NPS-PASCAL
A Microcomputer-based Implementation of the
PASCAL Programming Language

by

Konrad Stephen Tinius

March 1980

Thesis Advisor: Bruce J. MacLennan

Approved for public release; distribution unlimited
DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.
NPS-PASCAL is a student research project at the Naval Postgraduate School, the goal of which is the implementation of the PASCAL programming language on a microcomputer system. NPS-PASCAL will consist of two programs, a compiler which produces intermediate code, and an interpreter.
which will interpret the intermediate code, or a translator, which will produce target machine code. NPS-PASCAL is designed to conform to the requirements of the PASCAL Standard, as defined by the British Standards Institute/International Standards Organization Working Draft/3.

The compiler program, the subject of this thesis, performs the lexical, syntactic and semantic analysis of a PASCAL program. NPS-PASCAL is written in INTEL's PL/M-80 programming language and executes on the CP/M operating system.
NPS-PASCAL
A Microcomputer-based implementation of the
PASCAL programming language

by

Konrad Stephen Tinius
Captain, United States Air Force
B.S., The Ohio State University, 1972

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1980

Author

Konrad Stephen Tinius

Approved by:

Thesis Advisor

Second Reader

Chairman, Department of Computer Science

Dean of Information and Policy Sciences


Abstract

NPS-PASCAL is a student research project at the Naval Postgraduate School, the goal of which is the implementation of the PASCAL programming language on a microcomputer system. NPS-PASCAL will consist of two programs, a compiler which produces intermediate code, and an interpreter, which will interpret the intermediate code, or a translator, which will produce target machine code. NPS-PASCAL is designed to conform to the requirements of the PASCAL Standard, as defined by the British Standards Institute/International Standards Organization Working Draft/3.

The compiler program, the subject of this thesis, performs the lexical, syntactic and semantic analysis of a PASCAL program. NPS-PASCAL is written in INTEL's FL/V-80 programming language and executes on the CP/M operating system.
TABLE OF CONTENTS

I. INTRODUCTION................................................................. 7
   A. BACKGROUND................................................................. 7
   B. APPROACH................................................................. 7

II. NPS-PASCAL COMPILER IMPLEMENTATION.............................. 14
   A. NPS-PASCAL LANGUAGE BACKGROUND................................. 17
   B. COMPILER ORGANIZATION............................................... 11
   C. SCANFP................................................................. 11
   D. SYMBOL TABLE.......................................................... 14
      1. Symbol Table Construction...................................... 14
         a. Label Entries.................................................. 18
         b. Constant Entries............................................. 18
         c. Type Entries.................................................. 22
         d. Variable Entries............................................. 30
         e. Procedure and Function Entries........................... 46
   E. PARSER................................................................. 46
   F. CODE GENERATION...................................................... 52
      1. Storage space allocation......................................... 52
      2. Arithmetic Operations........................................... 56
      3. Set Operations.................................................. 57
      4. String Operations.............................................. 57
      5. Procedures and Functions...................................... 57
      6. Input-Output.................................................... 65
      7. NPS-PASCAL Pseudo Operators................................... 66

III. PROBLEMS IDENTIFIED AND CORRECTED.............................. 67

IV. PROBLEMS IDENTIFIED AND NOT CORRECTED......................... 77

V. CONCLUSIONS................................................................. 75
APPENDIX A - Compiler Error Messages .................................. 78
APPENDIX B - Intermediate Code DECODE Program ....................... 79
APPENDIX C - SYMLUMP Display Program .................................. 79
APPENDIX D - Compiler Source Code Structure ............................ 83
APPENDIX E - COMPIL, LINK and LOCATE Instructions .................. 85
APPENDIX F - Disk Directories .............................................. 89
NPS-PASCAL PROGRAM LISTINGS .............................................. 91
SYRTE.SRC .......................................................................... 91
SCAN.SRC ........................................................................ 99
PARSER.SRC ................................................................... 108
TABLES.SRC .................................................................... 114
SYMBOL.SRC ................................................................... 119
SYNTH1.SRC ................................................................... 126
SYNTH2.SRC ................................................................... 169
DESCETE.SRC ................................................................... 136
SYMTABLE.SRC .................................................................. 237
LIST OF REFERENCES ............................................................. 216
INITIAL DISTRIBUTION LIST .................................................. 227
I. INTRODUCTION

A. BACKGROUND

NPS-PASCAL is an implementation of the PASCAL programming language on a microcomputer system. NPS-PASCAL is a continuing research project in the Computer Science Department at the Naval Postgraduate School, Monterey, California. The original NPS-PASCAL design and programs were written by Maj Joaquin C. Gracida, USMC, and Lt Robert P. Stilwell (SC) USN, in their thesis submitted June 1979. Their work is contained in Ref. 1. Maj Gracida and Lt Stilwell implemented the basic constructs of the PASCAL language in a one-pass compiler and code generator. Thesis work was continued in June 1979 by Lt John L. Byrnes, USN, who added code to implement many missing constructs, and developed a number of user assistance programs. His work is contained in Ref. 2. Thesis work was continued again in October 1979, with the goal of completing the compiler portion of NPS-PASCAL. Follow-on thesis work will lead to an NPS-PASCAL interpreter/translator and a complete PASCAL system. In the discussion which follows, it is assumed that the reader is familiar with Refs. 1 and 2.

B. APPROACH

The first step in continuing the NPS-PASCAL project was to convert the source programs from PL/M to PL/M-80 and transfer them from the IFM 360/67-based timesharing system to the Intel Microprocessor Development System. This would
permit the compiler to be developed and debugged in a completely microprocessor oriented environment, and would eliminate the need to use the PL/M cross compiler.

The next step was to study the program listings and previous theses to gain a detailed familiarity with the project. Included in this step was acquiring a working knowledge of the Intel ISIS-II operating system and the PL/M-68 compiler and its attendant linking and loading programs and utilities. Since VPS-PASCAL is compiled under the ISIS-II operating system, but executes under the CP/M operating system, it was also necessary to learn the CP/M utilities for transferring files between systems, and the CP/M run-time decouplers, DDT and SID.

The largest portion of this thesis effort consisted of making corrections and additions to existing code, adding code where necessary, tracing execution to locate logic and data errors, correcting documentation, and running test PASCAL programs. Implementation of the record construct required changing the original grammar and correcting the parse tables.

To avoid testing the compiler with syntactically incorrect PASCAL programs, test programs were selected from the PASCAL User Manual and Report [3], from various student texts on PASCAL, and from the PASCAL Validation Suite[4]. The test programs from the Validation Suite were particularly helpful, in that they exercised the full range of any given PASCAL construct.
An attempt was made to upgrade and complete the SYMCTAB user assistance program described in Sec. 2. However, it was abandoned and a substitute program, SYMUMP, was developed. SYMUMP provides an ordered, addressed version of the symbol table, and provides a much more useful and efficient means of accessing the symbol table.

It was felt that it would be beneficial to include and consolidate the documentation and descriptions from the previous theses into a single document, so sections of Both 1 and 2 appear in this thesis. The appropriate sections were updated to reflect changes in the program code or structure. In others, descriptions were expanded and diagrams were added, or the section was included in its entirety.
II. NPS-PASCAL LANGUAGE IMPLEMENTATION

A. NPS-PASCAL LANGUAGE BACKGROUND

NPS-PASCAL is an implementation of the PASCAL language based on the ANSI/ISO Working Draft of Standard PASCAL [8], referred to in this thesis as "STANDARD PASCAL." NPS-PASCAL is in compliance with STANDARD PASCAL's definition of a conforming processor with the following three exceptions:

1) Identifiers, directives, and labels can be of any length, as prescribed by STANDARD PASCAL, provided their uniqueness can be determined from the first thirty characters.

2) Integers are limited to any value between \(-2,147,483,648\) and \(+2,147,483,647\). Real values can take on any negative or positive value consisting of fourteen digits multiplied by ten to the \(-64\)th power through ten to the \(+63\)rd power.

3) "TOP" is a special symbol, or reserved word, in the NPS-PASCAL vocabulary indicating "end of program."

Consequently, any program that conforms to the rules of STANDARD PASCAL, and meets the above listed qualifications, constitutes a syntactically correct NPS-PASCAL program.

The University of Toronto's parse table generator [6] was used to specify the NPS-PASCAL grammar in PALR/1 form. The generator operates on the IBM 7090/67 and produces parse tables for the language, thus permitting extensions and corrections to be made in an easy and efficient manner.
P. COMPILER ORGANIZATION

The compiler structure, diagrammed in Fig. 1, performs a single pass through the source program, produces an intermediate language file and may print an optional listing of the source program to the console. The one pass approach was taken to provide speed and to reduce the size of the compiler. The disadvantage of the one-pass design is the inability to specify the exact location where program execution resumes after a forward branch. To solve this problem, labels are placed in the intermediate code where execution should continue. The resolution of label locations is then the responsibility of the interpreter/translator as it scans the intermediate code.

The compiler builds the symbol table, converts all numbers to their internal representation, and generates the intermediate code file and the symbol table file. The compiler accepts input parameters to control the listing of the source program, production numbers, or token numbers. The creation of the intermediate file can also be suppressed if it is not needed.

C. SCANNER

The scanner analyzes the source program character by character and passes each token identified to the parser. The scanner can provide a listing of the source statements and eliminate comments.

The scanner is written in four sections which are selectively executed depending on the first non-blank
**FIGURE 1.**

NPS-PASCAL Compiler Structure
character of the input string. When the section to execute has been determined, the remainder of the token is scanned and placed in the input array ACCUM. The first byte of the ACCUM array contains the length of the token. In the case of tokens that exceed the size of the ARRAY (72 bytes), a continuation flag is set to allow the scanner and parser to accept the rest of the token.

The four sections of the scanner process strings, numbers, identifiers and reserved words, and special characters, respectively. The string processing section is executed whenever the first character of the token is a quotation mark. The scanner then accepts each succeeding character until a second quotation mark is found, indicating the end of the string. The section that processes numbers determines the type of the number being scanned as it scans each character. This determination is used by subroutines later in the compilation process to perform type checking and conversion to internal representation. When the scanner recognizes an identifier, it searches the vocabulary table to determine if it is a reserved word. If so, the scanner returns the token number associated with the reserved word. Special characters found in the vocabulary table are handled as separate tokens except in two cases. If a period is followed immediately by numeric characters, the scanner assumes a real number is being scanned. When a pair of special characters occurs consecutively, (for instance :=), the scanner passes both characters as a single token after
assigning the appropriate token number from the vocabulary table.

1. SYMBOL TABLE

The symbol table is used to store the attributes of labels, constants, type declarations, variable identifiers, procedures, functions and file declarations. This stored information is used by the compiler to verify that the program is semantically correct and to assist in code generation. Access to the symbol table is through various subroutines using based global variables to uniquely address the elements of each entry.

1. Symbol Table Construction.

The symbol table is an unordered linked list of entries which grows from the last byte of the compiler toward high memory. Individual entries are either accessed via a chained hash addressing technique (as illustrated in Figure 2), or by means of address pointer fields contained in other entries. This latter method of access is required since not all entries in the symbol table have an identifier, called a printname, associated with them.

Each location in the hash table contains the head of a singly linked list of entries whose printname, when evaluated, results in the same hash value. A zero in any cell of the hash table indicates that there are no entries whose printname produces that value. During symbol table construction or access, the global variable PRINTNAME contains the address of a string of bytes whose first

EXAMPLE:

Identifier A3 = (41E + 41F) MOD 128 = 71F
Identifier PA = (41E + 41F) MOD 128 = 71F

<table>
<thead>
<tr>
<th>HASH TABLE</th>
<th>SYMBOL TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>6749</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td></td>
</tr>
</tbody>
</table>

SYMBOL TABLE ACCESS

FIGURE 2.
element is the length of the current identifier, followed by
the identifier's ASCII characters. The global variable
SYMBOL contains the hash code value of the identifier. The
hash code is the sum of the hex values of the PRINCIPAL's
ASCII characters, modulo 128 (base 16). Entries that produce
the same hash code are linked together in the symbol table
by a chain which is accessed via the entry's collision
field. The chain is constructed in such a way as to have the
most recent entry at the head of the chain.

Each entry in the symbol table contains a number of
fields, some of which are common to all entries, and some of
which apply only to particular classes of entries. All
entries have the same first three fields: the collision
field in the first two bytes; the previous symbol table
entry address field in the third and fourth bytes; and the
form field in the fifth byte. The remaining fields are used
to uniquely describe each entry's attributes and
characteristics.

