1.5707633
Retourne la cotangente inverse de *x*.

ARCCOSH*(x)* = LOG*(x* + SQR*(x***x* − 1))
Retourne le cosinus hyperbolique inverse de *x(x* > = 1).

ARCCOTH*(x)* = LOG((*x* + 1)/(*x* − 1))/2
Retourne la cotengante hyperbolique inverse de *x*(ABS(*x* > 1)).

ARCCSC*(x)* = ATN(1/SQR(*x***x* − 1)) + (SGN*(x)* − 1)*1.5707633
Retourne la cosécante inverse de *x*(ABS*(x)* > 1).

ARCCSCH*(x)* = LOG((SGN*(x)**SQR(*x***x* + 1) + 1)/*x)*
Retourne la cosécante hyperbolique inverse de *x*(*x* > 0).

ARCSEC*(x)* = ATN(SQR)(*x***x* − *1)) + (SGN(x)* − 1)*1.5707633
Retourne la sécante inverse de *x*(ABS*(x)* > = 1).

ARCSECH*(x)* = LOG((SQR(− *x***x* + 1) + 1)/*x*)
Retourne la sécante hyperbolique inverse de *x*(0 < *x* < = 1).

ARCSIN*(x)* = ATN(*x*/SQR(− *x***x* + 1))
Retourne le sinus inverse de *x*(ABS*(x)* < 1).

ARCSINH(*x*) = LOG(*x* + SQR(*x***x* + 1))
Retourne le sinus hyperbolique inverse de *x*.

ARCTANH(*x*) = LOG((1 + *x*)/(1 − *x*))/2
Retourne la tangente hyperbolique inverse de *x*(ABS(*x*) < 1).

COSH(*x*) = (EXP(*x*) + EXP(−*x*))/2
Retourne le cosinus hyperbolique de *x*.

COT(*x*) = 1/TAN(*x*)
Retourne la cotangente de *x*(*x* < > 0).

COTH(*x*) = EXP(−*x*)/(EXP(*x*) − EXP(−*x*))*2 + 1
Retourne la cotangente hyperbolique de *x*(*x* < > 0).

CSC(*x*) = 1/SIN(*x*)
Retourne la cosécante de *x*(*x* < > 0).

CSCH(*x*) = 2(EXP(*x*) − EXP(−*x*))
Retourne la cosécante hyperbolique de *x*(*x* < > 0).

$LOG_a(x)$ = LOG(*x*)/LOG(*a*)
Retourne le logarithme à base *a* de *x* (*a* > 0, *x* > 0).

$LOG_{10}(x)$ = LOG(*x*)/2.30258509
Retourne le logarithme à base 10 de *x*(*x* > 0).

$MOD_a(x)$ = − INT ((*x*/*a*))**a* + .05)*SGN(*x*/*a*)

Retourne *x* modulo *a* : le reste de la division de *x* par *A*.

SEC(*x*) = 1/COS(*x*)
Retourne la sécante de *x*(*x* < > π/2).

SECH(*x*) = 2/(EXP(*x*) + EXP(−*x*))
Retourne la sécante hyperbolique de *x*.

SINH(*x*) = (EXP(*x*) − EXP(−*x*))/2
Retourne le sinus hyperbolique de *x*.

TANH(*x*) = − EXP(−*x*)/EXP(*x*) + EXP(−*x*))*2 + 1
Retourne la tangente hyperbolique de *x*.

APPENDICE B

COMMANDES D'ÉDITION

Cet appendice résume les fonctions d'édition avec le clavier d'Apple II.

→ Déplace le curseur vers l'avant de la ligne affichée. Chaque caractère sur lequel il passe est recopié en mémoire comme s'il avait été frappé au clavier. N'altère pas l'affichage sur l'écran.

← Déplace le curseur en arrière sur la ligne affichée, effaçant les caractères sur lequel il passe de la mémoire mais non de l'écran.

REPT Provoque la répétition d'un caractère frappé aussi longtemps que les deux touches restent enfoncées. La touche REPT doit être enfoncée *après* l'autre touche.

CTRL-X Apple II ignore la ligne courante affichée et déplace le curseur à la marge gauche de la ligne suivante.

Séquences de ESC

Les sept commandes d'édition suivantes sont des séquences de deux touches. Pour chaque cas, pressez ESC, relâchez-le, puis pressez la seconde touche.

ESC-A Déplace le curseur d'une position vers la droite. N'altère ni l'affichage, ni la mémoire.

ESC-B Déplace le curseur d'une position sur la gauche. N'altère ni l'affichage, ni la mémoire.

ESC-C Déplace le curseur une ligne en-dessous. N'altère ni l'affichage, ni la mémoire.

ESC-D Déplace le curseur une ligne au-dessus. N'altère ni l'affichage, ni la mémoire.

ESC-E Efface tous les caractères, du curseur à la fin de la ligne affichée.

ESC-F Efface tous les caractères, du curseur à la fin de la ligne affichée.

ESC-@ Efface l'écran et ramène le curseur à l'angle supérieur gauche.

Commandes de l'éditeur

Les quatre commandes suivantes d'édition demandent la présence du moniteur autostart. Elles ne sont effectives qu'en mode *édition*. Entrez dans ce mode en pressant ESC ; pour le quitter, pressez toute autre touche que I, J, K, M, REPT, CTRL ou SHIFT.

I — Déplace le curseur d'une ligne vers le haut sans quitter le mode éditeur.

J — Déplace le curseur d'une position vers la gauche sans quitter le mode éditeur.

K — Déplace le curseur d'une position vers la droite sans quitter le mode éditeur.

M — Déplace le curseur d'une ligne vers le bas sans quitter le mode éditeur.

APPENDICE C

MESSAGES D'ERREURS

Les messages d'erreurs sont groupés en trois catégories : Integer Basic, Applesoft, et DOS, et sont listés à chaque fois par ordre alphabétique (anglais).
Les messages du DOS et beaucoup de messages d'Applesoft disposent de codes d'erreurs associés. Après une erreur ayant provoqué un branchement ONERR GOTO, le code de cette erreur pourra être trouvé en position mémoire 222. Le tableau C.1, à la fin de cet appendice, liste les messages d'erreurs par ordre de leurs numéros de code.

MESSAGES D'ERREURS EN INTEGER BASIC

***** > 255 ERR**
Une valeur devant être comprise entre 0 et 255 et hors de cette gamme.

***** > 32767 ERR**
Une valeur supérieure à 32767 ou inférieure à −32767 a été entrée ou calculée.

***** 16 FORS ERR**
Plus de 16 boucles FOR sont actives.

***** 16 GOSUBS ERR**
16 instructions GOSUB ont été exécutées de plus que de RETURN.

***** BAD BRANCH ERR** *(erreur de mauvais branchement)*
On a tenté un branchement à un numéro de ligne inexistant.

***** BAD NEXT ERR** *(erreur de mauvais NEXT)*
Un NEXT sans FOR a été proposé.

***** BAD RETURN ERR** *(erreur de mauvais RETURN))*
Il y a plus de RETURN que de GOSUB.

***** DIM ERR** *(erreur de dimensionnement)*
Le même tableau a été dimensionné plus d'une fois.

***** MEM FULL ERR** *(erreur de mémoire pleine)*
On a besoin de plus de mémoire qu'il n'y en a de disponible.

***** NO END ERR** *(erreur d'absence de fin)*
La dernière instruction exécutée d'un programme n'était pas END.

***** RANGE ERR** *(erreur de gamme)*
On a référencé un tableau avec un indice inférieur à zéro ou supérieur à la dimension du tableau ; ou encore, un argument dans des instructions HLIN, VLIN, PLOT, TAB ou VTAB était hors de la gamme prescrite.

***** RETYPE LINE** *(refrappez la ligne)*
Erreur sur la réponse à INPUT. Un message de diagnostic est d'abord affiché, puis cette directive.

***** STRING ERR** *(erreur de chaîne)*
Une opération illégale sur une chaîne a été exécutée.

***** STRING OVFL ERR** *(erreur de passement de chaîne)*
Une chaîne s'est vue attribuer plus de caractères que sa dimension ne le permet.

***** SYNTAX ERR** *(erreur de syntaxe)*
Erreur de syntaxe : orthographe, ponctuation, séquence ; ou toute erreur non couverte par les autres messages.

***** TOO LONG ERR** *(erreur de trop grande longueur)*
Plus de 12 parenthèses ont été emboîtées, ou plus de 128 caractères ont été entrés sur une ligne.

MESSAGES D'ERREURS EN APPLESOFT

?BAD SUBSCRIPT ERROR *(erreur de mauvais indiçage)*
On a référencé un tableau avec un mauvais nombre pour l'indice, ou avec des indices excédants ses dimensions. Code 107.

?CAN'T CONTINUE ERROR *(erreur d'impossibilité de continuer)*
On a essayé de *continuer* le programme (avec la commande CONT) alors qu'aucun programme n'existe, qu'une erreur fatale s'est produite, ou qu'une modification au programme a été introduite.

?DIVISION BY ZERO ERROR *(erreur de division par zéro)*
On a essayé de diviser par une expression égale à zéro. Code 133.

?FORMULA TOO COMPLEX ERROR *(erreur de formule trop complexe)*
Plus de deux instructions de la forme IF *chaîne* THEN ont été exécutées. Code 191.

?ILLEGAL DIRECT ERROR *(erreur d'instruction directe illégale)*
Une commande INPUT, DEF FN ou GET a été introduite en mode direct.

?ILLEGAL QUANTITY ERROR *(erreur de quantité illégale)*
Une valeur numérique excède la gamme acceptable pour une fonction numérique, chaîne, un ordre graphique, etc. Code 53.

?NEXT WITHOUT FOR ERROR *(erreur de NEXT sans FOR)*
Un NEXT sans FOR a été exécuté. Un NEXT sans nom de variable ne produit cette erreur que s'il n'y a pas de FOR actif. Code 0.

?OUT OF DATA ERROR *(erreur de message de données)*
Davantage d'éléments DATA ont été lus qu'il n'y en avait. Code 42.

?OUT OF MEMORY ERROR *(erreur de sortie de mémoire)*
Peut être causé par : un programme trop long, trop de variables, plus de 10 niveaux de boucles FOR emboîtées, plus de 24 niveaux de sous-programmes emboîtés, plus de 36 niveaux de parenthèses emboîtées, LOMEM: établi trop haut, ou HIMEM: trop bas. Code 77.

?OVERFLOW ERROR *(erreur de dépassement)*
Un nombre trop petit ou trop grand a été entré ou calculé. La gamme autorisée va d'environ $-1.7E+38$ à $1.7E+38$. Code 69.

?REDIM'D ARRAY ERROR *(erreur de redimensionnement de tableau)*
Une instruction DIM a été exécutée pour un tableau déjà dimensionné. La plupart des erreurs surviennent lorsque le tableau a été dimensionné par défaut. Code 120.

?RETURN WITHOUT GOSUB ERROR *(erreur de RETURN sans GOSUB)*
Plus de RETURN que de GOSUB ont été exécutés. Code 22.

?STRING TOO LONG ERROR *(erreur de chaîne trop longue)*
On a essayé de concaténer des chaînes pour plus de 255 caractères. Code 176.

?SYNTAX ERROR *(erreur de syntaxe)*
Une faute d'orthographe, de ponctuation, de séquence, ou toute erreur non couverte par les autres messages. Code 16.

?TYPE MISMATCH ERROR *(erreur de désaccord de type)*
On a utilisé une expression ou une variable à la place d'une chaîne, ou vice-versa. Se manifeste aussi lorsque les deux parties d'une instruction d'affectation ne s'accordent pas. Code 163.

