Apple II
Technical Notes

Developer Technical Support

Pascal
#14: Apple Pascal 1.3 TREESEARCH and IDSEARCH

Revised by: Cheryl Ewy November 1988
Written by: Cheryl Ewy June 1985

This Technical Note describes the TREESEARCH and IDSEARCH routines which were built into Pascal 1.2 and earlier, but which are separate entities for Pascal 1.3.

Introduction

In Apple II Pascal versions 1.0 through 1.2, TREESEARCH and IDSEARCH were special-purpose built-in routines which could be called from within a Pascal program. The routines existed primarily for use by the Compiler and Libmap but were also available for use by applications. In Apple II Pascal 1.3, the routines were removed due to space requirements. Since some applications use these routines, they are being supplied as 6502 codefiles which can be linked to Pascal programs. To use the routines, the Pascal host program must declare them as EXTERNAL (see the following sections for details). After compiling the host program, use the Linker to link the file TRS.CODE (TREESEARCH) or the file IDS.CODE (IDSEARCH).

The TREESEARCH Function

TREESEARCH is a fast assembly language function for searching a binary tree with a particular kind of structure. The external declaration is:

```
FUNCTION TREESEARCH (ROOTPTR : ^NODE;
                     VAR NODEPTR : ^NODE;
                     NAMEID : PACKED ARRAY [1..8] OF CHAR) :INTEGER;
        EXTERNAL;
```

The call syntax is:

```
RESULT := TREESEARCH (ROOTPTR, NODEPTR, NAMEID);
```

where ROOTPTR is a pointer to the root node of the tree to be searched, NODEPTR is a reference to a pointer variable to be updated by TREESEARCH, and NAMEID contains the eight-character name to be searched for in the tree.

The nodes in the binary tree are assumed to be linked records of the type:

```
NODE = RECORD
              NAME : PACKED ARRAY[1..8] OF CHAR;
              LEFTLINK, RIGHTLINK : ^NODE;

              {other fields can be anything}

       END;
```

The actual names of the type and the field identifiers are not important; TREESEARCH assumes only that the first eight bytes of the record contain an eight-character name and are followed by two pointers to other nodes.

It is also assumed that names are not duplicated within the tree and are assigned to nodes according to an alphabetical rule; for a given node, the name of the left subnode is alphabetically less than the name of the node, and the name of the right subnode is alphabetically greater than the name of the node. Finally, any links that do not point to other nodes should be NIL.

TREESEARCH can return any of three values:

0 The name passed to TREESEARCH (as the third parameter) has been found in the tree. The node pointer (second parameter) now points to the node with the specified name.

1 The name is not in the tree. If it is added to the tree, it should be the right subnode of the node pointed to by the node pointer.

-1 The name is not in the tree. If it is added to the tree, it should be the left subnode of the node pointed to by the node pointer.

The TREESEARCH function does not perform any type checking on the parameters passed to it.

The IDSEARCH Procedure

IDSEARCH is a fast assembly language procedure that scans Apple II Pascal source text for identifiers and reserved words. Note that IDSEARCH recognizes only identifiers and reserved words—you have to scan for special characters and comments yourself.

The external declaration is:

```
       PROCEDURE IDSEARCH (VAR OFFSET:INTEGER;
                           VARBUFFER:BYTESTREAM);
              EXTERNAL;
```