There are eight different types of entries in the
NPS-PASCAL symbol table. Each of these types has a unique
three bit code in the right-most three bits of its form
field. The remaining five bits in the form field further
subdivide the entry types among the eight classes according
to the particular characteristics of the type involved. The
form field bit assignments are summarized in Table 1. The
characteristics are described in detail as each type of
symbol table entry is presented below.
### FCPM Field Organization

<table>
<thead>
<tr>
<th>Form Value</th>
<th>Name of Entry</th>
<th>Bit Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>22H</td>
<td>Label</td>
<td></td>
</tr>
<tr>
<td>x1H</td>
<td>Constant</td>
<td></td>
</tr>
<tr>
<td>01H</td>
<td>Unsigned identifier</td>
<td>00 000 011</td>
</tr>
<tr>
<td>41H</td>
<td>Signed identifier</td>
<td>01 100 011</td>
</tr>
<tr>
<td>09H</td>
<td>Integer</td>
<td>00 000 011</td>
</tr>
<tr>
<td>11H</td>
<td>Real</td>
<td>00 100 011</td>
</tr>
<tr>
<td>19H</td>
<td>String</td>
<td>01 000 011</td>
</tr>
<tr>
<td>x2H</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>42H</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>4AH</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>52H</td>
<td>Char</td>
<td></td>
</tr>
<tr>
<td>5AH</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>7AH</td>
<td>Type declaration</td>
<td></td>
</tr>
<tr>
<td>x3H</td>
<td>Variable</td>
<td></td>
</tr>
<tr>
<td>03H</td>
<td>Scalar</td>
<td></td>
</tr>
<tr>
<td>09H</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>17H</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>13H</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>23H</td>
<td>Complex</td>
<td></td>
</tr>
<tr>
<td>29H</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>x4H</td>
<td>Procedure</td>
<td></td>
</tr>
<tr>
<td>04H</td>
<td>Procedure</td>
<td></td>
</tr>
<tr>
<td>x5H</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>05H</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>x6H</td>
<td>File</td>
<td></td>
</tr>
<tr>
<td>26H</td>
<td>File</td>
<td>00 000 111</td>
</tr>
<tr>
<td>x7H</td>
<td>User defined</td>
<td></td>
</tr>
<tr>
<td>07H</td>
<td>Scalar</td>
<td></td>
</tr>
<tr>
<td>2FH</td>
<td>Enumerated subrange</td>
<td></td>
</tr>
<tr>
<td>4FH</td>
<td>Integer subrange</td>
<td>00 000 111</td>
</tr>
<tr>
<td>67H</td>
<td>Character subrange</td>
<td></td>
</tr>
<tr>
<td>17H</td>
<td>Array</td>
<td></td>
</tr>
<tr>
<td>1FH</td>
<td>Record</td>
<td></td>
</tr>
<tr>
<td>5FH</td>
<td>Field (of record)</td>
<td></td>
</tr>
<tr>
<td>9FH</td>
<td>Tag field</td>
<td></td>
</tr>
<tr>
<td>6FH</td>
<td>Variant field</td>
<td></td>
</tr>
<tr>
<td>27H</td>
<td>Set</td>
<td></td>
</tr>
<tr>
<td>2F8</td>
<td>Field</td>
<td></td>
</tr>
<tr>
<td>37H</td>
<td>Pointer</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.

---

17
a. Label entries

The form field of a label entry has the value of 02H. The hash value of the label's printname is in the next byte; the hash value is stored for collision resolution later. The length of the label follows in the next one byte field. The printname characters appear, one per byte, after the length field. A two byte field following the printname characters contains a sequentially generated integer value which is assigned as the label's internal label number. This value is used as the target for branching in the intermediate code. An example of a label entry is shown in Fig. 3.

b. Constant Entries

The form field of a constant symbol table entry identifies the type of entry, and the particular type of the constant as well. There are five valid types of constants in NPS-PASCAL: an unsigned identifier with FORM = 01H; a signed identifier with FORM = 02H; an integer with FORM = 04H; a real value with FORM = 11H; and a string constant with FORM = 10H. Following the form field are the printname hash field, the length field, and the printname characters.

The value field may consist of another length field and the printname characters in the case of identifier and string constants, or it may contain the internal representation of a constant number (two bytes for integers or eight bytes for reals). Two examples of constant entries are shown in Figs. 4 and 5.
<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>7327E</td>
<td>AEH</td>
</tr>
<tr>
<td>7321E</td>
<td>01F</td>
</tr>
<tr>
<td>7322E</td>
<td>7FH</td>
</tr>
<tr>
<td>7325E</td>
<td>03F</td>
</tr>
<tr>
<td>7324E</td>
<td>06H</td>
</tr>
<tr>
<td>7325E</td>
<td>06H</td>
</tr>
<tr>
<td>7306E</td>
<td>07E</td>
</tr>
<tr>
<td>7307E</td>
<td>06H</td>
</tr>
<tr>
<td>7308E</td>
<td>07H</td>
</tr>
<tr>
<td>7309E</td>
<td>08H</td>
</tr>
<tr>
<td>730AH</td>
<td>09H</td>
</tr>
</tbody>
</table>

**Symbol Table Label Entry**

*Figure 3.*
<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>7317E</td>
<td>23E</td>
</tr>
<tr>
<td>7318E</td>
<td>23E</td>
</tr>
<tr>
<td>7319E</td>
<td>23E</td>
</tr>
<tr>
<td>731AE</td>
<td>23E</td>
</tr>
<tr>
<td>731BE</td>
<td>23E</td>
</tr>
<tr>
<td>731CE</td>
<td>23E</td>
</tr>
<tr>
<td>731DE</td>
<td>23E</td>
</tr>
<tr>
<td>731EE</td>
<td>23E</td>
</tr>
<tr>
<td>731FE</td>
<td>23E</td>
</tr>
<tr>
<td>7320E</td>
<td>23E</td>
</tr>
<tr>
<td>7321E</td>
<td>23E</td>
</tr>
<tr>
<td>7322E</td>
<td>23E</td>
</tr>
<tr>
<td>7323E</td>
<td>23E</td>
</tr>
<tr>
<td>7324E</td>
<td>23E</td>
</tr>
<tr>
<td>7325E</td>
<td>23E</td>
</tr>
<tr>
<td>7326E</td>
<td>23E</td>
</tr>
<tr>
<td>7327E</td>
<td>23E</td>
</tr>
<tr>
<td>7328E</td>
<td>23E</td>
</tr>
<tr>
<td>7329E</td>
<td>23E</td>
</tr>
<tr>
<td>732AE</td>
<td>23E</td>
</tr>
<tr>
<td>732BE</td>
<td>23E</td>
</tr>
<tr>
<td>732CE</td>
<td>23E</td>
</tr>
<tr>
<td>732DE</td>
<td>23E</td>
</tr>
<tr>
<td>732EE</td>
<td>23E</td>
</tr>
<tr>
<td>732FE</td>
<td>23E</td>
</tr>
<tr>
<td>7330E</td>
<td>23E</td>
</tr>
<tr>
<td>7331E</td>
<td>23E</td>
</tr>
<tr>
<td>7332E</td>
<td>23E</td>
</tr>
<tr>
<td>7333E</td>
<td>23E</td>
</tr>
<tr>
<td>7334E</td>
<td>23E</td>
</tr>
<tr>
<td>7335E</td>
<td>23E</td>
</tr>
<tr>
<td>7336E</td>
<td>23E</td>
</tr>
<tr>
<td>7337E</td>
<td>23E</td>
</tr>
<tr>
<td>7338E</td>
<td>23E</td>
</tr>
<tr>
<td>7339E</td>
<td>23E</td>
</tr>
<tr>
<td>733AE</td>
<td>23E</td>
</tr>
<tr>
<td>733BE</td>
<td>23E</td>
</tr>
<tr>
<td>733CE</td>
<td>23E</td>
</tr>
<tr>
<td>733DE</td>
<td>23E</td>
</tr>
<tr>
<td>733EE</td>
<td>23E</td>
</tr>
<tr>
<td>733FE</td>
<td>23E</td>
</tr>
<tr>
<td>7340E</td>
<td>23E</td>
</tr>
<tr>
<td>7341E</td>
<td>23E</td>
</tr>
<tr>
<td>7342E</td>
<td>23E</td>
</tr>
<tr>
<td>7343E</td>
<td>23E</td>
</tr>
<tr>
<td>7344E</td>
<td>23E</td>
</tr>
<tr>
<td>7345E</td>
<td>23E</td>
</tr>
<tr>
<td>7346E</td>
<td>23E</td>
</tr>
<tr>
<td>7347E</td>
<td>23E</td>
</tr>
<tr>
<td>7348E</td>
<td>23E</td>
</tr>
<tr>
<td>7349E</td>
<td>23E</td>
</tr>
<tr>
<td>734AE</td>
<td>23E</td>
</tr>
<tr>
<td>734BE</td>
<td>23E</td>
</tr>
<tr>
<td>734CE</td>
<td>23E</td>
</tr>
<tr>
<td>734DE</td>
<td>23E</td>
</tr>
<tr>
<td>734EE</td>
<td>23E</td>
</tr>
<tr>
<td>734FE</td>
<td>23E</td>
</tr>
<tr>
<td>7350E</td>
<td>23E</td>
</tr>
<tr>
<td>7351E</td>
<td>23E</td>
</tr>
<tr>
<td>7352E</td>
<td>23E</td>
</tr>
<tr>
<td>7353E</td>
<td>23E</td>
</tr>
<tr>
<td>7354E</td>
<td>23E</td>
</tr>
<tr>
<td>7355E</td>
<td>23E</td>
</tr>
<tr>
<td>7356E</td>
<td>23E</td>
</tr>
<tr>
<td>7357E</td>
<td>23E</td>
</tr>
<tr>
<td>7358E</td>
<td>23E</td>
</tr>
<tr>
<td>7359E</td>
<td>23E</td>
</tr>
<tr>
<td>735AE</td>
<td>23E</td>
</tr>
<tr>
<td>735BE</td>
<td>23E</td>
</tr>
<tr>
<td>735CE</td>
<td>23E</td>
</tr>
<tr>
<td>735DE</td>
<td>23E</td>
</tr>
<tr>
<td>735EE</td>
<td>23E</td>
</tr>
<tr>
<td>735FE</td>
<td>23E</td>
</tr>
<tr>
<td>7360E</td>
<td>23E</td>
</tr>
<tr>
<td>7361E</td>
<td>23E</td>
</tr>
<tr>
<td>7362E</td>
<td>23E</td>
</tr>
<tr>
<td>7363E</td>
<td>23E</td>
</tr>
<tr>
<td>7364E</td>
<td>23E</td>
</tr>
<tr>
<td>7365E</td>
<td>23E</td>
</tr>
<tr>
<td>7366E</td>
<td>23E</td>
</tr>
<tr>
<td>7367E</td>
<td>23E</td>
</tr>
<tr>
<td>7368E</td>
<td>23E</td>
</tr>
<tr>
<td>7369E</td>
<td>23E</td>
</tr>
<tr>
<td>736AE</td>
<td>23E</td>
</tr>
<tr>
<td>736BE</td>
<td>23E</td>
</tr>
<tr>
<td>736CE</td>
<td>23E</td>
</tr>
<tr>
<td>736DE</td>
<td>23E</td>
</tr>
<tr>
<td>736EE</td>
<td>23E</td>
</tr>
<tr>
<td>736FE</td>
<td>23E</td>
</tr>
<tr>
<td>7370E</td>
<td>23E</td>
</tr>
<tr>
<td>7371E</td>
<td>23E</td>
</tr>
<tr>
<td>7372E</td>
<td>23E</td>
</tr>
<tr>
<td>7373E</td>
<td>23E</td>
</tr>
<tr>
<td>7374E</td>
<td>23E</td>
</tr>
<tr>
<td>7375E</td>
<td>23E</td>
</tr>
<tr>
<td>7376E</td>
<td>23E</td>
</tr>
<tr>
<td>7377E</td>
<td>23E</td>
</tr>
<tr>
<td>7378E</td>
<td>23E</td>
</tr>
<tr>
<td>7379E</td>
<td>23E</td>
</tr>
<tr>
<td>737AE</td>
<td>23E</td>
</tr>
<tr>
<td>737BE</td>
<td>23E</td>
</tr>
<tr>
<td>737CE</td>
<td>23E</td>
</tr>
<tr>
<td>737DE</td>
<td>23E</td>
</tr>
<tr>
<td>737EE</td>
<td>23E</td>
</tr>
<tr>
<td>737FE</td>
<td>23E</td>
</tr>
<tr>
<td>7380E</td>
<td>23E</td>
</tr>
<tr>
<td>7381E</td>
<td>23E</td>
</tr>
<tr>
<td>7382E</td>
<td>23E</td>
</tr>
<tr>
<td>7383E</td>
<td>23E</td>
</tr>
<tr>
<td>7384E</td>
<td>23E</td>
</tr>
<tr>
<td>7385E</td>
<td>23E</td>
</tr>
<tr>
<td>7386E</td>
<td>23E</td>
</tr>
<tr>
<td>7387E</td>
<td>23E</td>
</tr>
<tr>
<td>7388E</td>
<td>23E</td>
</tr>
<tr>
<td>7389E</td>
<td>23E</td>
</tr>
<tr>
<td>738AE</td>
<td>23E</td>
</tr>
<tr>
<td>738BE</td>
<td>23E</td>
</tr>
<tr>
<td>738CE</td>
<td>23E</td>
</tr>
<tr>
<td>738DE</td>
<td>23E</td>
</tr>
<tr>
<td>738EE</td>
<td>23E</td>
</tr>
<tr>
<td>738FE</td>
<td>23E</td>
</tr>
<tr>
<td>7390E</td>
<td>23E</td>
</tr>
<tr>
<td>7391E</td>
<td>23E</td>
</tr>
<tr>
<td>7392E</td>
<td>23E</td>
</tr>
<tr>
<td>7393E</td>
<td>23E</td>
</tr>
<tr>
<td>7394E</td>
<td>23E</td>
</tr>
<tr>
<td>7395E</td>
<td>23E</td>
</tr>
<tr>
<td>7396E</td>
<td>23E</td>
</tr>
<tr>
<td>7397E</td>
<td>23E</td>
</tr>
<tr>
<td>7398E</td>
<td>23E</td>
</tr>
<tr>
<td>7399E</td>
<td>23E</td>
</tr>
<tr>
<td>739AE</td>
<td>23E</td>
</tr>
<tr>
<td>739BE</td>
<td>23E</td>
</tr>
<tr>
<td>739CE</td>
<td>23E</td>
</tr>
<tr>
<td>739DE</td>
<td>23E</td>
</tr>
<tr>
<td>739EE</td>
<td>23E</td>
</tr>
<tr>
<td>739FE</td>
<td>23E</td>
</tr>
<tr>
<td>73A0E</td>
<td>23E</td>
</tr>
<tr>
<td>73A1E</td>
<td>23E</td>
</tr>
<tr>
<td>73A2E</td>
<td>23E</td>
</tr>
<tr>
<td>73A3E</td>
<td>23E</td>
</tr>
<tr>
<td>73A4E</td>
<td>23E</td>
</tr>
<tr>
<td>73A5E</td>
<td>23E</td>
</tr>
<tr>
<td>73A6E</td>
<td>23E</td>
</tr>
<tr>
<td>73A7E</td>
<td>23E</td>
</tr>
<tr>
<td>73A8E</td>
<td>23E</td>
</tr>
<tr>
<td>73A9E</td>
<td>23E</td>
</tr>
<tr>
<td>73AAE</td>
<td>23E</td>
</tr>
<tr>
<td>73ABE</td>
<td>23E</td>
</tr>
<tr>
<td>73ACF</td>
<td>23E</td>
</tr>
<tr>
<td>73ADF</td>
<td>23E</td>
</tr>
<tr>
<td>73AEE</td>
<td>23E</td>
</tr>
<tr>
<td>73AFE</td>
<td>23E</td>
</tr>
<tr>
<td>73B0F</td>
<td>23E</td>
</tr>
<tr>
<td>73B1F</td>
<td>23E</td>
</tr>
<tr>
<td>73B2F</td>
<td>23E</td>
</tr>
<tr>
<td>73B3F</td>
<td>23E</td>
</tr>
<tr>
<td>73B4F</td>
<td>23E</td>
</tr>
<tr>
<td>73B5F</td>
<td>23E</td>
</tr>
<tr>
<td>73B6F</td>
<td>23E</td>
</tr>
<tr>
<td>73B7F</td>
<td>23E</td>
</tr>
<tr>
<td>73B8F</td>
<td>23E</td>
</tr>
<tr>
<td>73B9F</td>
<td>23E</td>
</tr>
<tr>
<td>73BAF</td>
<td>23E</td>
</tr>
<tr>
<td>73BDF</td>
<td>23E</td>
</tr>
<tr>
<td>73C0F</td>
<td>23E</td>
</tr>
<tr>
<td>73C1F</td>
<td>23E</td>
</tr>
<tr>
<td>73C2F</td>
<td>23E</td>
</tr>
<tr>
<td>73C3F</td>
<td>23E</td>
</tr>
<tr>
<td>73C4F</td>
<td>23E</td>
</tr>
<tr>
<td>73C5F</td>
<td>23E</td>
</tr>
<tr>
<td>73C6F</td>
<td>23E</td>
</tr>
<tr>
<td>73C7F</td>
<td>23E</td>
</tr>
<tr>
<td>73C8F</td>
<td>23E</td>
</tr>
<tr>
<td>73C9F</td>
<td>23E</td>
</tr>
<tr>
<td>73CAF</td>
<td>23E</td>
</tr>
<tr>
<td>73CFF</td>
<td>23E</td>
</tr>
</tbody>
</table>