?UNDEF'D FUNCTION ERROR *(erreur de fonction non définie)*
On a fait appel à une fonction définie par l'utilisation qui n'a jamais été définie. Code 224.

?UNDEF'D STATEMENT ERROR *(erreur d'instruction non définie)*
Branchement à un numéro de ligne inexistant. Code 90

MESSAGES D'ERREURS DU DOS

DISK FULL *(disque plein)*

On a tenté de ranger plus d'informations sur le disque qu'il n'en peut contenir. Avec un disque plein, ce message pourra remplacer des messages plus appropriés (par exemple, FILE NOT FOUND). Code 9.

END OF DATA *(fin des données)*

On a essayé de lire une portion de fichier de texte qui n'a jamais été écrite. Code 5.

FILE LOCKED *(fichier verrouillé)*

On a essayé d'employer SAVE, BSAVE, WRITE, DELETE ou RENAME avec un fichier verrouillé. Code 10.

FILE NOT FOUND *(fichier non trouvé)*

On a référencé un fichier absent du disque. Cette erreur ne se produit que si la commande du DOS ne crée pas le fichier lorsqu'il n'existe pas. Code 6.

FILE TYPE MISMATCH *(désaccord de type de fichier)*

Une commande du DOS a référencé un fichier qui n'est pas du type requis. Les commandes LOAD, RUN et SAVE ne peuvent être employées qu'avec des fichiers programmes. La commande CHAIN ne peut être utilisée qu'avec des fichiers programmés en Integer Basic. Les commandes OPEN, READ, WRITE, APPEND, POSITION et EXEC ne peuvent être employées qu'avec des fichiers de texte. Enfin, BLOAD, BSAVE et BRUN ne peuvent être utilisés qu'avec des fichiers binaires. Code 13.

I/O ERROR *(erreur d'entrée-sortie)*

Une tentative infructueuse d'écrire ou lire un disque. Les causes usuelles sont : la porte de l'unité à disquettes est ouverte, le disque n'a pas été initialisé, il n'y a pas de disque dans l'unité, le disque est mauvais Code 8.

LANGUAGE NOT AVAILABLE *(langage non disponible)*

On a essayé de changer de langage avec FP ou INT alors qu'il ne se trouvait ni en ROM ni sur le disque, ou on a tenté de charger ou lancer un programme alors que son langage n'était pas disponible. Code 1.

NO BUFFERS AVAILABLE *(pas de buffers disponibles)*

On a demandé un autre buffer, mais ils étaient tous utilisés. Code 12.

NOT DIRECT COMMANDE *(commande non en direct)*

Les commandes suivantes du DOS ne peuvent intervenir que dans des instructions PRINT en mode programmé : APPEND, OPEN, POSITION, READ, et WRITE. Code 15.

PROGRAMM TOO LARGE *(programme trop long)*

Une commande du DOS a tenté de charger un fichier sur disque dans la mémoire d'Apple II, mais celle-ci était trop petite pour le recevoir. Code 14.

RANGE ERROR *(erreur de gamme)*

Un paramètre d'une commande du DOS excède la gamme spécifiée. Par exemple, le paramètre d'unité à disquettes doit être 1 ou 2. Codes 2 ou 3.

SYNTAX ERROR *(erreur de syntaxe)*

Une commande du DOS avec une faute d'orthographe, de ponctuation ou de séquence. Code 11.

VOLUME MISMATCH *(désaccord de volume)*

Le paramètre de volume (V) d'une commande du DOS ne correspond pas au numéro du volume du disque adressé. Code 7.

WRITE PROTECTED *(écriture protégée)*

On a tenté d'utiliser SAVE, BSAVE, ou WRITE sur un disque protégé en écriture. Code 4.

Tableau C.1. — Codes des erreurs.

PEEK(222)	Appel anglais	Définition	Langage
0	NEXT without FOR	Next sans FOR	Applesoft
1	Language not available	Langage non disponible	DOS
2 ou 3	Range error	Erreur de gamme	DOS
4	Write protected	Ecriture protégée	DOS
5	End of data	Fin des données	DOS
6	File not found	Fichier non trouvé	DOS
7	Volume mismatch	Désaccord de volumes	DOS
8	I/O error	Erreur d'entrée-sortie	DOS
9	Disk full	Disque plein	DOS
10	File locked	Fichier verrouillé	DOS
11	Syntax error	Erreur de syntaxe	DOS
12	No buffers available	Pas de buffers disponibles	DOS
13	File type mismatch	Désaccord de types de fichiers	DOS
14	Programm too large	Programme trop long	DOS
15	Not direct command	Pas de commande directe	DOS
16	Syntax error	Erreur de syntaxe	DOS
22	RETURN without GOSUB	RETURN sans GOSUB	Applesoft
42	Out of DATA	Plus de données	Applesoft
53	Illegal quantity	Quantité illégale	Applesoft
69	Overflow	Dépassement	Applesoft
77	Out of memory	Plus de mémoire	Applesoft
90	Undefined statement	Instruction non définie	Applesoft
107	Bad subscript	Mauvais indice	Applesoft
120	Redimensioned array	Tableau redimensionné	Applesoft
133	Division by zero	Division par zéro	Applesoft
163	Type mismatch	Désaccord de types	Applesoft
176	String too long	Chaîne trop longue	Applesoft
191	Formula too complex	Formule trop complexe	Applesoft
224	Undefined function	Fonction non définie	Applesoft
254	Bad response to an INPUT	Mauvaise réponse à un INPUT	Applesoft
255	CTRL-C has been struck	On a frappé CTRL-C	Applesoft

APPENDICE D

SOUS-PROGRAMMES INTRINSÈQUES

Les deux tableaux suivants donnent la liste de sous-programmes utiles, en langage machine, disponibles sur l'Apple II. Le tableau D.1 les liste par fonctions générales, sans fournir d'information complète. Recherchez le point d'entrée dans le tableau D.1 et reportez-vous à ce point dans la première colonne du tableau D.2 pour plus de détails, les registres affectés, etc.

Le tableau D.2 liste ces sous-programmes par ordre de points d'entrée. La troisième colonne montre les registres qui, éventuellement, doivent contenir des valeurs spécifiques avant exécution du sous-programme. La quatrième colonne montre quels registres sont affectés par l'exécution du sous-programme.

La plupart de ces sous-programmes disposent d'une commande équivalente en Basic, ou peuvent être appelés à partir du Basic par une simple instruction CALL. Les équivalents sont donnés dans le tableau D.2. Quelques commandes Basic listées ne sont disponibles qu'en Applesoft ; elles sont marquées d'un A.

Certains de ces sous-programmes, cependant, n'ont pas d'équivalent en aucune version du Basic, et ne peuvent être appeler par un simple CALL car un, ou plusieurs registres doivent être chargés avec des valeurs spécifiques préalablement à l'appel du sous-programme. Les techniques requises sont différentes avec les deux Basic.

L'Integer Basic fournit une solution remarquablement simple. Exécutez d'abord un CALL – 182 pour placer les valeurs courantes des registres en mémoire vive RAM. Puis, rangez ces valeurs avec POKE à la position mémoire 69 pour le registre A, 70 pour X et 71 pour le registre Y. Faites un CALL – 193 pour restituer ces valeurs aux registres, et appelez la position à laquelle commence le sous-programme à exécuter.

Cette procédure ne convient pas à l'Applesoft. Vous devez, ici, écrire et exécuter un sous-programme en langage machine qui charge les registres avec les valeurs désirées, puis exécuter une instruction de saut JSR, en langage assembleur au point d'entrée du sous-programme.

Tableau D.1. — Sous-programmes intrinsèques par fonction.

	Fonction	Point d'entrée
Graphique basse résolution	Trace un point graphique basse résolution.	$F800
	Trace une ligne horizontale basse résolution.	$F819
	Trace une ligne verticale basse résolution.	$F828
	Efface et remet au noir les 48 lignes graphiques basse résolution (si on est en mode texte, établit des " @ " inversés).	$F832
	Efface et remet au noir les 40 lignes supérieures basse résolution (ou place des " @ " inversés).	$F836
	Incrémente de 3 la couleur graphique courante basse résolution.	$F85F
	Etablit la couleur graphique basse résolution.	$F864
	Lit la couleur d'un point graphique basse résolution.	$F871
	Etablit le mode graphique basse résolution, efface l'écran, et établit une fenêtre de texte de quatre lignes.	$FB40
Entrées	Attend une touche alors que le curseur flashe, et amorce le générateur de nombres aléatoires aux positions $4E et $4F.	$FD1B
	Comme ci-dessus, sauf que les codes d'échappement sont aussi autorisés.	$FD35
	Envoie le retour chariot à l'écran, puis permet l'entrée d'une ligne entière jusqu'à 256 caractères.	$FD67
Sorties	Envoie trois blancs au périphérique sélectionné.	$F948
	Envoie de un à 256 blancs au périphérique sélectionné.	$F94A
	Envoie un retour chariot et un passage à la ligne suivante à l'écran.	$FC62
	Sort un caractère sur le périphérique sélectionné.	$FDED
	Sort un caractère pour la fenêtre de texte.	$FDF0
Sortie sonnette	Envoie le caractère sonnette (BELL) (code 7 en ASCII) au périphérique courant.	$FBD9
	Emet un « bip » de 0,1 seconde via le haut-parleur interne.	$FBE4
	Affiche un message ERR et émet un « bip ».	$FF2D
	Emet un « bip » via le haut-parleur interne.	$FF3A

Tableau D.1. — Sous-programmes intrinsèques par fonction (suite).

	Fonction	Point d'entrée
Fenêtre de texte	Etablit l'écran sur 24 rangées et 40 colonnes.	$FB2F
	Déroule la fenêtre de texte de une ligne.	$FC70
Commande du curseur	Envoie un caractère de retour (marche arrière) à l'écran, remettant à jour la position du curseur.	$FC10
	Déplace le curseur à la ligne supérieure. S'il est déjà en haut de l'écran, il ne bouge pas.	$FC1A
	Déplace le curseur à la ligne inférieure sans changer de colonne.	
	Déroule la fenêtre de texte si le curseur est en bas de l'écran.	$FC66
Effacement de l'écran	Efface la fenêtre de texte à partir de la position courante du curseur et vers l'angle droit inférieur de l'écran.	$FC42
	Efface la fenêtre de texte à partir de coordonnées passées aux registres, vers l'angle droit inférieur de l'écran.	$FC46
	Efface totalement la fenêtre de texte et ramène le curseur à l'angle supérieur gauche.	$FC58
	Efface le texte à partir de la position courante du curseur et jusqu'à la fin de la ligne.	$FC9C
Mode vidéo	Etablit le mode vidéo inversé (noir sur blanc).	$FE80
	Etablit le mode vidéo normal (blanc sur noir).	$FE84
Affichage du contenu des registres	Affiche les contenus des registres X et Y (dans le format YYXX) sur le périphérique sélectionné.	$F940
	Affiche les contenus des registres A et X (format AAXX) sur le périphérique sélectionné.	$F941
	Affiche le contenu de X sur le périphérique sélectionné.	$F944
	Affiche le contenu de A sur le périphérique sélectionné.	$FDDA
Dépassement des contenus des registres	Restaure le contenu des registres (valide uniquement si le programme intrinsèque à $FF4A a d'abord été exécuté).	$FF3F
	Sauvegarde le contenu des registres aux positions réservées en page zéro.	$FF4A
Divers	Lit l'état d'une manette pour jeux.	$FB1E
	Exécute une boucle de temporisation.	$FCA8
	Retourne au Basic, éliminant le programme et les variables de la mémoire.	$FEB0
	Point d'entrée du moniteur.	$FF69

Tableau D.2. — Sous-programmes intrinsèques par points d'entrée.