The call syntax is:

```
       IDSEARCH (OFFSET, BUFFER);
```

To use IDSEARCH, you must include the following declarations in your program. Note that the variables (except BUFFER) must be declared in exactly the order and types shown.

```
TYPE

       {SYMBOL is the enumerated type of symbols in the Apple // Pascal language}

       SYMBOL =      (IDENT,COMMA,COLON,SEMICOLON,LPARENT,RPARENT,DOSY,TOSY,
                     DOWNTOSY,ENDSY,UNTILSY,OFSY,THENSY,ELSESY,BECOMES,LBRACK,
                     RBRACK,ARROW,PERIOD,BEGINSY,IFSY,CASESY,REPEATSY,WHILESY,
                     FORSY,WITHSY,GOTOSY,LABELSY,CONSTSY,TYPESY,VARSY,PROCSY,
                     FUNCSY,PROGSY,FORWARDSY,INTCONST,REALCONST,STRINGCONST,
                     NOTSY,MULOP,ADDOP,RELOP,SETSY,PACKEDSY,ARRAYSY,RECORDSY,
                     FILESY,OTHERSY,LONGCONST,USESSY,UNITSY,INTERSY,IMPLESY,
                     EXTERNLSY,OTHERWSY);

       {The reserved words corresponding to the above symbols are as follows -

       DOSY       - DO       WITHSY     - WITH        RELOP      - IN
       TOSY       - TO       GOTOSY     - GOTO        SETSY      - SET
       DOWNTOSY   - DOWNTO   LABELSY    - LABEL       PACKEDSY   - PACKED
       ENDSY      - END      CONSTSY    - CONST       ARRAYSY    - ARRAY
       UNTILSY    - UNTIL    TYPESY     - TYPE        RECORDSY   - RCORD
       OFSY       - OF       VARSY      - VAR         FILESY     - FILE
       THENSY     - THEN     PROCSY     - PROCEDURE   USESSY     - USES
       ELSESY     - ELSE     FUNCSY     - FUNCTION    UNITSY     - UNIT
       BEGINSY    - BEGIN    PROGSY     - PROGRAM     INTERSY    -.INTERFACE
       IFSY       - IF                    SEGMENT     IMPLESY    -.IMPLEMENTATION
       CASESY     - CASE     FORWARDSY  - FORWARD     EXTERNLSY  - EXTERNAL
       REPEATSY   - REPEAT   NOTSY      - NOT         OTHERWSY   - OTHERWISE
       WHILESY    - WHILE    MULOP      - AND,DIV,MOD
       FORSY      - FOR      ADDOP      - OR                            }

       {OPERATOR expands the multiplicative (MULOP), additive (ADDOP) and
       relational (RELOP) operators}

       OPERATOR =    (MUL,RDIV,ANDOP,IDIV,IMOD,PLUS,MINUS,OROP,LTOP,LEOP,
                     GEOP,GTOP,NEOP,EQOP,INOP,NOOP);

       ALPHA = PACKED ARRAY [1..8] OF CHAR;

VAR
       {the next four variables must be declared in the order shown}
       OFFSET : INTEGER;
       SY : SYMBOL;
       OP : OPERATOR;
       ID : ALPHA;
```

IDSEARCH begins by looking for an identifier at the character pointed to by BUFFER[OFFSET] and assumes that this character is alphabetic. IDSEARCH produces the following results:

- BUFFER[OFFSET] points to the character following the identifier just found.
- ID contains the first eight alphanumeric characters of the identifier just found, left justified and padded with spaces as necessary.
- SY contains the symbol associated with the identifier just found if the identifier is a reserved word or IDENT if the identifier is not a reserved word. For example, the identifier MOD translates to MULOP; the identifier ARRAY translates to ARRAYSY, and the identifier MYLABEL translates to IDENT.
- OP contains the operator value which corresponds to the identifier just found if the identifier is an operator, or NOOP if the identifier is not an operator. For

example, the identifier `MOD` translates to `IMOD`, the identifier `ARRAY` translates to `NOOP`, and the identifier `MYLABEL` translates to `NOOP`.

The following is an example of calling IDSEARCH:

```
BEGIN
       IF BUFFER[OFFSET] IN ['A'..'Z','a'..'z'] THEN
              IDSEARCH (OFFSET, BUFFER);
END;
```

The following is an algorithmic representation of IDSEARCH:

```
PROCEDURE IDSEARCH (VAR OFFSET:INTEGER; VAR BUFFER:BYTESTREAM);
BEGIN
       ID := ScanIdentifier (OFFSET, BUFFER);
       SY := LookUpReservedWord (ID);
       OP := LookUpOperator (ID);
END;
```

ScanIdentifier increments OFFSET until BUFFER[OFFSET] is no longer part of an identifier, copying the first eight alphanumeric characters passed into ID (left justified, padding with spaces).

LookUpReservedWord translates an identifier into the associated symbol (defaulting to IDENT).

LookUpOperator translates an identifier into the associated operator (defaulting to NOOP).