SYMBOL TABLE UNSIGNED INTEGER

CONSTANT ENTRY

FIGURE 4.
<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H</td>
<td>\ COLLISION</td>
</tr>
<tr>
<td>02H</td>
<td>\ ADDRESS</td>
</tr>
<tr>
<td>02H</td>
<td>\ PREVIOUS</td>
</tr>
<tr>
<td>05H</td>
<td>\ ENTRY ADDRESS</td>
</tr>
<tr>
<td>10H</td>
<td>FORM</td>
</tr>
<tr>
<td>20H</td>
<td>HASH</td>
</tr>
<tr>
<td>04H</td>
<td>PRINTLENGTH</td>
</tr>
<tr>
<td>04H</td>
<td>ASCII CHARACTER P</td>
</tr>
<tr>
<td>04H</td>
<td>ASCII CHARACTER O</td>
</tr>
<tr>
<td>04H</td>
<td>ASCII CHARACTER I</td>
</tr>
<tr>
<td>04H</td>
<td>ASCII CHARACTER L</td>
</tr>
<tr>
<td>04H</td>
<td>STRING LENGTH</td>
</tr>
<tr>
<td>04H</td>
<td>ASCII CHARACTER P</td>
</tr>
<tr>
<td>04H</td>
<td>ASCII CHARACTER O</td>
</tr>
<tr>
<td>04H</td>
<td>ASCII CHARACTER I</td>
</tr>
<tr>
<td>04H</td>
<td>ASCII CHARACTER L</td>
</tr>
</tbody>
</table>

SYMBOL TABLE STRING CONSTANT ENTRY

FIGURE 5.
c. Type entries

WPS-PASCAL has two kinds of type entries in its symbol table: simple type entries and type declaration entries. The simple type entry can either be one of WPS-PASCAL's standard types, or a previously defined simple type declaration (scalar or subrange). In the latter case, a simple type entry is made in the symbol table, with a pointer to the scalar or subrange type declaration entry. In the former case, one of the following standard types will be assigned to the type entry.

Integer - The values of this type are a subset of the whole numbers whose range is the set of values:

\[-\text{maxint}, -\text{maxint}+1, \ldots, -1, 0, 1, \ldots, \text{maxint}-1, \text{maxint}\]

where \(\text{maxint} = 32,767\).

Real - The values are a subset of the real numbers consisting of fourteen digits multiplied by ten to the \(-64\)th power through ten to the \(+53\)rd power.

Boolean - The values are denoted by the identifiers "false" and "true", such that false is less than true.

Character - The values of this type are the defined set of characters described in Pef F. The following relationships hold for character types:
(1) The subset of character values representing the limits A through Z is ordered and continuous.

(2) The subset of character values representing the upper case letters A through Z is ordered and continuous.

(3) The subset of character values representing the lower case letters a through z is ordered and continuous.

Type declarations entries, however, are generated from user-defined types found elsewhere in the source program. It is possible to define a chain of type declarations. An example would be an array of the type array which is itself of type integer.

The symbol table entry for a type is as follows. An integer type has the FORM value of 42H, a real type has the FORM value of 44H, a character type has the FORM value of 52H, and a boolean type has the FORM value of 5AH. A FORM value of 7AH indicates that an additional type declaration entry must be accessed. The field following the FORM is a one-byte field containing the hash value of the printname. The next byte contains the printname's length, which is followed by the printname characters of the type identifier. The last two bytes contain the address of the specified type. Examples of simple type entries are shown in Figs. 6 - 9.
TYPE MY = INTEGER;

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>7322H</td>
<td>CE</td>
</tr>
<tr>
<td>7323H</td>
<td>DF</td>
</tr>
<tr>
<td>7324H</td>
<td>16H</td>
</tr>
<tr>
<td>7325H</td>
<td>73E</td>
</tr>
<tr>
<td>7326H</td>
<td>49H</td>
</tr>
<tr>
<td>7327H</td>
<td>70H</td>
</tr>
<tr>
<td>7328H</td>
<td>33F</td>
</tr>
<tr>
<td>7329H</td>
<td>49H</td>
</tr>
<tr>
<td>732AH</td>
<td>55H</td>
</tr>
<tr>
<td>732BH</td>
<td>49H</td>
</tr>
<tr>
<td>732CH</td>
<td>96H</td>
</tr>
<tr>
<td>732DH</td>
<td>21H</td>
</tr>
</tbody>
</table>

SYMBOL TABLE SIMPLE TYPE INTEGER ENTRY

FIGURE 6.
<table>
<thead>
<tr>
<th>Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>7322H</td>
<td>80H</td>
</tr>
<tr>
<td>7323H</td>
<td>32H</td>
</tr>
<tr>
<td>7324H</td>
<td>18H</td>
</tr>
<tr>
<td>7325H</td>
<td>78H</td>
</tr>
<tr>
<td>7326H</td>
<td>32H</td>
</tr>
<tr>
<td>7327H</td>
<td>78H</td>
</tr>
<tr>
<td>7328H</td>
<td>03H</td>
</tr>
<tr>
<td>7329H</td>
<td>4FH</td>
</tr>
<tr>
<td>732AH</td>
<td>55H</td>
</tr>
<tr>
<td>732BH</td>
<td>48H</td>
</tr>
<tr>
<td>732CH</td>
<td>19H</td>
</tr>
<tr>
<td>732DH</td>
<td>01H</td>
</tr>
</tbody>
</table>

SYMBOL TABLE SIMPLE TYPE ENTRY

FIGURE 7.
<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>7322H</td>
<td>22H</td>
</tr>
<tr>
<td>7323H</td>
<td>23H</td>
</tr>
<tr>
<td>7324H</td>
<td>1EH</td>
</tr>
<tr>
<td>7325H</td>
<td>73H</td>
</tr>
<tr>
<td>7326H</td>
<td>2AH</td>
</tr>
<tr>
<td>7327H</td>
<td>23H</td>
</tr>
<tr>
<td>7328H</td>
<td>23H</td>
</tr>
<tr>
<td>7329H</td>
<td>4FH</td>
</tr>
<tr>
<td>732AH</td>
<td>55H</td>
</tr>
<tr>
<td>732BH</td>
<td>4FH</td>
</tr>
<tr>
<td>732CH</td>
<td>2AH</td>
</tr>
<tr>
<td>732DH</td>
<td>21H</td>
</tr>
</tbody>
</table>

SYMBOL TABLE SIMPLE TYPE ENTRY

FIGURE 5.
<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>7322H</td>
<td>32H</td>
</tr>
<tr>
<td></td>
<td>\ COLLISION</td>
</tr>
<tr>
<td>7323H</td>
<td>32H</td>
</tr>
<tr>
<td></td>
<td>\ ADDRESS</td>
</tr>
<tr>
<td>7324H</td>
<td>1EH</td>
</tr>
<tr>
<td></td>
<td>\ PREVIOUS SETL</td>
</tr>
<tr>
<td>7325H</td>
<td>73E</td>
</tr>
<tr>
<td></td>
<td>\ ENTRY ADDRESS</td>
</tr>
<tr>
<td>7326H</td>
<td>4AH</td>
</tr>
<tr>
<td></td>
<td>FCTY</td>
</tr>
<tr>
<td>7327H</td>
<td>70E</td>
</tr>
<tr>
<td></td>
<td>FASH</td>
</tr>
<tr>
<td>7328H</td>
<td>33E</td>
</tr>
<tr>
<td></td>
<td>PRINTNAME LENGTH</td>
</tr>
<tr>
<td>7329H</td>
<td>4FE</td>
</tr>
<tr>
<td></td>
<td>ASCII CHARACTER N</td>
</tr>
<tr>
<td>732AH</td>
<td>65H</td>
</tr>
<tr>
<td></td>
<td>ASCII CHARACTER U</td>
</tr>
<tr>
<td>732BH</td>
<td>4DF</td>
</tr>
<tr>
<td></td>
<td>ASCII CHARACTER M</td>
</tr>
<tr>
<td>732CH</td>
<td>14H</td>
</tr>
<tr>
<td></td>
<td>SETEL ADDRESS OF</td>
</tr>
<tr>
<td>732DH</td>
<td>21H</td>
</tr>
<tr>
<td></td>
<td>PAREN TYPE</td>
</tr>
</tbody>
</table>

**SYMBOL TABLE SIMPLE TYPE ENTRY**

**FIGURE 9.**
There are seven different user definable types in NPS-PASCAL. A type declaration entry is constructed whenever a scalar type, subrange type, array type, record type, set type, file type, or pointer type is encountered.