Point d'entrée	Fonction	Registres à charger avant appel	Registres affectés	Equivalent Basic
$F800	Trace un point graphique basse résolution en page 1.	Rangée dans A, colonne dans Y	Aucun	PLOT
$F819	Trace une ligne horizontale basse résolution	Rangée dans A, col. gauche dans Y, col. dr. en mémoire à 44.	A, Y	HLIN
$F828	Trace une verticale en basse résolution.	Col. dans Y, rangée sup. dans A, rangée inf. en mémoire à 45.	Aucun	VLIN
$F832	Efface et ramène les 48 lignes graphiques basse résolution (si en mode texte, établit des @ partout).	Aucun.	A, Y	CALL – 1998
$F836	Efface les rangées graphiques basse résolution, sans toucher à la fenêtre de texte.	Aucun.	A, Y	GR (voir $FB40)
$F85F	Incrémente de trois la couleur graphique basse résolution.	Aucun.	A	CALL – 1985
$F864	Etablit la couleur graphique basse résolution.	Code de couleur dans A.	A	COLOR
$F871	Lit la couleur d'un point graphique basse résolution.	Rangée dans A, colonne dans Y.	A (contient le code de la couleur)	SCRN
$F940	Affiche les contenus de Y et X (format YYXX) sur l'écran ou tout autre périphérique.	Aucun.	Aucun	CALL – 1728
$F941	Affiche A et X (AAXX) comme ci-dessus.	Aucun.	Aucun	CALL – 1727
$F944	Affiche le contenu de X.	Aucun.	Aucun	CALL – 1724
$F948	Envoie trois espaces au périphérique choisi (déterminé par le contenu de CSW).	Aucun.	X, A	CALL – 1720
$F94A	Envoie de 1 à 256 espaces au périphérique choisi.	Nombre d'espaces dans A (ou 256 espaces)	Aucun.	SPC ( )*
$FB1E	Lit l'état des manettes de jeux 0, 1, 2 ou 3.	Numéro de manette dans X.	0 à FF dans Y contenu de A détruit	PDL( )
$FB2F	Etablit l'écran en texte 40 rangées sur 24 colonnes.	Aucun.	A	TEXT
$FB40	Etablit mode graphique basse résolution, efface l'écran et établit une fenêtre de texte de quatre lignes.	Aucun.	A, Y	GR
$FBD9	Envoie le caractère « sonnette » (BELL) (code 7 en ASCII) au périphérique de sortie courant.	Aucun.	A, Y	CALL – 1063
$FBE4	Le haut-parleur d'Apple II émet un « bip » pendant 0,1 s.	Aucun.	A, Y	CALL – 1052
$FC10	Envoie le caractère un pas arrière à l'écran, met à jour la position du curseur.	Aucun.	A	CALL – 1008
$FC1A	Place le curseur sur la ligne au-dessus. S'il se trouve sur la ligne supérieure, ne bouge pas.	Aucun.	A	CALL – 998
$FC42	Efface la fenêtre de texte à partir de la position du curseur et jusqu'à l'angle inférieur droit de l'écran.	Aucun.	A, Y	CALL – 958
$FC46	Efface la fenêtre de texte depuis les coordonnées passées aux registres et jusqu'à l'angle inférieur droit de l'écran.	Colonne dans Y, rangée dans A.	A, Y	CALL – 954
$FC58	Efface totalement la fenêtre de texte et ramène le curseur à l'angle supérieur gauche de l'écran.	Aucun.	A, Y	HOME*
$FC62	Envoie un retour chariot et une nouvelle ligne à l'écran.	Aucun.		CALL – 926
$FC66	Descend le curseur d'une ligne sur la même colonne. Déroule le texte d'une ligne si le curseur était en bas de l'écran.	Aucun	A, Y	CALL – 922
$FC70	Déroule la fenêtre de texte d'une ligne.	Aucun.	A, Y	CALL – 912
$FC9C	Efface le texte, de la position courante du curseur à la fin de la ligne. La position du curseur ne change pas.	Aucun.	A, Y	CALL – 868

Point d'entrée	Fonction	Registres à charger avant appel	Registres affectés	Equivalent Basic
$FCA8	Exécute une boucle de temporisation de 0,5 ($5x^2+27x+26$) en microsecondes.	Valeur de x dans A.	A	CALL – 856
$FD1B	Attend une touche ; le curseur flashe. Le générateur de nombres aléatoires est initialisé aux positions mémoires 78 et 79.	Aucun.	Caractère retourné dans A X, Y	CALL – 756
$FD35	Comme $FD1B, mais les codes d'échappement sont aussi autorisés.	Aucun.	Caractère retourné dans A X, Y	CALL – 715
$FD67	Envoie un retour chariot à l'écran et autorise l'entrée d'une ligne de données de 256 caractères maximum.	Caractère d'appel en mémoire en 51	Y, A X contient la longueur de l'entrée. La donnée de l'entrée commence en mémoire en $200	
$FDDA	Affiche le contenu de l'accumulateur sous forme de deux digits hexadécimaux.	Donnée dans A.	A	CALL – 55
$FDED	Sort un caractère vers le périphérique sélectionné.	Caractère dans A.	Aucun	PRINT
$FDF0	Sort un caractère vers la fenêtre de texte.	Caractère dans A.	Aucun	PRINT
$FE80	Etablit le mode vidéo inversé (noir sur blanc)	Aucun.	Y	INVERSE* CALL – 384
$FE84	Etablit le mode vidéo normal (blanc sur noir)	Aucun.	Y	NORMAL* CALL – 380
$FEB0	Retourne au Basic, élimine programmes et variables de la mémoire.	Aucun.	A, X, Y	CALL – 336
$FF20	Affiche un message ERR et émet un « bip ».	Aucun.	A	CALL – 211
$FF3A	Le haut-parleur émet un « bip ».	Aucun.	A	CALL – 198
$FF3F	Restaure le contenu des registres (valide seulement si le programme intrinsèque à $FF4A a d'abord été exécuté).	Aucun.	Sont restaurés : Registre A de 69 ($45) Registre S de 72 ($48) Registre X de 70 ($46) Pointeur de pile de 73 ($49) Registre Y de 71 ($47)	
$FF4A	Sauvegarde le contenu des registres dans les positions réservées en page zéro : Registre A dans 69 ($45) Registre S dans 72 ($48) Registre X dans 70 ($46) Pointeur de pile dans 73 ($49) Registre Y dans 71 ($47)	Aucun.	Aucun	CALL – 182
$FF69	Point d'entrée du moniteur.	Aucun.	Aucun	CALL – 151

* en exposant précise que la commande Basic n'est disponible qu'en Applesoft.

APPENDICE E

POSITIONS UTILES DE PEEK ET POKE

Chacune des positions mémoires listées ci-après est exprimée en décimal, avec une valeur inférieure à 32767. Les positions au-dessus de 32767 sont exprimées en négatif et disposent d'un nombre positif équivalent. Ajoutez 65536 au nombre négatif pour l'obtenir (par exemple, 65536 – 16384 = 49152).
Certaines des fonctions décrites ici sont activées simplement en y accédant. Cela signifie que chaque fois qu'un PEEK accède à une position mémoire spécifiée, l'action indiquée intervient. Une instruction POKE portant sur une position mémoire spécifiée déclenche aussi l'action, mais en raison des caractéristiques du microprocesseur d'Apple II, cette action est réellement déclenchée *deux fois*. Dans ce cas, POKE correspond à deux PEEK. Habituellement, cela ne fait aucune différence, mais dans le cas de – 16336 (le « clic » du haut-parleur), il y en aura une.

POSITIONS POUR COMMANDE DE LA FENÊTRE DE TEXTE ET DU CURSEUR

32 Marge gauche de la fenêtre de texte

Spécifie la colonne de la marge gauche de la fenêtre de texte. PEEK retourne une valeur entre 0 et 39, où 0 est le bord gauche de l'écran. La modification de cette position n'affecte pas la largeur de la fenêtre ; les bords gauche et droite sont tous deux déplacés.

Si votre POKE excède 39 à cette position, ou si la valeur lue, plus la largeur de la fenêtre de texte excède 40, quelques-unes des sorties, destinées à l'écran, ou toutes, iront en mémoire hors de l'espace écran, ce qui risquera de détruire votre programme ou vos données.

33 Largeur de la fenêtre de texte

Spécifie la largeur de la fenêtre de texte. Cette valeur est comprise entre 1 et 40. Une modification de cette position établit la marge droite à la colonne en nombre spécifié de caractères à partir de la marge gauche (de position 32).

Un zéro à cette position (donc, une largeur nulle) peut détruire l'interpréteur Basic. Si vous faites un POKE de plus de 40, ou si la valeur lue en 33, plus celle de 32 (marge gauche) excède 40, quelques-unes des sorties destinées à l'écran, ou toutes, seront placées en mémoire hors de la zone écran, ce qui risquera de détruire programmes ou données.

34 Bord supérieur de la fenêtre de texte

Spécifie le bord supérieur de la fenêtre de texte. La valeur doit être comprise entre 0 et 23 ; un 0 désigne la rangée supérieure de l'écran et un 23, la rangée inférieure. Si vous faites un POKE de plus de 23, tout ou partie de vos sorties destinées à l'écran iront en mémoire hors de la zone écran, risquant de détruire des données importantes. N'établissez pas le bord supérieur en-dessous du bord inférieur.

35 Bord inférieur de la fenêtre de texte

Spécifie le bord inférieur de la fenêtre de texte, avec une valeur entre 0 et 23. Le 0 est le bord supérieur de l'écran, le 23, le bord inférieur. Avec un POKE supérieur à 23, tout ou partie de vos sorties destinées à l'écran iront en mémoire dans la zone hors écran, risquent de détruire des données importantes. N'établissez pas le bord inférieur au-dessus du bord supérieur.

36 Position horizontale du curseur

Spécifie la position horizontale courante du curseur. Un PEEK retourne une valeur entre 0 et 39 ; c'est la position du curseur relative à la marge de gauche de la fenêtre de texte (et non nécessairement relative au bord gauche de l'écran). Cette position peut servir à établir un point au-delà du bord droit de la fenêtre de texte, où agira le prochain PRINT, mais le curseur n'y restera que jusqu'à l'affichage du caractère suivant. Ne mettez pas ici une valeur qui, ajoutée à la marge gauche (en position 32), excède 39.

Ce PEEK est équivalent à la fonction Applesoft POS.

37 Position verticale du curseur

Spécifie la position verticale courante du curseur. PEEK retourne une valeur de 0 à 23, relative au sommet de l'écran (et non de la fenêtre de texte). Ne mettez pas ici plus de 23.

POSITIONS POUR TRAITEMENT DES ERREURS

216 Indicateur d'erreur

Implique que ONERR GOTO est actif. Si le bit 7 de cette position est à 1 (donc, si la valeur de cette position est de 128 ou plus), c'est qu'une instruction ONERR GOTO a été rencontrée et que le contrôle est passé au numéro de ligne spécifié lorsque l'erreur est survenue. Un POKE de moins de 128 désactive un ONERR GOTO exécuté précédemment.

218 et 219 Numéro de ligne d'erreur

Lorsqu'une erreur déclenche un branchement avec ONERR GOTO, ces positions spécifient le numéro de ligne à laquelle l'erreur s'est produite. Vous l'obtiendrez avec PEEK (219)*256 + PEEK(218).

222 Code d'erreur

Spécifie quel type d'erreur est survenu. Les codes d'erreurs et leurs descriptions sont donnés dans l'appendice C.

POSITIONS POUR LE CLAVIER

– 16384 Caractère du clavier

Lit le clavier. Si la valeur, à cette position, est supérieure à 127 (donc, si le bit 7 est à 1), une touche a été actionnée. Déterminez le code ASCII de cette touche la plus récemment pressée en soustrayant 128 de cette valeur.

– 16368 Indicateur du clavier

Remet à zéro l'échantillonnage du clavier (bit 7 de la position – 16384) de façon que le caractère suivant puisse être reçu.

POSITIONS POUR SORTIE DES « CLICS »

– 16352 Clic pour la cassette

Produit un clic audible sur le jack de sortie cassette.

– 16336 Clic pour le haut-parleur

Produit un clic pour le haut-parleur interne.

COMMUTATEURS D'AFFICHAGE

Les positions mémoires suiantes positionnent des commutateurs déterminant les caractéristiques de l'affichage. Il n'y a pas de commutateurs réels, physiques ; seules, les commandes PEEK et POKE les affectent. Il existe quatre commutateurs positionnés individuellement comme le montre la figure E.1. Lorsque le mode texte est déterminé, le seul autre commutateur ayant quelqu'effet est celui de la Page 1/Page 2.

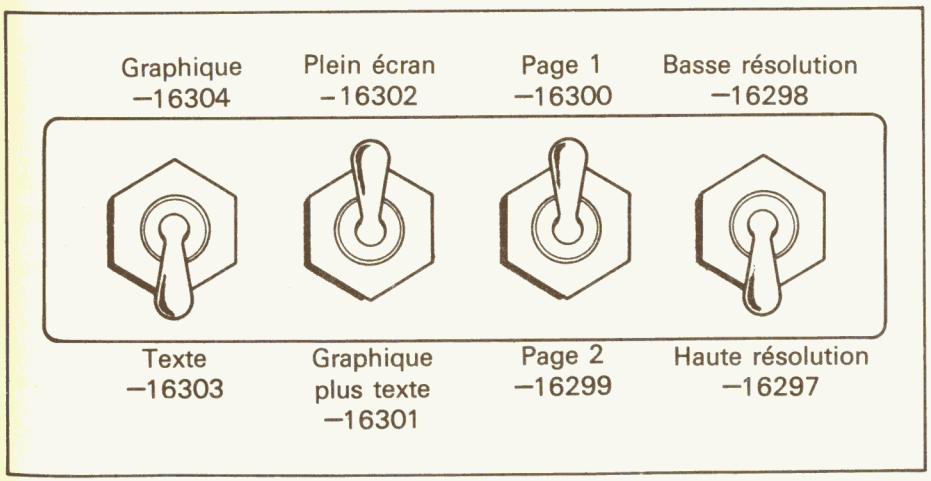


Fig. E.1. — Positions pour texte et graphique avec PEEK et POKE.

– 16304 Sélection du mode graphique
Sélectionne le mode graphique. L'écran graphique n'est pas effacé, en noir. Le mode graphique peut être basse ou haute résolution, en page 1 ou 2, plein écran graphique ou mêlé texte-graphique. Ces caractéristiques sont déterminées par les autres positions.

– 16303 Sélection du mode texte
Sélectionne le mode texte. Le texte peut être de page 1 ou page 2, en fonction des autres positions mémoires.

– 16302 Sélection du plein écran graphique
Sélectionne le graphique plein écran. Si l'écran est en mode texte, ce ne sera pas visible tant que la position – 16304 n'aura pas été adressée.

– 16301 Sélection graphique plus texte
Etablit une fenêtre de texte de quatre lignes au bas de l'écran. Si l'écran est en mode texte, ce ne sera pas visible tant que la position – 16304 n'aura pas été adressée.

– 16300 Sélection de page 1 d'écran
Sélectionne la page 1 graphique ou de texte.

– 16299 Sélection de la page 2 d'écran
Sélectionne la page 2 graphique ou de texte.

– 16298 Sélection du graphique basse résolution
Sélectionne le graphique basse résolution. Si l'écran est en mode texte, ce ne sera pas visible tant que – 16304 n'aura pas été adressé.

– 16297 Sélection du graphique haute résolution
Sélectionne le graphique haute résolution. Si l'écran est en mode texte, ce ne sera pas visible tant que – 16304 n'aura pas été adressé.

POSITIONS POUR COMMANDES DE JEUX

Ces positions mémoires mettent en service (« ON ») ou non (« OFF »). Les sorties de commandes de jeux, testent les poussoirs actionnés ou non, et activent un échantillonnage de sortie. La figure E.2 en donne le détail.
Toutes les entrées et sorties pour instructions PEEK et POKE sont ici reliées au connecteur de contrôle des jeux, montré par la figure E.3.

– 16296 Annonciateur 0 off
Débranche la sortie de la commande de jeux (annonciateur) numéro 0. La tension à la broche 15 du connecteur est établie à environ 0 volt (haut, TTL).

– 16295 Annonciateur 0 on
Branche la sortie de commande de jeux numéro 0. La tension à la broche 15 est d'environ 5 V (bas, TTL).

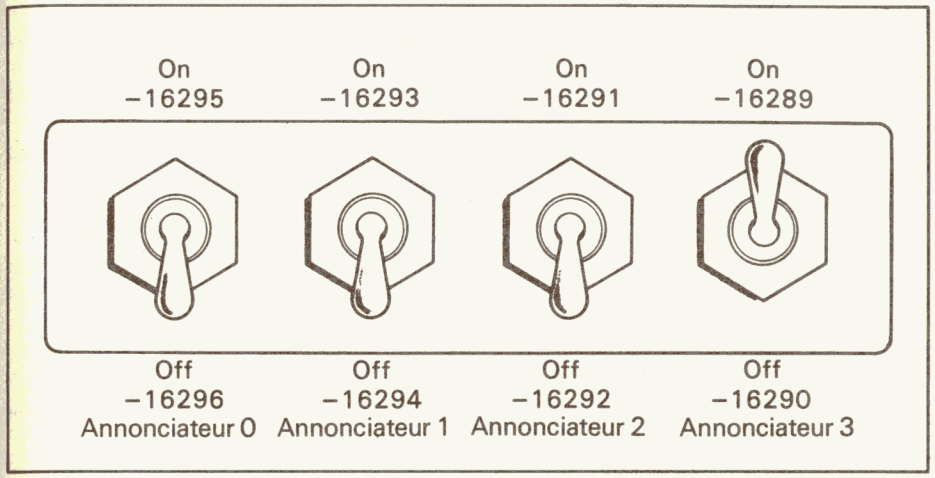


Fig. E.2. — Manipulation des commandes de sorties de jeux (« annonciateurs »).

– 16294 Annonciateur 1 off
Débranche la sortie de commande de jeux numéro 1. La tension à la broche 14 est d'environ 0 volt.

– 16293 Annonciateur 1 on
Branche la sortie de commande de jeux numéro 1. La tension en 14 est d'environ 5 V.

– 16292 Annonciateur 2 off
Débranche la sortie de commande de jeux numéro 2. La tension à la broche 13 est d'environ 0 volt.

– 16291 Annonciateur 2 on
Branche la sortie de commande de jeux numéro 2. La tension à la broche 13 est d'environ 5 V.

– 16290 Annonciateur 3 off
Débranche la sortie de commande de jeux numéro 3. La tension à la broche 12 est d'environ 0 volt.

– 16289 Annonciateur 3 on
Branche la sortie de commande de jeux numéro 3. La tension à la broche 12 est d'environ 5 V.

– 16287 Lecture du poussoir 0
Lorsque le bouton-poussoir de la commande de jeux 0 est pressé, la valeur de cette position excède 127 ; sinon, elle est égale ou inférieure à 127. Le poussoir est connecté à la broche 2 du connecteur général.

– 16286 Lecture du poussoir 1
Le poussoir pressé, la valeur de cette position excède 127 ; sinon, elle est égale ou inférieure à 127. Ce poussoir est connecté à la broche 3.

– 16285 Lecture du poussoir 2
Le poussoir pressé, la valeur de cette position excède 127 ; sinon, elle est égale ou inférieure à 127. Ce poussoir est connecté à la broche 4.

– 16272 Sortie échantillonnage
La broche 5 du connecteur de commande de jeux est normalement à +5 V. Un PEEK à – 16285 la ramène à zéro volt pour une demi-seconde. POKE déclenchera l'échantillonnage (ou « Strobe ») deux fois.

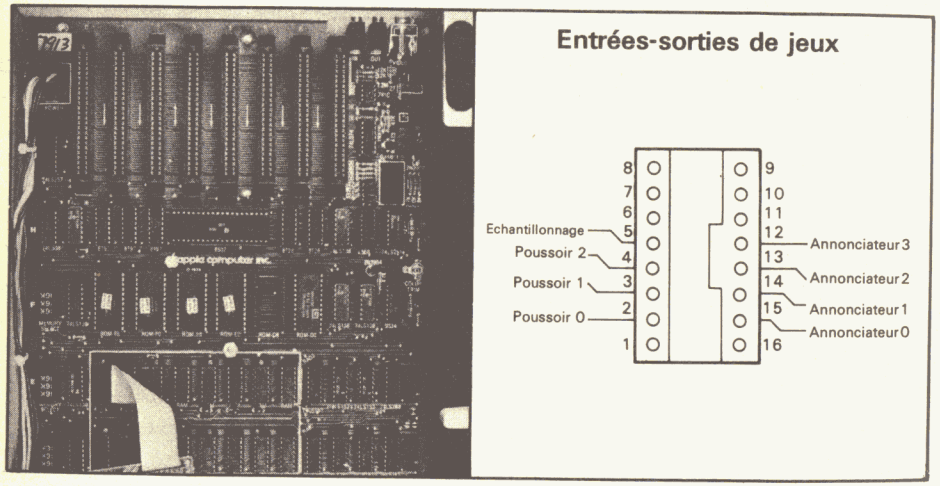


Fig. 4.3. — Commandes de jeux, en entrées-sorties.

APPENDICE F

MOTS RÉSERVÉS EN BASIC

pple II interprète chacun des mots suivants — réservés — comme des commandes, des ıstructions ou des fonctions en Basic. La seule exception se manifeste lorsque ces mots font partie de chaînes encadrées par des guillemets. N'utilisez donc pas ces mots réservés dans les noms de vos variables. Méfiez-vous surtout des mots réservés courts.
Vous pouvez introduire des mots réservés avec des espaces inclus ; Apple II les comprimera en chassant les espaces.