(1) Scalar Types. By definition, a scalar type is an ordered set of values whose identifiers are enumerated to denote their values. The first field entry for scalar types has the value MP. Scalar entries are the only type declaration entries that have an accessible printname. Consequently, the next two fields hold the printname case value and length. The printname characters follow these fields. The next field is a byte value containing the enumerated value of the scalar identifier. The enumerated values (0, 1, 2..., ) are assigned to the scalars in the order in which they appear in the declaration. The final field is a two byte field storing the symbol table address of the parent type. The scalar type entry will be pointed to by the variable entry claiming this type. An example of a scalar type entry is presented in Fig. 10.

(2) Subrange Types. A subrange type is a duplicate declaration of any other previously defined scalar type, integer type, or character type, but with a specified lower and upper bound on its elements. The first field of a subrange entry is mp for enumerated elements, mp for integer elements, and mp for character elements. Bytes six and seven store the address of the subrange element's parent type. Bytes eight and nine hold the low value of the range.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\backslash</td>
<td>COLON</td>
</tr>
<tr>
<td>\backslash</td>
<td>AT_EOF</td>
</tr>
<tr>
<td>\backslash</td>
<td>PREVIOUS_STEP</td>
</tr>
<tr>
<td>\backslash</td>
<td>ENTRY_ADDRESS</td>
</tr>
<tr>
<td>FORM</td>
<td></td>
</tr>
<tr>
<td>MAIN</td>
<td></td>
</tr>
<tr>
<td>PRINTABLE_LENGTH</td>
<td></td>
</tr>
<tr>
<td>ASCII_CHARACTER_C</td>
<td></td>
</tr>
<tr>
<td>ASCII_CHARACTER_D</td>
<td></td>
</tr>
<tr>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td>SETB_ADDRESS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SYMBOL TABLE SIMPLE TYPE ENTRY (SCALAP)**

**FIGURE 10.**
while the next two bytes contain the high value of the range. The following field is two bytes long and stores the total number of elements in the range. The displacement vector is not stored with the subrange, since any given subrange could serve as the index to arrays of different base types. The displacement vector is stored instead with the array entry itself. This entry will be pointed to by a variable entry claiming this type. An example of a subrange type entry is shown in Fig. 11.

(3) ARRAY TYPES. The preceding two type declaration entries in WPS-PASCAL are called simple type entries. They are symbol table entries using a single, predefined type. Structured types are compositions of types. In other words, one or more types are used to describe a single symbol table entry. A structured type will have a type declaration entry which contains the printname, and which points to the structure type entry.

The array type is a structured type consisting of a fixed number of components that are all of the same type, called the component type. The number of components is specified as a scalar or subrange type and is referred to as the index type. INTEGFR and REAL types are not legal index types; however, the scalar or subrange type can be of the type integer.

The symbol table format for an array entry has the form value of 17H. The following byte specifies the number of indices, or dimensions in the array. The next two
TYPE PRINT = RED...BLUE;

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>73A2E</td>
<td>02E</td>
<td>\ COLLISION</td>
</tr>
<tr>
<td>73A3E</td>
<td>02F</td>
<td>\ ADDRESS</td>
</tr>
<tr>
<td>73A4E</td>
<td>04E</td>
<td>\ PREVIOUS SUBRANGE</td>
</tr>
<tr>
<td>73A5E</td>
<td>72E</td>
<td>\ ENTRY ADDRESS</td>
</tr>
<tr>
<td>73A6E</td>
<td>27H</td>
<td>\ FORM</td>
</tr>
<tr>
<td>73A7E</td>
<td>27H</td>
<td>\ SUBLINK ADDRESS OF</td>
</tr>
<tr>
<td>73A8E</td>
<td>73H</td>
<td>\ PARENT TYPE</td>
</tr>
<tr>
<td>73A9E</td>
<td>27H</td>
<td>\ SUBRANGE</td>
</tr>
<tr>
<td>73AAE</td>
<td>02E</td>
<td>\ LOW VALUE</td>
</tr>
<tr>
<td>73ABE</td>
<td>02E</td>
<td>\ SUBRANGE</td>
</tr>
<tr>
<td>73ACE</td>
<td>0CE</td>
<td>\ HIGH VALUE</td>
</tr>
<tr>
<td>73ADE</td>
<td>23E</td>
<td>\ NUMBER OF ELEMENTS</td>
</tr>
</tbody>
</table>

SYMBOL TABLE SUBRANGE TYPE ENTRY

FIGURE 11.

71
fields are both two bytes long, the first containing the address of the component type; the second containing the total storage requirement for the array in bytes. The eleventh byte of the entry holds a value designating the type of the array's component as defined in Table 2. A two byte field follows with the symbol table address of the type entry of the array's first dimension. This is followed by a two byte field which contains the displacement vector for this dimension. The displacement vector for each dimension represents the distance in bytes between two elements of the array which have a difference of one in the corresponding subscript. If the array has more than one dimension, four more bytes are allotted in the symbol table to store the address and displacement vector of each additional dimension. This entry will be pointed to by the variable entry claiming this type. An example of an array type entry is shown in Fig. 12.

(4) Record Types. A record is another FIRE-PASCAL structured type. This structure has a fixed number of components, called fields, each of which can be of any defined type. The symbol table entry for a record has the form field value of 1FE. Bytes six and seven contain the storage requirements in bytes for the entire record. Bytes eight and nine store the symbol table address of the type entry of the last field contained in the record structure. The remaining field entries are located by chaining backward to the parent record entry via the previous symbol table.

37
**BASIC TYPE OF COMPONENTS**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning (Type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>02F</td>
<td>Ordinate</td>
</tr>
<tr>
<td>01T</td>
<td>Integer</td>
</tr>
<tr>
<td>OPY</td>
<td>Character</td>
</tr>
<tr>
<td>23F</td>
<td>Real</td>
</tr>
<tr>
<td>24T</td>
<td>Complex</td>
</tr>
<tr>
<td>25F</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

**TABLE 2.**
<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7300H</td>
<td>02H</td>
<td>C Collisions</td>
</tr>
<tr>
<td>7301H</td>
<td>06H</td>
<td>Address</td>
</tr>
<tr>
<td>7302H</td>
<td>16H</td>
<td>Previous symbol</td>
</tr>
<tr>
<td>7303H</td>
<td>73H</td>
<td>Entry address</td>
</tr>
<tr>
<td>7304H</td>
<td>17H</td>
<td>Form</td>
</tr>
<tr>
<td>7305H</td>
<td>01H</td>
<td>Number of dimensions</td>
</tr>
<tr>
<td>7306H</td>
<td>26H</td>
<td>Symbol address of symbol</td>
</tr>
<tr>
<td>7307H</td>
<td>77H</td>
<td>Component type</td>
</tr>
<tr>
<td>7308H</td>
<td>06H</td>
<td>Total storage required</td>
</tr>
<tr>
<td>7309H</td>
<td>22H</td>
<td>In bytes</td>
</tr>
<tr>
<td>730AH</td>
<td>22H</td>
<td>Type of component</td>
</tr>
<tr>
<td>730BH</td>
<td>21H</td>
<td>Array</td>
</tr>
<tr>
<td>730CH</td>
<td>20H</td>
<td>Offset</td>
</tr>
<tr>
<td>730DH</td>
<td>14H</td>
<td>Symbol address of symbol</td>
</tr>
<tr>
<td>730EH</td>
<td>72H</td>
<td>First dimension</td>
</tr>
<tr>
<td>730FH</td>
<td>44H</td>
<td>Displacement vector</td>
</tr>
<tr>
<td>7300H</td>
<td>03H</td>
<td>Of first dimension</td>
</tr>
</tbody>
</table>

Symbol Table Array Type Entry

Figure 12.
entry address. An example of a record type entry is shown in Fig. 13.

Each record field consists of an identifier and a type. The form field of a record entry has a value of 5FH. The following two fields are bytes for the hash and the length of the printname. The next field holds the printname characters. The address of the parent record is stored in the next two bytes. The following field has a one byte length and is used to store the record field’s type. The value stored is also taken from Table 2. Two more bytes are used to store the symbol table address of the type just indicated. The last field of this entry is two bytes long and holds the offset of the record field from the record base.

NFS-PASCAL supports the variant field and tag field constructs of records. These two kinds of record fields have symbol table entries similar to the one described above for fields, with the exception of the form field, which is IFH for variant fields, and 9FH for tag fields. An example of a field entry is shown in Fig. 14.

(5) Set Types. The set structure defines a set of values which is the power set of a declared base type. The base type is required to be a scalar or subrange type. The set type symbol table entry has a form field value of 27H. The following two bytes contain the symbol table address of the set type identifier. An example of a set type entry is shown in Fig. 15.
<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>738EH</td>
<td>28H</td>
</tr>
<tr>
<td>739CH</td>
<td>28H</td>
</tr>
<tr>
<td>73D6H</td>
<td>7EH</td>
</tr>
<tr>
<td>729EH</td>
<td>7EH</td>
</tr>
<tr>
<td>728EH</td>
<td>1EH</td>
</tr>
<tr>
<td>709CH</td>
<td>12H</td>
</tr>
<tr>
<td>7291F</td>
<td>22H</td>
</tr>
<tr>
<td>7092H</td>
<td>44H</td>
</tr>
<tr>
<td>7092H</td>
<td>7CH</td>
</tr>
</tbody>
</table>

**SYMBOL TABLE RECORD TYPE FN19Y**

**FIGURE 12.**
```plaintext
<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>7294H</td>
<td>49H</td>
</tr>
<tr>
<td>7295H</td>
<td>02H</td>
</tr>
<tr>
<td>7296H</td>
<td>2FH</td>
</tr>
<tr>
<td>7297H</td>
<td>73H</td>
</tr>
<tr>
<td>7298E</td>
<td>5FE</td>
</tr>
<tr>
<td>7026H</td>
<td>17H</td>
</tr>
<tr>
<td>7299H</td>
<td>22E</td>
</tr>
<tr>
<td>729EH</td>
<td>52H</td>
</tr>
<tr>
<td>729FH</td>
<td>45H</td>
</tr>
<tr>
<td>72A0H</td>
<td>83H</td>
</tr>
<tr>
<td>72A1H</td>
<td>73H</td>
</tr>
<tr>
<td>72A2H</td>
<td>63H</td>
</tr>
<tr>
<td>72A3H</td>
<td>14H</td>
</tr>
<tr>
<td>72A4H</td>
<td>21H</td>
</tr>
<tr>
<td>72A5H</td>
<td>00H</td>
</tr>
<tr>
<td>72A6H</td>
<td>00H</td>
</tr>
</tbody>
</table>
```

SYMBOL TABLE RECORD FILE ENTRY

FIGURE 14.
**TYPE FLAG=SET OF CLCPS**

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>73BEH</td>
<td>06H</td>
</tr>
<tr>
<td>73B9H</td>
<td>02H</td>
</tr>
<tr>
<td>73FAE</td>
<td>D9H</td>
</tr>
<tr>
<td>73F9H</td>
<td>74E</td>
</tr>
<tr>
<td>73FCH</td>
<td>27H</td>
</tr>
<tr>
<td>73FCH</td>
<td>2FE</td>
</tr>
<tr>
<td>73FCH</td>
<td>77E</td>
</tr>
</tbody>
</table>

SYMBOL TABLE SET TYPE ENTRY

**FIGURE 13.**
File Types. A file type consists of a sequence of components, all of the same type, is called a file. A file type indicates a natural ordering of the components, whose position in the file defines the sequence. A file type declaration entry in the symbol table has a form field value of FFH. The symbol table address of the file type's identifier is contained in the next two bytes. An example of a file type entry is shown in Fig. 16.