INTEGER BASIC

ABS	END	LET	PDL	SAVE
AND	FOR	LIST	PEEK	SCRN
ASC	GOSUB	LOAD	PLOT	SGN
AT	GOTO	LOMEM:	POKE	STEP
AUTO	GR	MAN	POP	TAB
CALL	HIMEM:	MOD	PRINT	TEXT
COLOR=	HLIN	NEW	PR#	THEN
CON	IF	NEXT	REM	TO
DEL	IN#	NOT	RETURN	TRACE
DIM	INPUT	NOTRACE	RND	VLIN
DSP	LEN	OR	RUN	VTAB

APPLESOFT

En Applesoft, les mots réservés ne prélèvent chacun qu'un octet de mémoire. Les adresses sont listées ici avec ces mots. Ils sont aussi listés par ordre numérique dans l'appendice I. Applesoft ne reconnaîtra pas le mot réservé TO correctement si :

1. Le premier caractère non espacé précédent TO est la lettre A ;
2. Un ou plusieurs espaces séparent le T du O.

ABS	(212)	HTAB	(150)	REM	(178)
AND	(205)	IF	(173)	RESTORE	(174)
ASC	(230)	IN#	(139)	RESUME	(166)
AT	(197)	INPUT	(132)	RETURN	(177)
ATN	(225)	INT	(211)	RIGHT$	(233)
CALL	(140)	INVERSE	(158)	RND	(219)
CHR$	(231)	LEFT$	(232)	ROT=	(152)
CLEAR	(189)	LEN	(227)	RUN	(172)
COLOR=	(160)	LET	(170)	SAVE	(183)
CONT	(187)	LIST	(188)	SCALE=	(153)
COS	(222)	LOAD	(182)	SCRN(	(215)
DATA	(131)	LOG	(220)	SGN	(210)
DEF	(184)	LOMEM:	(164)	SHLOAD	(154)
DEL	(133)	MID$	(234)	SIN	(223)
DIM	(134)	NEW	(191)	SPC(	(195)
END	(128)	NEXT	(130)	SPEED=	(169)
EXP	(221)	NORMAL	(157)	SQR	(218)
FLASH	(159)	NOT	(198)	STEP	(199)
FN	(194)	NOTRACE	(156)	STOP	(179)
FOR	(129)	ON	(180)	STORE	(168)
FRE	(214)	ONERR	(165)	STR$	(228)
GET	(190)	OR	(206)	TAB(	(192)
GOSUB	(176)	PDL	(216)	TAN	(224)
GOTO	(171)	PEEK	(226)	TEXT	(137)
GR	(136)	PLOT	(141)	THEN	(196)
HCOLOR=	(146)	POKE	(185)	TO	(193)
HGR	(145)	POP	(161)	TRACE	(155)
HGR2	(144)	POS	(217)	USR	(213)
HIMEM:	(163)	PRINT	(186)	VAL	(229)
HLIN	(142)	PR#	(138)	VLIN	(143)
HOME	(151)	READ	(135)	VTAB	(162)
HPLOT	(147)	RECALL	(167)	WAIT	(181)
				XDRAW	(149)

DOS

Les commandes de DOS sont considérées comme des mots réservés lorsqu'elles sont utilisées en mode immédiat ou dans une instruction PRINT commençant par un caractère CTRL-D (code 4 en ASCII).

APPEND	CHAIN	INIT	POSITION	SAVE
BLOAD	CLOSE	LOAD	READ	UNLOCK
BRUN	DELETE	LOCK	RENAME	VERIFY
BSAVE	EXEC	OPEN	RUN	WRITE

APPENDICE G

ORGANISATION DES MÉMOIRES

ORGANISATION GÉNÉRALE

Les mémoires d'Apple II se divisent en trois catégories : vives, à lecture-écriture (aussi appelées mémoires à accès aléatoire, ou RAM, de « random access memory ») ; les mémoires mortes, à lecture seulement (on ROM, de « read-only memory ») ; et positions d'entrées-sorties, notées E/S (ou « I/O », de « input-output »). Les positions 0 à 49151 ($BFFF en hexadécimal) sont en RAM ; 49152 ($C000) à 53247 ($CFFF) sont en ROM. Votre système ne dispose pas nécessairement de mémoire à toutes ces positions. Par exemple, avec 16K de RAM, les positions 16384 ($4000) à 49151 ($BFFF) ne seront pas utilisables.

Le tableau G.1 montre les attributions mémoires. Remarquez l'existence de deux blocs libres, entourant les deux pages graphiques haute résolution. Le pointeur système LOMEM: garde la trace de la frontière inférieure de cette zone libre et le pointeur HIMEM: marque sa frontière supérieure. Cette RAM peut servir à stocker bon nombre de choses, parmi lesquelles l'interpréteur Applesoft sur disque ou cassette, le DOS, le graphique haute résolution, et vos programmes et variables.

Tableau G.1. — Organisation de la mémoire.

Position		Type de mémoire	Fonction
Décimal	Hex		
0-255	$0-$OFF	RAM	Programmes système
256-511	$100-$1FF	RAM	Pile système
512-767	$200-$2FF	RAM	Buffer d'entrée clavier
768-1023	$300-$3FF	RAM	Positions des vecteurs du moniteur
1024-2047	$400-$7FF	RAM	Texte et graphique basse résolution, page 1
2048-3071	$800-$BFF	RAM	Texte et graphique basse résolution, page 2
3072-8191	$C00-$1FFF	RAM	Libre
8192-16383	$2000-$3FFF	RAM	Graphique haute résolution, page 1
16384-24575	$4000-$5FFF	RAM	Graphique haute résolution, page 2
24576-49151	$6000-$BFFF	RAM	Libre
49152-49279	$C000-$C07F	I/O	Positions spéciales
49280-49407	$C080-$C0FF	I/O	Espace pour cartes d'E/S périphériques
49408-51199	$C100-$C7FF	I/O	Carte mémoire périphérique
51200-53247	$C800-$CFFF	I/O	Carte d'expansion mémoire périphérique
53248-65535	$D000-$FFFF	ROM	Integer Basic, Applesoft, Monitor ou Autostart Monitor, etc.

LES INTERPRÉTEURS

Comme le montre le tableau G.1, l'interpréteur Integer Basic réside toujours en ROM. L'interpréteur Applesoft aussi si votre système dispose de la carte firmware ou de la carte système langage. Autrement, l'interpréteur Applesoft occupe environ 10K octets en mémoire, à partir de 2048 ($800).

MÉMOIRE POUR LE DOS

Il vous faut au moins 16K de mémoire pour employer le DOS. Lorsqu'il est appelé, le DOS occupe environ 10K au sommet de la mémoire. HIMEM: est établi juste en-dessous. La figure G.1 montre quelles sections sont occupées par le DOS, selon la taille mémoire des systèmes. Remarquez qu'il vous faut au moins 24K de RAM pour supporter à la fois le DOS et les disques (ou cassettes), et au moins 32K pour employer la page 1 graphique haute résolution avec le DOS. La même figure montre aussi clairement le conflit entre l'Applesoft disque ou cassette et le graphique haute résolution de la page 1.
Le DOS emploie des sections additionnelles de la mémoire au cours du chargement (voir la figure G.2). Tout ce qui se trouvait antérieurement dans ces zones sera perdu après l'appel du DOS.

MÉMOIRE POUR L'INTEGER BASIC

Les programmes en Integer Basic résident en haut de la mémoire vive libre, à partir de HIMEM:. Comme le montre la figure G.3, HIMEM: est ajusté automatiquement dès lors que vous ajoutez, supprimez ou changez des lignes de programme.
Les variables sont rangées à partir de LOMEM: et montent. Au fur et à mesure, LOMEM: est ajusté automatiquement. Chaque variable numérique est rangée en mémoire avec quatre attributs: Le nom de la variable, l'octet du commutateur « on/off » de DSP, la position mémoire de la variable suivante, et la valeur, ou les valeurs réelles de la variable.
Le nom de la variable peut atteindre 100 caractères. Chacun d'eux est représenté en mémoire par son code ASCII, avec le bit de plus fort poids à 1.

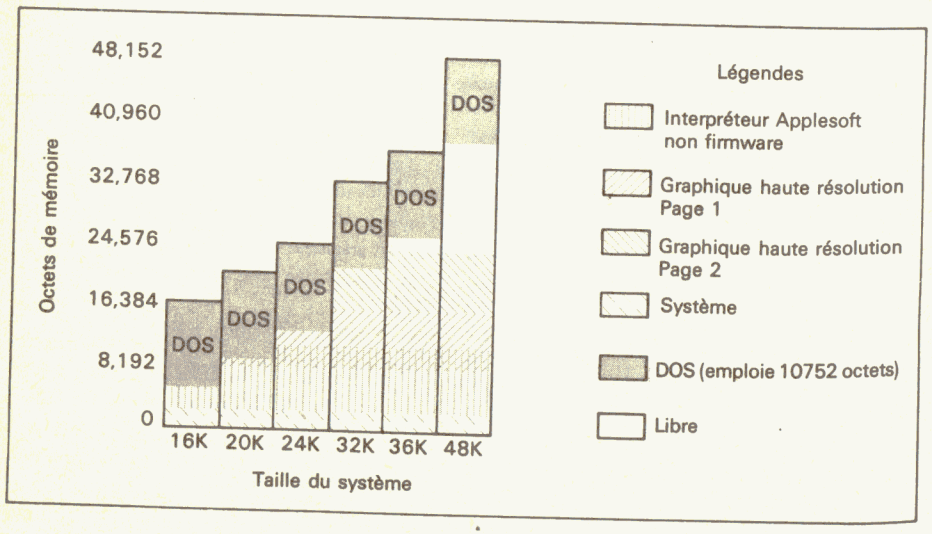


Fig. G.1. — Organisation de la mémoire vive (RAM).

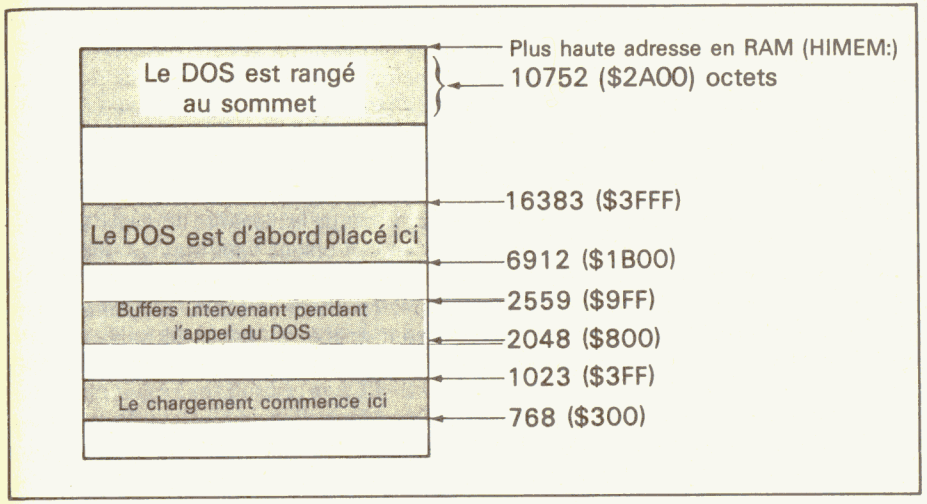


Fig. G.2. — L'occupation mémoire pendant l'appel du DOS.

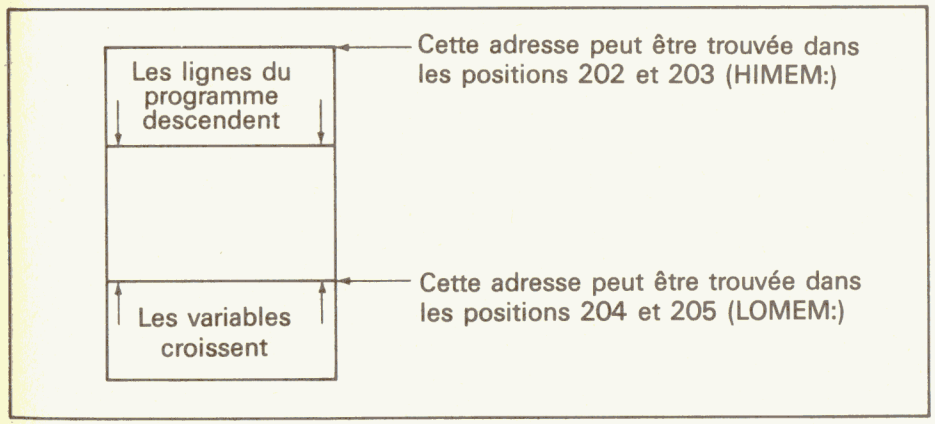


Fig. G.3. — La carthographie des programmes en mémoire avec l'Integer Basic.