Pointer Types. NPS-PASCAL supports dynamic variables which are generated without any correlation to the static structure of the program. These variables are assigned a special type called pointer type. The form field value is set to 3FFH, while bytes six and seven hold the symbol table address of the pointer type's parent entry. An example of a pointer type entry is shown in Fig. 17.

d. Variable Entries

Each variable declared in an NPS-PASCAL program is inserted into the symbol table. The form field of the variable entry contains a value which describes the type of the variable. The values for this field and the associated types are shown in Table 1. Following the form field are the fields containing the variable identifier's printname, text value, length, and the printname characters. A two-byte field which contains the variable's starting address in memory appears after the printname characters. This address is an offset from the base of the variable area, called the
TYPE DATA=FILE OF HW:

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>737EH</td>
<td>00H</td>
</tr>
<tr>
<td>737EH</td>
<td>00H</td>
</tr>
<tr>
<td>7402H</td>
<td>0FH</td>
</tr>
<tr>
<td>7401H</td>
<td>78H</td>
</tr>
<tr>
<td>7402H</td>
<td>27H</td>
</tr>
<tr>
<td>7403H</td>
<td>22H</td>
</tr>
<tr>
<td>7404H</td>
<td>78H</td>
</tr>
<tr>
<td>7400H</td>
<td>00H</td>
</tr>
</tbody>
</table>

\* GTN SVA: FILE: ADBS

SYMBOL TABLE FILE TYPE ENTRY

FIGURE 16.
<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>7472H</td>
<td>26H</td>
<td>COLLISION</td>
</tr>
<tr>
<td>7471H</td>
<td>22H</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>7470H</td>
<td>21H</td>
<td>PREVIOUS SETEL</td>
</tr>
<tr>
<td>7469H</td>
<td>74H</td>
<td>ENTRY ADDRESS</td>
</tr>
<tr>
<td>7464H</td>
<td>37H</td>
<td>FORM</td>
</tr>
<tr>
<td>7463H</td>
<td>94H</td>
<td>SETEL ADDRESS OF</td>
</tr>
<tr>
<td>7462H</td>
<td>7CH</td>
<td>POINTER TYPE</td>
</tr>
</tbody>
</table>

SYMBOL TABLE POINTER TYPE ENTRY

FIGURE 17.
Program Reference Table (PRT), which address is assigned by the NPS-PASCAL code generator. The variable's type determines the number of bytes assigned to store the variable in the PRT. The compiler keeps a running total of the amount of storage assigned to all variables, and includes this value in the pseudo code at the completion of a successful program compilation. The interpreter/translator subsequently converts the relative addresses in the intermediate code to absolute address in the final target machine. Next is a two byte field which contains the SHTBL address of the variable's type. In the case of the standard Pascal types integer (FORM = $0B$E), real ($13$E), character ($13$E) and boolean ($23$E), this is the address of that type in the BUILTIN$STBL$. In the case of integer and character subranges ($27$E), this field contains the address of the subrange type entry. In the case of a scalar ($23$E), this field contains the address of the last of a series of scalar ($27$E) entries. The remaining scalar entries are located by chaining backward to the variable entry via the previous symbol table entry address. If the variable is a complex declaration, (array, record, set, file or pointer), this field contains the address of the complex type's entry in the symbol table. If the variable is of a type previously defined in the program, this field contains a pointer to that type declaration. Examples of variable entries are shown in Figs. 18 - 20.
VAR X:INTEGER;

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0E0Fh</td>
<td>00H          \  COLLISION</td>
</tr>
<tr>
<td>0E0Ch</td>
<td>00H          /  ADDRESS</td>
</tr>
<tr>
<td>0E0Dh</td>
<td>03H          \  PREVIOUS SETBL</td>
</tr>
<tr>
<td>0E0Eh</td>
<td>02H          /  ENTRY ADDRESS</td>
</tr>
<tr>
<td>0E0Fh</td>
<td>03H          /  FORM</td>
</tr>
<tr>
<td>0E10h</td>
<td>08H          /  EASE</td>
</tr>
<tr>
<td>0E11h</td>
<td>01H          /  PRINTNAME LENGTH</td>
</tr>
<tr>
<td>0E12h</td>
<td>08H          /  ASCII CHARACTER X</td>
</tr>
<tr>
<td>0E13h</td>
<td>03H          \  PRT LOCATION</td>
</tr>
<tr>
<td>0E14h</td>
<td>02H          /  ASSIGNED</td>
</tr>
<tr>
<td>0E15h</td>
<td>06H          \  SYMBOL TABLE ADDRESS</td>
</tr>
<tr>
<td>0E16h</td>
<td>01H          /  OF PARENT TYPE</td>
</tr>
</tbody>
</table>

SYMBOL TABLE VARIABLE ENTRY (INTEGER)

FIGURE 18.
VAR CP:(PLUS,MIXUS,TIMES);

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>7014H</td>
<td>02H</td>
</tr>
<tr>
<td>7015H</td>
<td>00H</td>
</tr>
<tr>
<td>7016H</td>
<td>07H</td>
</tr>
<tr>
<td>7017H</td>
<td>08H</td>
</tr>
<tr>
<td>7018H</td>
<td>03H</td>
</tr>
<tr>
<td>7019H</td>
<td>1FE</td>
</tr>
<tr>
<td>701AH</td>
<td>02H</td>
</tr>
<tr>
<td>701BH</td>
<td>4FH</td>
</tr>
<tr>
<td>701CH</td>
<td>50H</td>
</tr>
<tr>
<td>701DH</td>
<td>08H</td>
</tr>
<tr>
<td>701EH</td>
<td>32H</td>
</tr>
<tr>
<td>701FH</td>
<td>3EH</td>
</tr>
<tr>
<td>7020H</td>
<td>06H</td>
</tr>
</tbody>
</table>

SYMBOL TABLE VARIABLE ENTRY (SCALAR)

FIGURE 19.
VAR A:ARRAY[1..5] OF INTEGER;

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>7721E</td>
<td>20E</td>
</tr>
<tr>
<td></td>
<td>\ COLLISION</td>
</tr>
<tr>
<td>7722E</td>
<td>20E</td>
</tr>
<tr>
<td></td>
<td>\ ADDRESS</td>
</tr>
<tr>
<td>7723H</td>
<td>14H</td>
</tr>
<tr>
<td></td>
<td>\ PREVIOUS SETPL</td>
</tr>
<tr>
<td>7724E</td>
<td>77E</td>
</tr>
<tr>
<td></td>
<td>\ ENTRY ADDRESS</td>
</tr>
<tr>
<td>7725E</td>
<td>23H</td>
</tr>
<tr>
<td></td>
<td>FORM</td>
</tr>
<tr>
<td>7726H</td>
<td>41H</td>
</tr>
<tr>
<td></td>
<td>HASH</td>
</tr>
<tr>
<td>7727E</td>
<td>01H</td>
</tr>
<tr>
<td></td>
<td>PRINTNAME LENGTH</td>
</tr>
<tr>
<td>7728E</td>
<td>41H</td>
</tr>
<tr>
<td></td>
<td>ASCII CHARACTER A</td>
</tr>
<tr>
<td>7729H</td>
<td>03H</td>
</tr>
<tr>
<td></td>
<td>\ PRT LOCATION</td>
</tr>
<tr>
<td>772AH</td>
<td>20E</td>
</tr>
<tr>
<td></td>
<td>\ ASSIGNED</td>
</tr>
<tr>
<td>772EE</td>
<td>3AE</td>
</tr>
<tr>
<td></td>
<td>\ SETEL ADDRESS</td>
</tr>
<tr>
<td>772CE</td>
<td>77H</td>
</tr>
<tr>
<td></td>
<td>OF ARRAY TYPE ENTRY</td>
</tr>
</tbody>
</table>

SYMBOL TABLE VARIABLE ENTRY (COMPLEX)

FIGURE 20.
Procedure and Function Entries

Every procedure and function in an NPS-PASCAL program has an associated entry in the symbol table. In the case of a procedure entry, the form field is assigned the value 043. The hash value, length of the printname, and the printname characters immediately follow the form field. A one byte field follows and stores the number of parameters associated with the procedure. A two byte field is next, storing the symbol table location of a listing of the procedure's parameter types. This listing is referenced by the compiler to ensure proper mapping, and is located immediately after the final procedure entry in the symbol table. Following the parameter type's address field in the procedure entry are three more two byte fields. The first field gives the PRT address assigned to the procedure identifier. The second field gives the PRT address assigned to the procedure save block pointer (S3P). The S3P permits recursive subroutine calls, and will be explained in the section on Code Generation. The final field in the entry holds a label value that must be branched to when the procedure is invoked. An example of a procedure entry is shown in Fig. 21.

A function entry in the symbol table duplicates a procedure entry with two exceptions. A function entry has a form field value of 05E; and one byte field is added at the end of the entry to designate the type of the function.
**PROCEEDURE LC (A: INTEGER; Y: INTEGER):**

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>746EH</td>
<td>02H</td>
</tr>
<tr>
<td>746FH</td>
<td>26H</td>
</tr>
<tr>
<td>746FH</td>
<td>60H</td>
</tr>
<tr>
<td>7470EH</td>
<td>74H</td>
</tr>
<tr>
<td>7471H</td>
<td>04H</td>
</tr>
<tr>
<td>7472H</td>
<td>1BH</td>
</tr>
<tr>
<td>7473H</td>
<td>02H</td>
</tr>
<tr>
<td>7474H</td>
<td>4CH</td>
</tr>
<tr>
<td>7475EH</td>
<td>4FH</td>
</tr>
<tr>
<td>7476H</td>
<td>02H</td>
</tr>
<tr>
<td>7477EH</td>
<td>97H</td>
</tr>
<tr>
<td>7478EH</td>
<td>74H</td>
</tr>
<tr>
<td>7479EH</td>
<td>22H</td>
</tr>
<tr>
<td>747AH</td>
<td>00H</td>
</tr>
<tr>
<td>747BEH</td>
<td>26H</td>
</tr>
<tr>
<td>747CE</td>
<td>29H</td>
</tr>
<tr>
<td>747DFH</td>
<td>01H</td>
</tr>
<tr>
<td>747EH</td>
<td>30H</td>
</tr>
</tbody>
</table>

\texttt{SYMBOL TABLE PROCEDURE} \texttt{ENTRY}

\texttt{FIGURE 21.}
Function type assignments are also taken from Table 2. An example of a function entry is shown in Fig. 22.

(1) **Formal Parameters.** Formal parameters provide a mechanism that allows a procedure or function to be repeated with various values being substituted. The formal parameters are declared in the procedure or function declaration and can be of four types: value parameters, variable parameters, procedure parameters and function parameters. Each declared parameter has an associated symbol table entry. A value parameter entry has exactly the same format as the variable entry. A variable parameter entry also duplicates a variable symbol table entry, with the exception of the form field. The high order bit of the form field is set to one for all variable parameters. Procedure and function parameters are entered as described above for procedure and function symbol table entries.

Figure 23 illustrates a sample series of symbol table entries with a procedure entry followed by various formal parameter entries. Note that the final few bytes show the listing of the procedure's parameter types that will be utilized for mapping actual parameters into the formal parameters.

**E. PARSER**

The parser is a table driven automaton and is modelled after the ALCOL-M [7]. The LALF( ) parser generator [6] produced the required parse tables and the vocabulary table, VOCAB. The parser operates by receiving tokens from the
FUNCTION Y7(F:A:REAL):REAL;

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>746DH</td>
<td>02H</td>
</tr>
<tr>
<td>746EH</td>
<td>02H</td>
</tr>
<tr>
<td>746FH</td>
<td>06H</td>
</tr>
<tr>
<td>7470H</td>
<td>04H</td>
</tr>
<tr>
<td>7471H</td>
<td>05H</td>
</tr>
<tr>
<td>7471E</td>
<td>05H</td>
</tr>
<tr>
<td>7471F</td>
<td>03H</td>
</tr>
<tr>
<td>7472H</td>
<td>02H</td>
</tr>
<tr>
<td>7473H</td>
<td>02H</td>
</tr>
<tr>
<td>7474H</td>
<td>59H</td>
</tr>
<tr>
<td>7475H</td>
<td>59H</td>
</tr>
<tr>
<td>7476E</td>
<td>22H</td>
</tr>
<tr>
<td>7477H</td>
<td>0AH</td>
</tr>
<tr>
<td>7478H</td>
<td>74H</td>
</tr>
<tr>
<td>7479E</td>
<td>22H</td>
</tr>
<tr>
<td>747AE</td>
<td>00H</td>
</tr>
<tr>
<td>747BE</td>
<td>46H</td>
</tr>
<tr>
<td>747CH</td>
<td>22H</td>
</tr>
<tr>
<td>747DE</td>
<td>07H</td>
</tr>
<tr>
<td>747EH</td>
<td>06H</td>
</tr>
<tr>
<td>747FH</td>
<td>1BE</td>
</tr>
</tbody>
</table>

SYMBOL TABLE FUNCTION ENTRY

FIGURE 22.
PROCEDURE AND PARAMETERS SYMBOL TABLE ENTRY

FIGURE 23.
scanner, analyzing them to determine if they are a part of the NPS-PASCAL grammar, then accepts or rejects the token according to the grammar. If the token is accepted, one of two actions is taken. The parser may stack the token and continue to request tokens in the lookahead state, or it may recognize the right part of a valid production and apply the production state. This results in a stack reduction. If the parser rejects the token, or determines that the token received does not constitute a valid right part of any production in the grammar, a syntax error message will be printed to the console and the RECOVER procedure is called.