L'octet commutateur de la commande DSP indique si DSP est active pour cette variable. Cet octet est normalement à zéro, mais passe à 1 lorsque DSP est exécutée pour cette variable, et revient à zéro avec un NO DSP.
L'adresse de la variable suivante est rangée en deux octets, l'octet de faible poids d'abord. La donnée est rangée par paires d'octets, ceux de faible poids d'abord. Si la variable n'est pas un tableau, il n'y a qu'une paire. S'il s'agit d'un tableau, on trouvera une paire par élément, dans l'ordre à partir de l'élément zéro. Aucune distinction n'est faite entre une variable simple et un tableau avec le même nom ; la variable simple est l'élément zéro du tableau.

Les chaînes sont rangées de façon semblable. Le nom de la variable, l'octet DSP et l'adresse de la variable suivante sont rangés de la même façon que les variables numériques. Le code ASCII de chaque caractère de la chaîne occupe un octet, le bit de fort poids étant à 1. Le dernier caractère de la chaîne est suivi par un octet de terminaison de chaîne, dans lequel le bit de fort poids est à zéro.

MÉMOIRE POUR APPLESOFT

Les lignes des programmes en Applesoft occupent le bas de la mémoire libre vive (RAM), à partir de LOMEM: comme le montre la figure G.4. Lorsque vous ajoutez, supprimez ou changez des lignes, LOMEM: s'ajuste automatiquement. Les variables numériques simples et les pointeurs de chaînes sont rangés directement au-dessus du programme. Les pointeurs de tableaux et de chaînes de tableaux sont stockés au-dessus des variables simples. Les *valeurs* des chaînes sont stockées au sommet de la mémoire, à partir de HIMEM: qui s'ajuste automatiquement.

Chaque variable numérique et chaque pointeur de chaîne emploient sept octets. Chaque variable réelle demande deux codes ASCII, soit deux octets, pour le nom de la variable (tous deux avec le bit de fort poids à 0). La valeur est rangée en notation scientifique avec un octet pour l'exposant et quatre octets pour le mantisse. Les octets de la mantisse sont dans l'ordre, du plus fort ou plus faible poids.

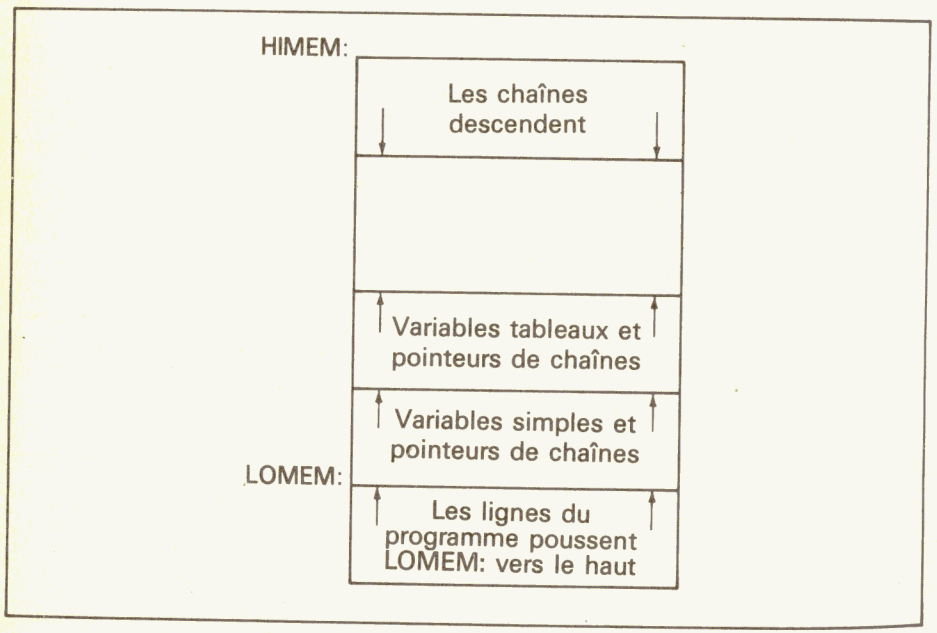


Fig. G.4. — La mémoire pour les programmes en Applesoft.

Chaque variable entière emploie aussi deux codes ASCII (deux octets) pour le nom de la variable (le bit de plus fort poids est à 1) et deux octets pour sa valeur, l'octet de fort poids en premier.

Chaque pointeur de chaîne emploie deux caractères ASCII (deux octets) pour le nom de la variable (le bit de fort poids est à 1, son voisin à 0), un octet pour la longueur de la chaîne

et deux octets pour la valeur de la chaîne, octet de faible poids d'abord. Les trois dernièers octets d'une variable entière et les deux derniers octets d'un pointeur de chaîne restent inutilisés.

Les tableaux numériques et les pointeurs de tableaux sont rangés immédiatement au-dessus des variables. Le nom de la variable est un code ASCII sur deux octets, avec les deux bits de plus fort poids à 0 pour des variables réelles, à 1 pour des variables entières, à 10 pour les pointeurs de chaînes.

Le nom de la variable est suivi par deux octets donnant la position de la variable suivante, relativement au premier octet du nom de cette variable. L'octet de faible poids d'abord. Suit ensuite un octet pour le nombre de dimensions, puis viennent deux octets par dimension (le plus fort poids d'abord) donnant la taille de chaque dimension. Ces tailles sont listées dans l'ordre inverse, c'est-à-dire que la taille de la première dimension vient à la fois. Chaque élément du tableau est alors listé, de l'élément (0,0,...,0) à l'élément (N,N,...N). Ils sont rangés dans l'ordre, avec l'index de celui qui se trouve le plus à gauche incrémenté d'abord. Chaque élément d'un tableau de réels occupe cinq octets, un pour l'exposant, et quatre (le plus fort poids d'abord) pour la mantisse. Chaque entier occupe deux octets, fort poids d'abord. Chaque pointeur de chaîne emploie trois octets, un pour la longueur de la chaîne et deux (faible poids d'abord) pour son adresse.

Les valeurs des chaînes sont rangées au sommet de la mémoire vive libre. Elles demandent un octet par caractère. Les chaînes dupliquées ne sont rangées qu'une fois ; deux pointeurs, ou plus, peuvent pointer une unique position. Lorsque de nouvelles valeurs chaînes sont créées, elles sont placées dans l'espace disponible voisin (HIMEM: s'ajuste vers le bas). Les chaînes qui ne sont plus utilisées restent en mémoire. La fonction FRE les élimine et repositionne HIMEM:.

APPENDICE H

LE FORMAT DE « DISK II »

Les informations sont rangées sur la disquette sur 35 *pistes* concentriques. Elles sont numérotées de 0 à 34 (soit $0 à $22). Chaque piste est divisée en 16 segments, appelés *secteurs* et numérotés de 0 à 15 ($0 à $F). Chaque secteur peut stocker 256 octets de données. Au total, les 455 secteurs du disque mémorisent jusqu'à 116480 octets.
Le DOS transfère les données du et vers le disque, par secteur. Il emploie deux séries de buffers de 256 octets en mémoire, l'une pour l'écriture et l'autre pour la lecture, pour chaque fichier actif.
Chaque type de fichier (texte, programme et binaire) dispose de son propre format sur disque. Les fichiers de texte sont sauvegardés en code ASCII, un octet par caractère. Un octet à zéro marque la fin du fichier. Tous les octets d'un fichier de texte sont interprétés comme étant du texte.
Les deux premiers octets du premier secteur d'un programme Basic donnent la longueur du programme, l'octet de faible poids d'abord. Le reste contient le programme, en code ASCII. Avec un fichier Applesoft, les mots réservés occupent un seul octet et ne sont pas écrits complètement. Voyez l'appendice H pour leur liste en ordre alphabétique, et I pour leur liste par ordre numérique.
Les deux premiers octets du premier secteur d'un fichier binaire montrent l'adresse de départ des données binaires en mémoire vive, le premier octet d'abord. Les deux octets suivants donnent la longueur du fichier, le faible poids d'abord. Le reste contient les données binaires.

LA LISTE PISTES/SECTEURS

Le DOS écrit normalement sur le disque s'il trouve un secteur libre. Cela signifie qu'un fichier multisecteur peut être distribué sur plusieurs pistes. Le DOS établit une liste des numéros de secteurs et de pistes utilisés par chaque fichier et la range dans un ou plusieurs secteurs de la disquette. C'est *la liste (ou le fichier) pistes/secteurs.*
Chaque secteur de la liste pistes/secteurs contient un pointeur visant le secteur suivant de cette liste (s'il y en a un), et pouvant pointer jusqu'à 122 secteurs de fichiers.
Si un secteur de fichier est inutilisé, son pointeur dans la liste pistes/secteurs est à 0. Si un secteur entier, au début de la liste pistes/secteurs n'a que des pointeurs à zéro, ce secteur n'est pas sur le disque. De ce fait, si l'enregistrement numéro 5000 est le seul et unique enregistrement d'un fichier aléatoire dont le paramètre L est de 256 (un enregistrement par secteur), seuls deux secteurs de la disquette seront employés : l'un pour les données de l'enregistrement 5000, l'autre pour le 41e secteur de la liste pistes/secteurs.
L'octet 0 et les octets 3 à 12 de la liste pistes/secteurs ne servent pas. Les octets 1 et 2 contiennent les numéros de piste et secteur, respectivement, du secteur suivant de la liste. Si ces octets sont à zéro, ce secteur est le dernier de la liste.

LE RÉPERTOIRE

Le DOS utilise la piste 17 ($11) de la disquette pour un répertoire. Ce répertoire contient, pour chaque fichier, le nom du fichier, son type, le nombre de secteurs occupés (modulo 256) et la position du fichier dans la liste pistes/secteurs. La plupart de ces informations sont affichées sur l'écran par la commande CATALOG.

Chaque secteur du répertoire contient des informations pour un maximum de sept fichiers. Le répertoire commence à la piste 17, secteur 15. Lorsque ce secteur est plein, il continue secteur 14, etc., et ce jusqu'au secteur 1. Le répertoire peut contenir jusqu'à 84 fichiers.

L'octet 0 et les octets 3 à 10 de chaque secteur du répertoire sont inutilisés. Les octets 1 et 2 contiennent, respectivement, les numéros de piste et secteur du secteur suivant du répertoire. S'ils sont tous deux à zéro, ce secteur est le dernier. Les octets 11 à 255 contiennent les entrées du répertoire. Chacune occupe 35 octets ; la première va de l'octet 11 à 45, la seconde de 46 à 80, etc.

Les entrées du répertoire sont toutes rédigées sur le même format. Le tableau H.1 le présente. Le tableau H.2 explique comment le type de fichier est codé dans chaque entrée. Le secteur 0 de la piste 17 n'a pas d'entrées. Il contient les identificateurs d'états, la description physique et les informations d'espace disponible pour le disque. Le tableau H.3 donne les caractéristiques de cet important secteur, appelé la *table des matières du volume*.