RECOVER is a procedure that permits continued program compilation in spite of the detection of a syntax error. The parser backs up one state and attempts to continue parsing from that state. In the event of failure, the parser continues to back up until the end of the currently pending production is located. At that point the invalid token is completely bypassed, and an attempt is made to parse the following token. This process continues until an acceptable token is found.

The parse stacks in NPS-PASCAL consist of a state stack and eight auxiliary stacks. The auxiliary stacks are parallel to the parse stack and are used to store information extracted from the symbol table needed during code generation. The stacks are:
BASELOC - stores the symbol table address of the current identifier;

FORMFIELD - stores the form field value of the current identifier as reflected in the symbol table;

TYPESTACK - stores the type value of the identifier;

PPT$ADD - stores the PPT address of the identifier;

LABELSTACK - stores the label value to be used with branching instructions;

PAR$NUM - stores the number of formal parameters associated with a procedure or function;

PAR$NUMLOC - stores the symbol table address of the list of formal parameter types associated with a procedure or function;

EXPR$STK - stores the type value of an expression.

F. CODE GENERATION

The parser not only verifies the syntax of the source statements, but also controls the generation of the intermediate code by associating semantic actions with production rules. When a reduction takes place, the SYNTESIZE procedure (in SYNTH2.SRC) is called with the production number as a parameter. The SYNTESIZE procedure contains an extensive case statement keyed by the production number to perform the appropriate semantic actions. The syntax of the language and the semantic actions for each reduction are contained within the listing of the module SYNTH2.SRC.
Fundamental to understanding the compiler is a detailed knowledge of the MPS-PASCAL data structures, the pseudo operators, the use of procedures and functions, and the communication paths between the compiler and the user. These pseudo operators are described in detail in Ref. 1. These other elements are described below to assist in understanding the MPS-PASCAL compiler constructs and to explain the logic used to generate the intermediate code. That code will later be used to generate the target machine code.

1. Storage Space Allocation

The amount of storage allocated to a variable is a function of the type of the item. For each proper variable requiring storage space, the compiler specifies the number of bytes to be allotted, and keeps a running total of the number of bytes assigned. The total count is then passed to the code generator to establish the size of the Program Reference Table (PRT).

a. Byte Data

Byte data items are stored in a single byte in memory. Byte data items can represent characters, numbers, or boolean variables.

b. Integer Data

Integers are represented by two byte locations in memory, with the high order byte preceding the low order byte of the integer number. The storage design imitates the function of the 8080A microprocessor [5] in its movement of...
data from memory or from the stack into the processor's double byte registers during program execution. Integers are represented in two's complement form, with the high order bit acting as the sign bit. A zero high order bit indicates a positive integer, while a high order bit of one indicates a negative number.

c. Real Data

Real numbers are represented in binary coded decimal (BCD) format. Each real number is represented by fourteen decimal digits and is stored in eight consecutive bytes. When loading a BCD value onto the execution stack, the byte located at the lowest memory address contains the sign of the number along with the sign and magnitude of the exponent. Succeeding bytes represent two decimal digits and are ordered backwards, such that the byte closest to the exponent byte contains the last two decimal digits of the number, while the last byte contains the leftmost two decimal digits of the number. The format of a BCD number in memory is displayed in Fig. 24.

The exponent byte in a BCD number uses the high order bit to indicate the sign of the number: a one indicates positive, a zero negative. The remaining seven bits represent the exponent and its sign. The exponent is biased by 64 so that values greater than 64 (in seven bits) depict a positive exponent and values less than 64 depict a negative exponent; the exponent is the difference between 64 and the actual value. The bias allows exponent values.
REPRESENTATION OF $12.3456789$

$1.23456789 \times 10^{1}$

$1.23456789242$

<table>
<thead>
<tr>
<th>Memory Address</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1121E</td>
<td></td>
</tr>
<tr>
<td>1122E</td>
<td>0</td>
</tr>
<tr>
<td>1123E</td>
<td>42</td>
</tr>
<tr>
<td>1124E</td>
<td>2E</td>
</tr>
<tr>
<td>1125E</td>
<td>2F</td>
</tr>
<tr>
<td>1126E</td>
<td>6F</td>
</tr>
<tr>
<td>1127E</td>
<td>5E</td>
</tr>
<tr>
<td>1128E</td>
<td>3E</td>
</tr>
<tr>
<td>1129E</td>
<td>1E</td>
</tr>
<tr>
<td>1130E</td>
<td></td>
</tr>
</tbody>
</table>

BCD NUMBERS IN MEMORY

FIGURE 24.
ranging from -64 to +63. The DEC number always assures that the decimal point is normalized before the first digit.

d. String Data

Strings are stored sequentially. The first byte of the string stores the string length, thus limiting strings to a length of 255 bytes. Immediately following the length byte are the ASCII characters of the string.

2. Arithmetic Operations

a. Logical Operations

Logical, or boolean, operations act on byte values of zero and one only. A zero value indicates a false condition, while a non-zero value indicates true. Logical operations requiring comparison between two elements returns the value of the operation in the true or false form.

b. Integers

Arithmetic operations with integers are performed by taking the top two values from the execution stack, and placing them in the double byte registers in the 8080 microprocessor, and then carrying out the requested operation using the microprocessor's native functions. Integer arithmetic includes addition, subtraction, multiplication, division with truncation, modulo division, logical comparisons, and transformations to real (ECD) format. All computation results, except for real transformations, are returned to the execution stack in the two byte integer format. Relational operations or two
integer values are carried out in accordance with the rules for integer arithmetic.

b. Reals

Real arithmetic operations are more complex than those with integers due to the nature of the BCD format. The process is similar to that of integers in that pairs of real number bytes are moved to the 6060 registers. The required operation is performed, and the resulting real value is returned to the execution stack in the eight byte BCD format. Real values also follow the rules of integer arithmetic when involved in relational operations.

3. Set Operations

The set operations of set union, set difference, set intersection, set equality and inequality, set inclusion and set membership are not implemented in this version of NPS-PASCAL.

4. String Operations

The relational operators of equality and inequality have been implemented for strings. The remainder of the relational operators denote lexicographic ordering according to the character set ordering, and are not implemented in this version of NPS-PASCAL.

5. Procedures and functions

Procedures and functions, also called subroutines, give NPS-PASCAL the ability to display program segments as explicit subprograms. The only difference between a procedure and a function is that the function returns a
value to the top of the execution stack after it is invoked; a procedure does not. This means that a function call actually represents an arithmetic expression. Procedure calls, however, stand alone as program statements. An analysis of the following procedure and function implementation by Blanton and Moore [9] concluded that the current design is inadequate. Insufficient information is passed to allow parameter mapping from the execution stack to the PRT.

a. Invocation

Procedures and functions can be invoked with zero or more actual parameters. The list of actual parameters is substituted into the corresponding list of formal parameters declared in the procedure or function definition. If the formal parameter is a variable parameter, the actual parameter has to be a variable also. Should the formal parameter be a value parameter, then the actual parameter can be an expression, provided that the expression type matches the formal parameter type. For procedure and function formal parameters, the actual parameter must be a procedure or function identifier. Actual parameter types are checked against formal parameter types stored in the symbol table during program compilation. The method of passing actual parameters' values is via the execution stack. The procedure or function's memory location is generated in the form PRO <label>, where PRO is a mnemonic meaning "branch to"
subroutine", and \{label\} is the label value stored in the subroutine's symbol table entry.

b. Storage Allocation

All parameters and variables declared within a procedure or function are assigned a location in the PRT. These locations immediately follow the PRT location of the procedure or function identifier. Upon recognition of a complete subroutine, another PRT location is allocated. This location is called the Save Block Pointer (SBP) for the subroutine. The PRT locations extending from the subroutine's identifier location through the SBP make up a Procedure Control Block (PCB). The effect is that the PCB is a contiguous set of PRT cells, as seen in Fig. 25. The PCB construct is based on the one used in ALGOL-68 [12], and its usefulness is in recursive calls to a procedure or function.

c. Parameter Mapping

NPS-PASCAL uses a scheme similar to ALGOL-68 [12] in mapping the actual parameters of a procedure or function into its formal parameters. After recognition of a subroutine identifier, the actual parameters that are identifiers have their intermediate code generated in the form of a "PARX" or "PARMV" mnemonic followed by the PRT location of the actual parameter. These mnemonics load the execution stack with the values of the actual parameters. If the actual parameter is an expression, the expression result will be loaded automatically on top of the execution stack. Consequently, the compiler generates the mnemonic "PARX"
FUNCTION ZFPO (F, A: REAL): REAL;
VAR X, Z: REAL;
SY: STATE;

A Procedure Control Block

FIGURE 25.
after recognizing a complete expression that is acting as a value parameter. PARMX will not require any action by the code generator.

With the actual parameter in place, program control will branch to the procedure or function itself. The compiler generates code to place three items on top of the execution stack. The first item is the number of formal parameters \( f \) in the subroutine, the second is the PRT location of the subroutine's identifier (ILLOC), and the third is the SBP address in the PRT (SPPLLOC) of the subroutine. The compiler then generates the SAVP operator, followed by the total byte count of PRT storage \( t \) assigned for the subroutine's identifier and all formal parameters. This is followed by a listing of byte storage required by each formal parameter \( P_i \) in the PRT in descending order. The execution of the SAVP operator is expected to cause the following actions to be generated by the code generator.

1. The SBP location is examined
   (a) if \( \text{SBP} = 0 \) then \( \text{SBP} := 1 \), else
   (b) \( \text{SBP} > 0 \) and segment length \((\text{SPPLLOC} - \text{IDLOC} + 2)\) is obtained from the top of available memory, for example, at address \( x \). The PCB is then copied from the PRT to the memory segment at \( x \). The contents of the segment at \( x \) is then called the Save Block (SB). \( \text{SBP} := x \).
(2) The top two elements of the execution stack are deleted; the next element \( f \) is copied and deleted from the stack; \( P1 = p(1) \).

(3) If \( f = 0 \) then halt. All actual parameters have been copied into the formal parameter locations in the PCF.

(4) PFT location \( (IDLOC + t - p(1)) := \text{top of execution stack}; \) delete the top element of the execution stack; \( t := t - p(1); p(1) := p(1) - 1 \).

(5) \( f := f - 1; \) go to step (3).

This process ensures that recursively calling a subroutine will not destroy the local variables and parameters of any preceding calls.

1. Function Return Value

Coupled with the SAVP operator is the UNSP (unsave) operator that reverses the actions of SAVP. Two parameters are required at the top of the stack, the SFP locations in the PFT (SBPLOC), and the PFT location of the subroutine identifier (IDLOC). The actions, then, of UNSP are:

1. The value stored at IDLOC is copied to the top of the stack (this returns a value for the function calls; this value will be deleted for procedure calls).

2. If the value of SBPLOC is greater than 1 then the SB at location SBPLOC in the free memory area is copied back to the PCF and the memory is
VAR Y: INTEGER;

PROCEDURE LO (X: INTEGER; VAR Y: INTEGER);
    VAR TEMP: REAL;
    BEGIN
        TEMP := SORT(X);
        Y := TRUNC(TEMP);
    END;
    D := 6;
    ...
    LO(49, D);
    ...

STACK

<table>
<thead>
<tr>
<th>BEFORE SAVP</th>
<th>PRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>49</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AFTER SAVP, BEFORE UNSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs-</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AFTER UNSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs-</td>
</tr>
<tr>
<td>SPP</td>
</tr>
<tr>
<td>TEMP</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>LO</td>
</tr>
<tr>
<td>I</td>
</tr>
</tbody>
</table>

FIGURE 26.
freed. If SEP = 1 then SEP := 2. Consequently, the UNSP operator returns a value from function calls, and restores the PC3 in the event of recursive calls. Figure 26 shows the actions of the SAVP and UNSP operators on the PFT and the execution stack.

e. Forward Declared Procedures and Functions

To permit the invocation of a procedure or function prior to its definition, NPS-PASCAL utilizes a forward reference. The forward reference consists of the procedure (function) head, followed by the word `FORWARD'. When the procedure (function) is defined later in the program, the parameters are not repeated. `FORWARD' is not a reserved word in NPS-PASCAL. It is instead referred to as a directive. Directives are identifiers in NPS-PASCAL, that can only occur immediately after a procedure or function heading. Directives are contained in the `BUILTININST'.

f. Standard Procedures and Functions

The built-in procedures and functions that currently exist in NPS-PASCAL correspond to the standard procedures and functions specified in STANDARD PASCAL. Their operation, however, is considerably different from user defined procedures and functions. The compiler first generates code for any subroutine actual parameters. A mnemonic for the built-in procedure or function is then generated which tells the interpreter/translator that it must remove the parameters from the execution stack, perform
the requested operation, and return the result to the stack. The standard procedures for input and output (Read, Readln, Write, and WriteLn) will not require special action to be taken by the interpreter/translator. The remaining standard procedures dealing with files and pointer variables generate mnemonics that will require action by the interpreter/translator.

6. Input-Output

Input and output (I/O) can be handled in two ways: via console and via disk. Console I/O refers to the device the NPS-PASCAL user is utilizing to provide commands to the system — usually a CRT terminal or teletype. Disk I/O refers to utilizing auxiliary files on the disk for data manipulation.

Input from console I/O is achieved through READ or READLN statements. Console output is accomplished by the WRITE and WRITELN statements. Input to the console is accomplished by an operating system routine that reads one full console line into an input buffer. The code generator generates code to examine the buffer and convert ASCII characters contained within the buffer into appropriate NPS-PASCAL internal integer, real or string format. The input value is associated with the appropriate read statement variable parameter and then stored in the memory location allocated for that variable. A write statement takes the internal representations of integer, decimal, or byte values and converts them to their ASCII character
format. These values are then passed to an operating system print routine for console output. Constants and string variables are stored as ASCII strings in the intermediate code and the interpreter/translator will generate code to send them character by character to the system print routine.

Disk I/O is achieved through the same read and write statements utilized for console I/O. However, to read data from a disk file requires that the file identifier be specified as the first parameter in a read statement's list of actual parameters. The file identifier has to be specified in the same manner for disk write statements as well. The file identifiers used in read and write statements must be declared in a variable declaration part of a program block, or as a program parameter in the program declaration (called an external file). The file identifier has a specific PRT entry assigned by the compiler. At program execution, space will have to be allocated on the NPS-PASCAL stack for the File Control Block (FCB) information necessary to interface file operations with the operating system. Additionally, space should be provided for a 128 byte I/O buffer for every declared file.

7. NPS-PASCAL Pseudo Operators

A complete description of each of the NPS-PASCAL pseudo operators is presented in Ref. 2.
III. \textbf{PROBLEM IDENTIFICATION AND APPROACHES}

As noted in Ref. 2, the \textsc{Builtins} must be located at memory location 0177 in the executable module, since the collision field and previous entry addresses are calculated and entered by hand. Care must be taken during the \textsc{Link} and \textsc{Locate} programs to ensure that the \textsc{Builtins} is located properly. Since the \textsc{Link} program aids object modules together linearly, it is necessary to specify \textsc{TABLE.CES} as the first module in the command line to the \textsc{Link} program. While organizing the \textsc{Linked} together modules and adjusting the address into absolute code, the \textsc{Locate} program uses a default order of \textsc{Code}, \textsc{Stack}, \textsc{Data}, \textsc{Memory}. Constants in the \textsc{PLM-S9} source program (distinguished from variables by the \textsc{Data} directive), however, are allocated memory first, before any executable code. Forcing the memory address assignments to start at 0177 with the directive \texttt{CODE 0000} to the \textsc{Locate} program places \textsc{Builtins} at 022E, so a three byte dummy field was added right before the \textsc{Builtins} declaration. The first three bytes of the final CP/M executable file (16F5, 121E and 162F) are used to store a jump instruction which points to the compiler entry point.

The two previous theses used an \textsc{Eesec} simulator which ran on the IBM 360 and zeroed memory prior to loading the compiler. Many of the variables were not initialized, instead, relying on a zeroed memory location for their value. \textsc{PLM-S9} includes two directives, \textsc{Initial} and \textsc{Data},
which are used to set the initial value of variables and constants, respectively.

An additional difference between PL/I and PL/I-86 is that the latter allows an implicit dimension specifier. This allows the table declarations in TABLES.SRC and other long declarations to be made without knowing or counting the exact length of the data string. The implicit dimension specifier is invoked by entering an asterisk instead of a decimal constant, i.e. (*) instead of (43).

Due to a deficiency in the grammar and its associated tables, a record structure was not recognized until the END statement was parsed. It was then too late to initialize the variables used to analyze each record declaration. As an interim fix, the code to handle a record declaration had been written into the scanner portion of the compiler. Contrary to the structure of the compiler, when a record declaration was recognized by the token number, the record initializing code was executed. Correcting this problem was the subject of a project undertaken by Anderson and Myers [10] during a course in compiler theory at the Naval Postgraduate School. As a result of their work, this code was removed from the scanner, and placed in the production case statement where it belongs. The grammar was corrected, the parse tables regenerated, and changes to the existing tables were made by comparing the listings and typing changes by hand. In the SYNTU2.SRC module, production 55 was changed from
\(<\text{RECORD TYPE}\> ::= \text{RECORD} \ <\text{FIELD LIST}\> \ \text{END}\)

to

\(<\text{RFCCPT \ TYPE}\> ::= \ <\text{RECORD}\> \ <\text{FIELD LIST}\> \ \text{END}\)

and production 56 was added to read

\(<\text{RECORD}\> ::= \ \text{RECORD}.\)

A record is now recognized when the token RFCCPT is parsed, and the initialization of variables takes place correctly. All the remaining productions were renumbered to properly reflect the parse tables.

The user assistance program SYMBOTABLEF provided by the last thesis effort failed in attempting to print the symbol table for nearly every test program tried. Considerable effort was expended during the current effort to debug, modify and upgrade this program to a useful tool. Code was added to determine the actual location in memory of the symbol table during the compilation, and the symbol table is moved to that address for processing. The SYMBOTABLEF program was eventually abandoned for a number of reasons. First, it was attempting to read sequentially entries in the symbol table which were designed to be accessed via the hash table. All too often, the program crashed because it was not able to locate the beginning of the next entry. More frequently, though, the entry in the symbol table was incorrect, causing the SYMBOTABLEF program to use incorrect pointers, lengths, codes, etc. The SYMBOTABLEF program was replaced by a much simpler, but much more useful program, called SYMDUMP, which is described in the next paragraph.
The CP/VS utility DUMP was modified to print the contents of a file as a single column of hex character pairs, each representing a byte. Each pair is preceded by a four digit hex address, which corresponds to that byte's address in the symbol table, during compilation. The address of the beginning of the symbol table is a constant in the SYMDUMP program, and will have to be reset each time to reflect the new address of the symbol table whenever the compiler is changed. This necessitates reassembling SYMDUMP for each new version of the compiler, after determining the starting address of the symbol table from the previous SETEL entry address of the second entry. The output from the SYMDUMP program can be easily and efficiently scanned by hand to determine the contents of each entry. Collision address and previous entry address pairs, for instance, can usually be recognized on sight. Since the program is not data-dependent, it cannot crash due to improper symbol table entries. A description of the changes to the CP/VS utility DUMP.ASM is provided in Appendix C.

Examine the symbol tables from various test programs showed that the address of the parent type of simple variable declarations was not be entered properly. In production E6,

\[ \text{IDENT VAR STRING} ::= \text{IDENTIFIER} \]

code was added to save the parent type.

In the ENTR$SU$ENTRY procedure in SYMEL.SPC, the procedure SUBR$CASE was being called twice for the same
limit (upper) of the subrange. Code was added to modify the second call to examine the lower limit and thus correctly determine the number of entries in the subrange.

In most case statements throughout the compiler, there is no range checking done on the variable used to index into a case statement. In PL/M-30, if the index evaluates to a number greater than the number of case statements available, the result is undefined. In other cases, semicolons representing no-operation cases were omitted, causing the wrong code to be executed for a given case. Code was added to direct the index to the correct case.

In a few instances, PL/M-30 address variables (16-bit) were being passed to byte variables (8-bit), resulting in the eight high-order bits being truncated and lost. The offending variable declarations were corrected.

When the compiler was broken into modules, there were a substantial number of variables declared PUBLIC and EXTERNAL needlessly. When a variable was used only in the module in which it was declared, the PUBLIC declaration was deleted. A number of subroutines were declared PUBLIC in one module and not called, and declared external and called from only one other module. These subroutines were moved to the calling module and not declared PUBLIC or EXTERNAL.

The displacement vector associated with each array dimension was being calculated incorrectly and it was stored in the same symbol table entry as the subrange. The array offset (for non-zero-origin array dimensions) was being
calculated incorrectly. Code was added to temporarily stack the array declarations and subsequently enter them into the symbol table correctly. Code was also added to calculate the array offset and the displacement vector for each dimension.
II. REMAINING PROBLEMS

Signed identifier constant entries in the symbol table are identified as such by the FCST value 41, but the sign is not stored or applied to the value of the constant.

Arrays were only examined for correct identification and entry into the symbol table. Arrays on the right side of the assignment statement are not handled properly.

Since no interpreter has been written, there is still no way to validate the intermediate code produced. The compiler will compile some small test programs without crashing, but it frequently will crash or go into infinite loops.

The code in the modules SYMPOL.SRC, SYNTH1.SRC and SYNTESZ.SRC cannot be trusted to behave as described in the two previous thesis efforts. Each procedure needs to be examined on a line-by-line basis, with a possible eye toward rewriting substantial portions. In many, many cases, variables are ANDed or ORed with unexplained hex constants. The function of these constants should be determined and the hex constants should be named and documented. In many other instances, variables are shifted left or right and then again ANDed or ORed with hex constants. The shifting can be avoided by defining and documenting the appropriate masks. The global data base should be better organized, defined and documented. Variables enter procedures in unknown states, and are used or modified without range checking or any sort of validation. The ranges on case statement indices need to be checked before use, and each case should be a DO; END.
block, even if for a no-operation case, so that statements added will not introduce extra and erroneous cases.
II. CONCLUSIONS

NPS-PASCAL is still a long way from complete implementation. Major problems exist in the parse stack structure, in semantic action subroutines and in the symbol table construction and access. The groundwork for a viable PASCAL compiler has been started, but the compiler design needs a critical review and analysis.

The operation of this compiler is still dependent on the development of an SPS interpreter or translator to validate the pseudo operators generated. Completing the NPS-PASCAL compiler project will require a substantial investment of study and time.
APPENDIX A - Compiler Error Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td>Array dimension stack overflow; Simplify array declaration.</td>
</tr>
<tr>
<td>AM</td>
<td>Array nest overflow; Simplify declaration.</td>
</tr>
<tr>
<td>AT</td>
<td>Assignment type error: Type of expression not compatible with assignment variable type.</td>
</tr>
<tr>
<td>CF</td>
<td>Invalid expression: The variable types within the expression are not compatible.</td>
</tr>
<tr>
<td>CV</td>
<td>Incorrect control variable: The control variable has not been declared or is of type REAL.</td>
</tr>
<tr>
<td>LC</td>
<td>Duplicate constant name: Constant identifiers must be unique.</td>
</tr>
<tr>
<td>DF</td>
<td>Disk error: Recompile.</td>
</tr>
<tr>
<td>LT</td>
<td>Duplicate type name: Type identifiers must be unique.</td>
</tr>
<tr>
<td>EE</td>
<td>Exponent size error.</td>
</tr>
<tr>
<td>ET</td>
<td>Invalid expression type: The types of the variables used in an expression are incompatible.</td>
</tr>
<tr>
<td>IA</td>
<td>Invalid array index: Array index types must be scaler - INTEGER or REAL types are invalid.</td>
</tr>
<tr>
<td>IC</td>
<td>Invalid constant variable: Constant entry in symbol table is invalid.</td>
</tr>
<tr>
<td>IX</td>
<td>Integer size error.</td>
</tr>
<tr>
<td>IP</td>
<td>Improper parameter: The actual parameter type does not match the formal parameter type.</td>
</tr>
<tr>
<td>IR</td>
<td>Invalid read variable: Only INTEGER, REAL or STRING values can be read.</td>
</tr>
<tr>
<td>IS</td>
<td>Invalid subrange error: Check type and limits of declared subrange.</td>
</tr>
<tr>
<td>IT</td>
<td>Invalid type error: Array component type specification invalid.</td>
</tr>
<tr>
<td>IV</td>
<td>Variant stack overflow: Reduce the number of variant cases.</td>
</tr>
<tr>
<td>LS</td>
<td>Label syntax error: All labels must be integers.</td>
</tr>
</tbody>
</table>
NC  Incorrect character.
NE  Incorrect actual parameter: The actual parameter must be a variable and not an expression.
NP  No production: Syntax error in source line.
NS  Invalid set element: Set elements must be scalar.
PE  Parameter error: This parameter format can only be used in a write statement.
PK  Incorrect number of parameters: The total number of actual parameters fails to equal the total number of formal parameters.
RN  Record field stack overflow: Reduce the number of fields specified.
RT  WRITE$STM parameter error: The parameter has to be of type REAL.
SC  State stack overflow: simplify program.
TI  Invalid type identifier: Type identifier not previously declared.
TC  Symbol table overflow: Reduce number of declarations.
UL  Undefined label error: Label not declared in label statement.
UC  Invalid unary operator: Variable type must be INTEGER, REAL or subrange of INTEGER.
UP  Undeclared procedure: Procedure identifier not previously declared.
VN  Variable declaration stack overflow: Reduce the number of variables declared per line.
VO  Variable stack overflow: Reduce the length of variable printnames.
WP  WRITE$STM parameter error: The length parameter has to be of type integer.
APPENDIX E - Intermediated Code DECODE Program

The last thesis effort included a program called DECODE which will read the intermediate code file and convert the hex pseudo codes into the corresponding parameters. The parameters associated with certain operators, such as labels, branches and load immediate values are printed also. Integer and real numbers are converted to decimal format. Strings are displayed as ASCII characters.