Chaque groupe de quatre octets des octets 56 à 195 de la table des matières contient la carte de disponibilité pour l'une des pistes du disque. Chaque carte identifie quels secteurs de sa piste sont occupés, et lesquels restent libres. Un bit à 0 marque un secteur occupé, à 1, un secteur libre. Le tableau H.4 montre les octets et les secteurs qu'ils identifient.

Tableau H.1. — Format des entrées du répertoire.

Numéro relatif de l'octet	Contenu de l'octet
0	Numéro de piste de la liste pistes/secteurs. Passe à 255 Lorsque le fichier est effacé (les contenus précédents sont retenus, en relatif, par l'octet 34).
1	Numéro de secteur du fichier de la liste pistes/secteurs.
2	Type du fichier. Voir le tableau H.2.
3-32	Nom du fichier, en ASCII.
33	Nombre de secteurs utilisés par le fichier, modulo 256.
34	Marque de fin. Normalement à 0, mais modifiée selon le contenu précédent de l'octet relatif 0 lorsque le fichier est effacé.

Tableau H.2. — Codage des types de fichiers pour le répertoire.

Bit	Caractéristique
0	Le fichier est un programme en Integer Basic si ce bit est à 1.
1	Le fichier est un programme en Applesoft si ce bit est à 1.
2	Le fichier est un fichier binaire si ce bit est à 1.
3-6	Réserve pour extension ultérieure.
7	Le fichier est verrouillé si ce bit est à 1.
Si les bits 0 à 6 sont à zéro, le fichier est un fichier de texte.	

Tableau H.3. — Table des matières du volume (secteur 0, piste 17).

Octets	Description
0	Inutilisé.
1	Numéro de piste du premier secteur du répertoire.
2	Numéro de secteur du premier secteur du répertoire.
3	Numéro de référence du DOS.
4-5	Inutilisés.
6	Numéro de volume de la disquette.
7-38	Inutilisés.
39	Nombre maximal de paires pistes/secteurs possibles dans chaque secteur de la liste pistes/secteurs.
40-47	Inutilisés.
48-51	Masque pour la carte de disponibilité des secteurs.
52	Nombre de pistes par disquette.
53	Nombre de secteurs par disquette.
54-55	Nombre d'octets par secteur, octets de faible poids en 54, fort poids en 55.
56-59	Carte de disponibilité des secteurs, piste 0.
60-63	Carte de disponibilité des secteurs, piste 1.
64-195	Cartes de disponibilité des secteurs, pistes 2 à 195.
196-255	Inutilisés.

Tableau H.4. — Carte de disponibilité des secteurs.

Octet	Bit	Secteur
Premier	7	12
	6	11
	5	10
	4	9
	3	8
	2	7
	1	6
	0	5
Second	7	4
	6	3
	5	2
	4	1
	3	0
	2-0	Inutilisé
Troisième	Tous	Inutilisé
Quatrième	Tous	Inutilisé

APPENDICE I

CODES ASCII ET OCTETS DES MOTS RÉSERVÉS EN APPLESOFT

Le premier tableau de cet appendice donne les codes ASCII de 1 à 96 et le caractère représenté. Les codes ASCII de 96 à 127 produisent le même caractère sur l'écran de l'Apple II, bien que sur d'autres périphériques de sortie, ils se traduisent par des lettres minuscules. Aucune touche du clavier ne génère ces codes.
Les codes ASCII 128 à 255 répètent les codes 0 à 127. Aucune touche ne les produit.
Le second tableau donne la liste des mots réservés en Applesoft, par ordre de leur code qui n'occupe qu'un octet en mémoire. Ces codes vont de 128 à 255 et remplacent le mot complet, en mémoire et sur le disque. L'appendice F contient cette même liste de mots réservés mais par ordre alphabétique. Ces codes sont appelés « jetons » par les Américains (« tokens »), chaque jeton donnant accès à un mot réservé.

★

Codes des caractères ASCII

Code ASCII	Caractère affiché sur l'écran	Touche	Code ASCII	Caractère affiché sur l'écran	Touche
0		Ctrl-@	48	0	0
1		Ctrl-A	49	1	1
2		Ctrl-B	50	2	2
3		Ctrl-C	51	3	3
4		Ctrl-D	52	4	4
5		Ctrl-E	53	5	5
6		Ctrl-F	54	6	6
7	(sonnette)	Ctrl-G	55	7	7
8	(espace arrière)	Ctrl-H or ←	56	8	8
9		Ctrl-I	57	9	9
10	(ligne suivante)	Ctrl-J	58	:	:
11		Ctrl-K	59	;	;
12		Ctrl-L	60	<	<
13	(retour chariot)	Ctrl-M	61	=	=
14		Ctrl-N	62	>	>
15		Ctrl-O	63	?	?
16		Ctrl-P	64	@	@
17		Ctrl-Q	65	A	A
18		Ctrl-R	66	B	B
19		Ctrl-S	67	C	C
20		Ctrl-T	68	D	D
21	(espace avant)	Ctrl-U or →	69	E	E
22		Ctrl-V	70	F	F
23		Ctrl-W	71	G	G
24	(annulation ligne)	Ctrl-X	72	H	H
25		Ctrl-Y	73	I	I
26		Ctrl-Z	74	J	J
27		Esc	75	K	K
28		n.a.	76	L	L
29		Ctrl-Shift-M	77	M	M
30		Ctrl-^	78	N	N
31		n.a.	79	O	O
32	espace	barre d'esp.	80	P	P
33	!	!	81	Q	Q
34	"	"	82	R	R
35	#	#	83	S	S
36	$	$	84	T	T
37	%	%	85	U	U
38	&	&	86	V	V
39	'	'	87	W	W
40	(	(	88	X	X
41	)	)	89	Y	Y
42	*	*	90	Z	Z
43	+	+	91	[	n.a.
44	,	,	92	\	n.a.
45	-	-	93	]	Shift-M
46	.	.	94	^	^
47	/	/	95		n.a.

n.a. = non disponible sur le clavier d'Apple II

Les mots réservés Applesoft par ordre de leur code

Code	Mot	Code	Mot	Code	Mot
128	END	164	LOMEM:	200	+
129	FOR	165	ONERR	201	-
130	NEXT	166	RESUME	202	*
131	DATA	167	RECALL	203	/
132	INPUT	168	STORE	204	^
133	DEL	169	SPEED=	205	AND
134	DIM	170	LET	206	OR
135	READ	171	GOTO	207	>
136	GR	172	RUN	208	=
137	TEXT	173	IF	209	<
138	PR#	174	RESTORE	210	SGN
139	IN#	175	&	211	INT
140	CALL	176	GOSUB	212	ABS
141	PLOT	177	RETURN	213	USR
142	HLIN	178	REM	214	FRE
143	VLIN	179	STOP	215	SCRN(
144	HGR2	180	ON	216	PDL
145	HGR	181	WAIT	217	POS
146	HCOLOR=	182	LOAD	218	SQR
147	HPLOT	183	SAVE	219	RND
148	DRAW	184	DEF	220	LOG
149	XDRAW	185	POKE	221	EXP
150	HTAB	186	PRINT	222	COS
151	HOME	187	CONT	223	SIN
152	ROT=	188	LIST	224	TAN
153	SCALE=	189	CLEAR	225	ATN
154	SHLOAD	190	GET	226	PEEK
155	TRACE	191	NEW	227	LEN
156	NOTRACE	192	TAB(	228	STR$
157	NORMAL	193	TO	229	VAL
158	INVERSE	194	FN	230	ASC
159	FLASH	195	SPC(	231	CHR$
160	COLOR=	196	THEN	232	LEFT$
161	POP	197	AT	233	RIGHT$
162	VTAB	198	NOT	234	MID$
163	HIMEM:	199	STEP		

APPENDICE J

TABLES DE CONVERSION

Cet appendice contient les tables de conversion :

- Hexadécimal à binaire
- Hexadécimal à décimal (en entiers)

Conversion hexadécimale-binaire

Utilisez la table ci-dessous pour convertir des nombres hexadécimaux de 0 à 0F en valeurs binaires de 0000 à 1111, ou vice-versa.
Pour convertir des nombres binaires plus grands en hexadécimal, convertissez quatre bits à la fois, en allant de la droite vers la gauche. S'il y a moins de quatre bits dans le dernier groupe de gauche, ajoutez des zéros pour le compléter. En voici un exemple :

$$100101_2 = \underbrace{\underbrace{00100101_2}_{2\quad 5}}_{25_{16}}$$

Pour convertir des nombres hexadécimaux supérieurs à 0F en binaire, travaillez digit par digit. En voici un exemple :

$$\underbrace{\underbrace{37_{16}}_{3\quad 7}}_{01100111_2}$$

Hexadécimal	Binaire
00	0000
01	0001
02	0010
03	0011
04	0100
05	0101
06	0110
07	0111
08	1000
09	1001
0A	1010
0B	1011
0C	1100
0D	1101
0E	1110
0F	1111

TABLE DE CONVERSION HEXADÉCIMAL-DÉCIMAL D'ENTIERS

La table ci-dessous fournit une conversion directe en des nombres entiers en hexadécimal, dans la gamme de 0 à FFF, et des entiers en décimal, de 0 à 4095. Pour convertir des valeurs supérieures. Les valeurs de la table suivante peuvent être ajoutées :

Hexadécimal	Décimal	Hexadécimal	Décimal
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

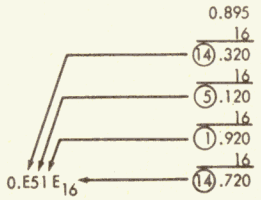
Les fractions hexadécimales peuvent être converties en fractions décimales ainsi :

1. Exprimez la fraction hexadécimale en un nombre entier de fois 16^{-n}, où n est le nombre de places hexadécimales significatives à la droite du point hexadécimal.
   $0.CA9BF3_{16} = CA9BF3_{16} \times 16^{-6}$
2. Trouvez l'équivalent décimal de l'entier hexadécimal.
   $CA9BF3_{16} = 13\ 278\ 195_{10}$
3. Multipliez l'équivalent décimal par 16^{-n}

$$\begin{array}{r} 13\ 278\ 195 \\ \times\ 596\ 046\ 448 \times 10^{-16} \\ \hline 0.791\ 442\ 096_{10} \end{array}$$

Les fractions décimales peuvent être converties en fractions hexadécimales par des multiplications successives de la fraction décimale par 16_{10}.
Après chaque multiplication, la portion entière ext extraite pour former une fraction hexadécimale construite à la droite du point hexadécimal. Cependant, puisque l'arithmétique décimale sert dans cette conversion, la portion entière de chaque produit doit être convertie en nombre hexadécimal.

Exemples : Convertir 0.895_{10} en son équivalent hexadécimal.



CONVERSION HEXADÉCIMAL-DÉCIMAL D'ENTIERS

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

CONVERSION HEXADÉCIMAL-DÉCIMAL D'ENTIERS

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1562	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

CONVERSION HEXADÉCIMAL-DÉCIMAL D'ENTIERS

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

CONVERSION HEXADÉCIMAL-DÉCIMAL D'ENTIERS

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

CONVERSION HEXADÉCIMAL-DÉCIMAL D'ENTIERS

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

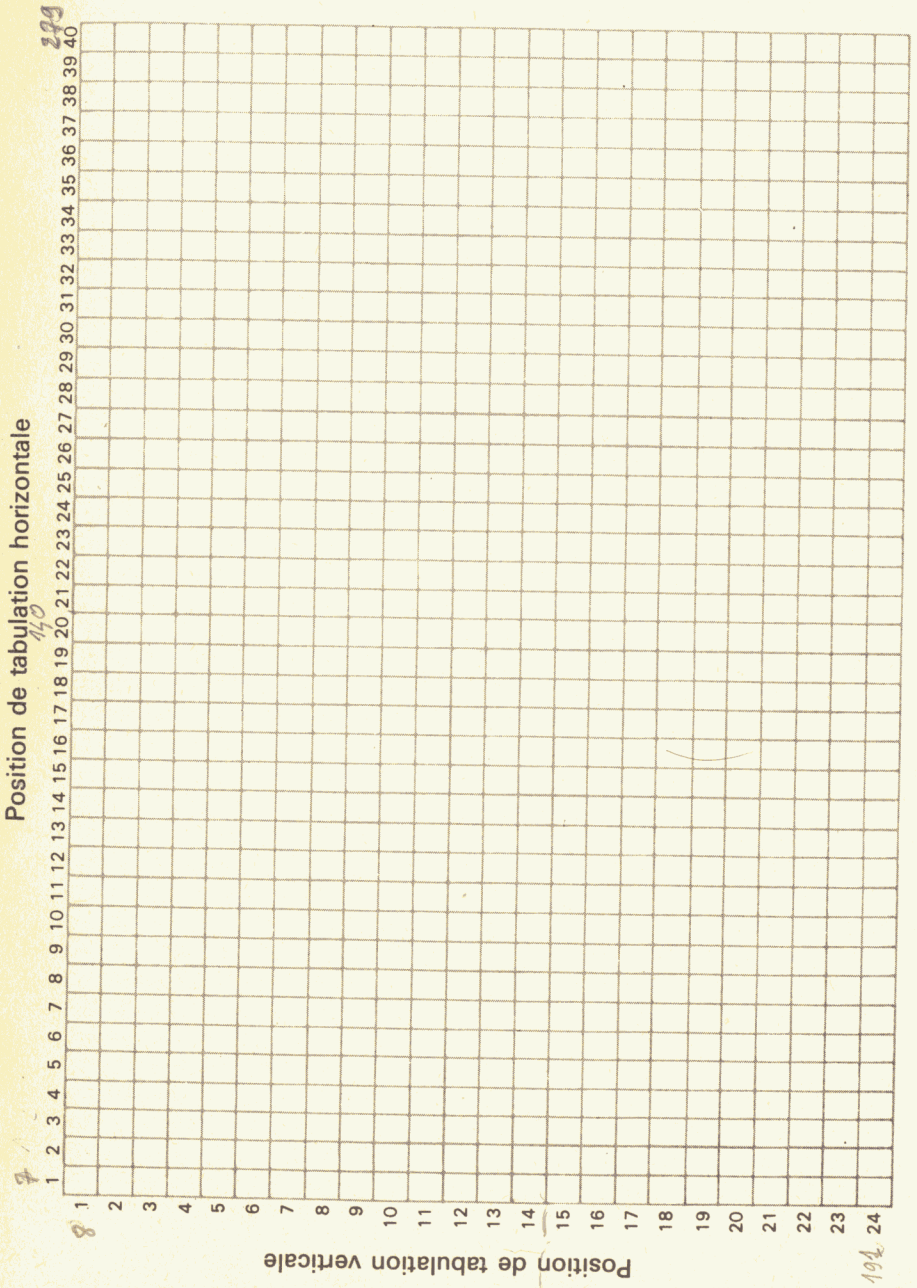
APPENDICE K

FORMATS D'AFFICHAGE SUR L'ÉCRAN

Utilisez les dessins de cet appendice pour planifier vos dessins sur l'écran. Sur ces formes, les rangées et les colonnes commencent en 1 pour le texte. Sur l'écran graphique basse résolution, les rangées et les colonnes commencent en 0, comme le font les commandes graphiques.

★

Ecran de texte



Ecran graphique basse résolution

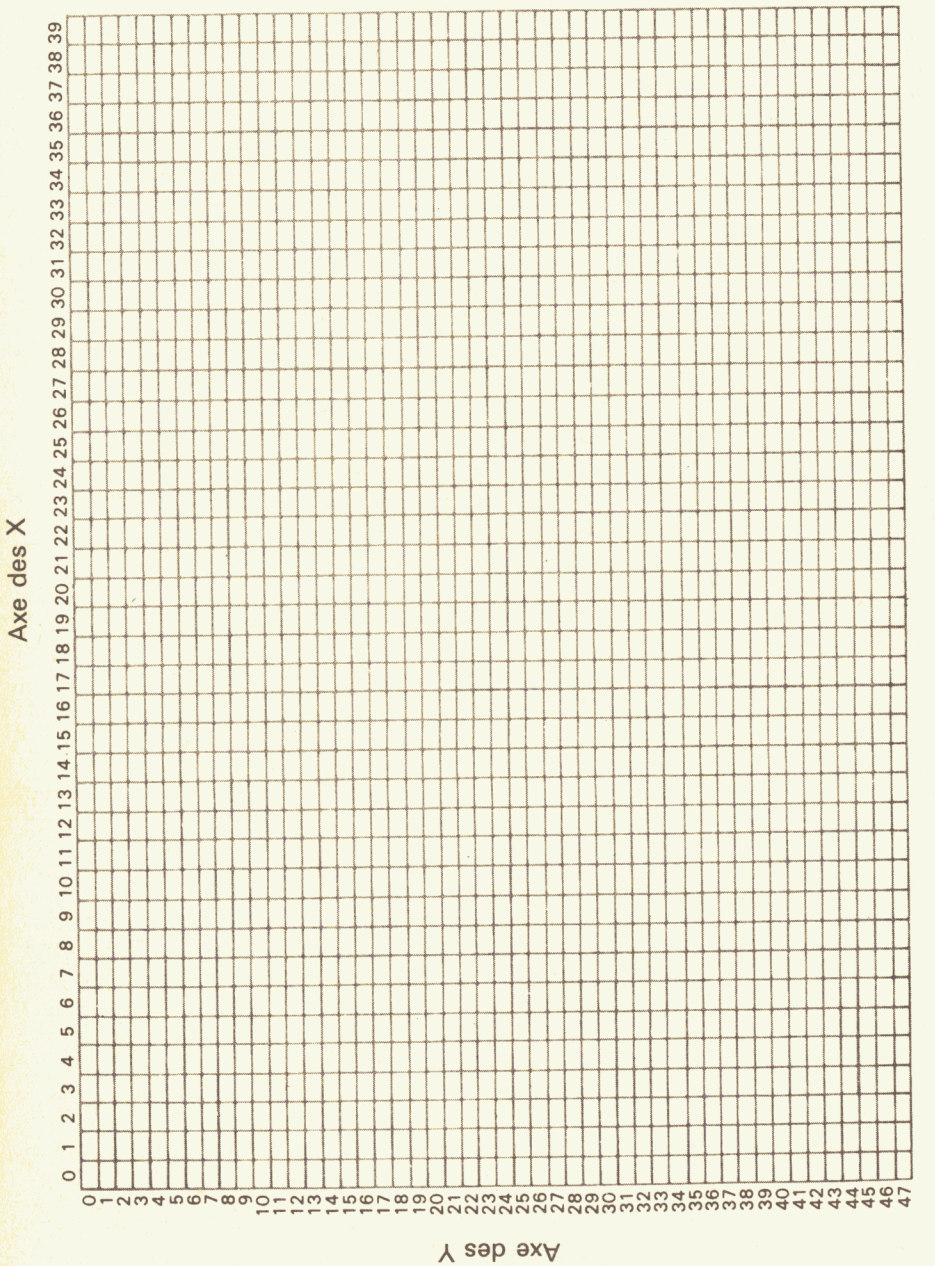


TABLE DES MATIÈRES

Collection Osborne

S. E. C. F.

Editions Radio

EXTRAIT DU CATALOGUE SECF EDITIONS RADIO

70 PROGRAMMES BASIC APPLE II par **Lon Poole et Mary Borchers.** — Chaque programme est accompagné d'un commentaire et d'exemples.
200 pages, format 21 × 29,7

PRATIQUE DE L'APPLE II par **Henri Lilen.** — Le meilleur ouvrage pour débuter sur l'Apple II grâce à une présentation originale très progressive, avec de nombreux exemples dans les deux Basics, le graphique, les jeux et sons.
192 pages, format 21 × 29,7

SCIENCE AND ENGINEERING PROGRAMS APPLE II EDITION par **J. Heilt** — Des programmes mathématiques et scientifiques utilisables pour de nombreuses applications techniques.
224 pages, format 21 × 27,5

INTERFACES POUR MICROPROCESSEURS ET MICRO-ORDINATEURS par **H. Lilen.** — Tout ce qu'il faut savoir pour relier les microprocesseurs, micro-ordinateurs et périphériques.
272 pages, format 16 × 24

6502 PROGRAMMATION EN LANGAGE ASSEMBLEUR par **L. Leventhal.** — Un cours complet de programmation, en langage assembleur du microprocesseur 6502.
480 pages, format 16 × 24

INITIATION AUX MICRO-ORDINATEURS NIVEAU 1 par **Adam Osborne.** — Pour apprendre à exploiter les micro-ordinateurs sans connaissance particulière préalable.
304 pages, format 14 × 20

INITIATION AUX MICRO-ORDINATEURS NIVEAU 2 par **Adam Osborne.** — Un ouvrage complet pour une connaissance approfondie des microprocesseurs et des micro-ordinateurs.
488 pages, format 16 × 24

AMPLIFICATEURS OPERATIONNELS, 100 APPLICATIONS par **G. Decès et H. Lilen.** — Des solutions pour les applications les plus représentatives de l'électronique moderne.
144 pages, format 16 × 24

MEMOIRES INTEGREES par **H. Lilen.** — Fonctionnement, choix, utilisation.
288 pages, format 16 × 24

CONVERTISSEURS AD/DA par **R. Fontenay.** — 55 montages pratiques de conversion analogique-numérique et numérique-analogique.
208 pages, format 16 × 24

ENVOI DU CATALOGUE SUR DEMANDE

S. E. C. F.

Editions Radio

Services commerciaux : 3, rue de l'Éperon, 75006 Paris - C.C.P. La Source 340 37.40 H

Magasin de vente : 9, rue Jacob, 75006 Paris - Tél. 329.63.70

MANUEL DE L'UTILISATEUR APPLE II

Ce livre constituera votre guide de l'ordinateur individuel Apple II. En un seul volume, il décrit cet ordinateur ainsi que ses périphériques les plus usuels, y compris les unités à disquettes et les imprimantes.
Qu'est-ce qu'un Apple II ? Comment le ferez-vous fonctionner ? Les deux premiers chapitres de ce livre répondent à ces questions.

Les quatre chapitres suivants vous enseignent à rédiger vos propres programmes en Basic. Le chapitre 5 expose en particulier comment employer l'unité à disquettes pour ranger des programmes et des listes de données et le chapitre 6 vous montre comment programmer du graphique.

Ensuite, le chapitre 7 explique ce qu'est le moniteur et le moniteur « Autostart » du point de vue du programmeur. Il vous montre également comment introduire des programmes en langage assembleur dans vos programmes en Basic.

Le chapitre 8 contient une description complète de chaque instruction et de chaque fonction disponibles avec les deux versions du Basic, y compris celles relatives aux disquettes.

9 782709 108805
ISBN 2 7091 0880 1
09 0253 6
S. E. C. F.
ÉDITIONS RADIO
Prix : 95,00 F