To use the DECODE program, compile a PASCAL program omitting the $C$ compiler toggle:

```
A>PASCAL TEST.PAS
```

When a successful compilation is complete, run the DECODE program on the intermediate file:

```
A>DECODE TEST.PIN
```

The contents of the intermediate file will be printed on the console.
APPENDIX C - SYMDUMP Symbol Table Display Program

A symbol table displaying program was developed to aid in examining the symbol table and debugging the compiler. It is based on the CP/M DUMP utility, and uses the starting address of the symbol table in memory.

To prepare the SYMDUMP program, the user must first use the standard CP/M utility DUMP to list the symbol table file. In this dump, the user determines the starting address of the symbol table by examining the previous entry address of the second entry. This address will change whenever the compiler is altered, since the symbol table is assigned to the first available memory address after the compiler.

Modify the CP/M utility DUMP as follows: after the label OPEN CY, change the argument of the LXI a from 0 to the starting address of the symbol table; after the label DUMP, delete the JNZ NODUMP instruction. Repare, reassemble and reload the program. The SYMDUMP program is now ready to be used on the .SYM file produced by the compiler:

A>SYMDUMP PROGRAM.SYM

SYMDUMP produces a vertical listing of the symbol table, one byte per line; each byte is preceded by its address in the symbol table.
APPENDIX 1 - Compiler Source Code Structure

A. MODULARIZATION

The PL/M version of the NPS-PASCAL compiler contained over 4700 lines of source code. When the compiler was transferred to the Intel Microprocessor Development System (MTS) and the ISIS-II operating system, it was broken up into manageable modules according to function. Each module now has fewer than 1000 lines of code, so editing is facilitated, and corrections to the compiler can be implemented much more rapidly. The two largest modules take less than 15 minutes each to recompile. A recompiled module can then be linked with the remaining modules. Maintaining the compiler as a single, large file would have caused excessively long edit sessions, and a recompile time of over an hour.

There are seven modules, each in a separate ISIS-II format file. SYSETS.SRC contains the interface to the PL/M operating system, including the disk and console input-output procedures, and the GETCHAR procedure. SCAV.SRC contains the input scanner. PARSER.SRC contains the parser and its supporting procedures, and most of the global variables. TABLES.SRC contains the built-in symbol table and the parse tables. SYMBOL.SRC consists of procedures which manipulate the symbol table, either writing into or reading from individual entries. SYNTFL.SRC contains the code synthesizer, procedures which use the parse stacks and which generate the intermediate code. SYNTFE.SRC consists solely
of the production case statement. Source listings of the
modules are provided following the appendices.

Modularizing the compiler introduces the PL/00
compiler directives PUBLIC and EXTERNAL. Any variable,
function or procedure which is declared in one module, and
referenced in another, must be declared PUBLIC in the first,
and EXTERNAL in the second. Functions and procedures which
have arguments must have those arguments in both
declarations, also.

The IXREF switch of the PL/00 compiler causes a
cross-reference to be appended to the source listing. The
cross-reference contains each source program identifier
(literal, constant, variable, function or procedure) which
occurs in the program, along with the line number of its
defining occurrence, the line numbers of any references to
it, and whether it is declared PUBLIC or EXTERNAL. This
cross reference is a very useful tool for locating
identifiers.

The IXREF switch of the PL/00 compiler causes a
temporary file with an .IXI extension to be created, which
contains information about each PUBLIC and EXTERNAL
declaration in the source program. These .IXI files, one for
each source module, are later collected and consolidated by
the IXREF program, which produces an inter-module cross
reference listing. This listing contains all PUBLIC and
EXTERNAL identifiers, and names the module in which the
identifier was declared PUBLIC, and lists all modules which
make an EXTERNAL reference to it. This list is also very useful during debugging.

3. LINKING AND LOCATING

The compiler, now separated into modules, must be recombined to form a body of executable code. This is accomplished by the LINK and LOCATE programs. The LINK program adds code from each of the modules and libraries referenced linearly, to form a single file. The LOCATE program locates the code at a particular address in memory and adjusts all of the relocatable addresses into absolute addresses.

C. TRANSFER FROM ISIS-II TO CP/M

Once the complete compiler has been located and adjusted, it needs to be transferred from the ISIS-II based system where the PL/M-80 compiler resides to a CP/M based system for execution. This is done with FROMISIS.COM, an undocumented program which runs under CP/M and reads a file from an ISIS-II format disk onto a CP/M format disk. The compiler is then processed by the undocumented program O2CPM.COM, which strips off any symbol table information, adds a JMP instruction to the entry point to the beginning of the compiler, and creates the executable form of the compiler. The symbol table information is placed in separate files with .SYM and .LIN extensions. These files can be deleted if empty or not used, or they can be saved for use with the debugging tool SID.
I. EXECUTION AND DEBUGGING

When invoking NPS-PASCAL on a PASCAL program, the 
compiler is treated as any other program under CP/M. Along 
with the file name of the PASCAL program to be compiled, 
NPS-PASCAL accepts up to four switches which cause it to 
print to the console the PASCAL source code, the production 
numbers, the token numbers, and cause it to suppress 
creation of the intermediate file.

The facilities of SID, the CP/M Symbolic Instruction 
Debugger, permit run-time debugging and execution tracing of 
the compiler. To use SID, it is necessary to include the 
FL/WS-68 compiler DEBUG switch when compiling the module of 
interest. The DEBUG switch causes the FL-68 compiler to 
include identifier and line-number locations with the file. 
This information is later stripped out by the ORGANIZATION program into the PASCAL.SYM and PASCAL.LIN files. These 
files are loaded by SID and used to reference and identify 
absolute machine addresses by symbolic expressions. 
Effective debugging of the compiler requires a detailed 
knowledge of the operation of SID as documented in the SID 
Users Manual. In transferring the compiler from FL/WS to 
FL/WS-68, it was necessary to shorten some of the identifier 
names to less than 16 characters to meet the requirements of 
SID.

In order to ascertain the proper operation of the 
compiler, it is also necessary to have accurate knowledge of 
the PASCAL language. To ensure testing the compiler with
programs of proper PASCAL syntax, that test programs were
taken either from the Pascal Manual and Report or the Pascal
Validation Suite.

The entire compilation, linking and loading, transfer to
CP/M, execution and debugging process is documented by
example in Appendix E.
APPENDIX E - COMPIL, LINK and LOAD Instructions

This appendix provides step-by-step directions for compiling the NFS-PASCAL compiler, linking, and locating the object modules, generating cross-reference listings, transferring the compiler to a CP/M based system, and executing and debugging the compiler. For additional information about the ISIS-II system, see Refs. 11 - 17. For additional information about operation under the CP/M system, see Ref. 14.

The NFS-PASCAL source files are compiled, linked and located under the ISIS-II operating system. First, compile each module with the appropriate switches to the PL/I-62 compiler:

-PLM62 :F1:SYSTRS.SRC XREF LREF DATT(24 MAR 83) DEBUG
-PLM62 :F1:TABLES.SRC XREF LREF DATT(29 MAR 83) DEBUG
-PLM62 :F1:PAPSTH.SRC XREF LREF DATT(29 MAR 83) DEBUG
-PLM62 :F1:SCAN.SRC XREF LREF DATT(29 MAR 83) DEBUG
-PLM62 :F1:SYMECL.SRC XREF LREF DATT(29 MAR 83) DEBUG
-PLM62 :F1:SYNE1.SRC XREF LREF DATT(29 MAR 83) DEBUG
-PLM63 :F1:SYNTH2.SRC XREF LREF DATT(29 MAR 83) DEBUG

Due to space limitations on a single disk, it may be necessary to copy the .LST files to another disk as they are generated, or to redirect the .LST file to the .:F1: disk with the PRINT switch:

-PLM63 :F1:SYNTH1.SRC XREF LREF DEBUG PRINT(:F1:SYNTH1.LST)
text, generate the inter-module cross-reference:

-IMREF :F1:PASCAL ("NPS-PASCAL Ver 1.7")

A printed copy of the inter-module cross-reference is very useful during debugging.

A "SUMMIT" file has been created to facilitate the LINKing and LOCATING process. If a different LINKing or LOCATING command string is desired, it can, of course, be entered by hand. To invoke the prepared file:

-SUMMIT :F1:PASCAL

The file :F1:PASCAL.CSI used by the SUMMIT command contains the following command lines:

-IFGETS :F1:PASCAL.LNK,:F1:PASCAL
-LOCATE :F1:PASCAL.LNK CODE(1234) MAP

Execution of these lines will create the files
:F1:PASCAL.LNK and :F1:PASCAL.

Leaving the ISIS-II disk containing the NPS-PASCAL compiler in drive 1, insert and boot a CP/M disk in drive J. The CP/M disk must contain, among other programs, the programs PROMISIS.COM, OBJCPM.COM and SID.COM. Transfer the NPS-PASCAL compiler from the ISIS-II disk to the CP/M disk:
A>PROVISYS PASCAL

Break out the .SYM and .LIN files and add the CMP instruction to locations 142H, 121H, and 1427H:

A>CECOPM PASCAL

This command will create three files from the PASCAL file: PASCAL.COM, the executable compiler, PASCAL.SYM and PASCAL.LIN, the files containing symbol table information for the run-time debugger SII.COM. When debugging with SII.COM, it is useful to have printed copies of the .SYM and .COM files. The file PASCAL (with no extension) can be deleted.

Create a PASCAL source program, for example TEST.PAS, with an available text editor. Invoke the NFS-PASCAL compiler:

A>PASCAL TEST.PAS

Up to four switches may be provided to the NFS-PASCAL compiler through the CP/M parms field immediately following the file specification:

A>PASCAL TEST.PAS SAPCL
The switches may appear in any order and have the following meanings:

A - List the source program lines.
G - List the production numbers.
C - Suppress creation of the intermediate file.
I - List the token numbers.

To invoke the run-time debugger SID.COM:

AXSID PASCAL.COM PASCAL.SYN
SID UPS 1.4
SYMBO}\$ D S E X T PC P X
$020 0120 0070
*1* PASCAL.IN
#
*7 TEST,PAS $ABCD

Then set up pass points, etc. and debug as necessary. For detailed instructions in the use of SID.COM, the run-time debugger, see Ref. 15.
APPENDIX F - Disk Directories

The XPS-PASCAL compiler is stored on two 1313-II format disks with directories as follows.

The source files, the compiled object files, and the

*.IKI files are on the first disk:

DIRECTORY OF :F1:DISK

NAME	EXT	PLKS	LEN:	ATTR
COPY	65	5047
'TIP	46	5733
PRINT	OBJ	2	70
.SCAN	.SRC	53	16347
SYSTS.IKI	6	549
SYSTS.OBJ	46	5434
SYNCL.SRC	241	72277
.SCAN	.OBJ	31	3633
SYNTE2.SRC	406	56247
PARSER.OBJ	56	6969
CONVP.SRC	37	4577
SYNTE1.SRC	443	53328
SYNTF1.CRF	104	22577
SYNTE2.OPJ	130	16128
TABLES.CRF	35	4547
PASCAL.CSD	7	227
PARSEP.IKI	15	1742
SYSTS.SRC	69	11128
SYNT1.IKI	17	1987
PASCAL.LNK	482	62306
PASCAL	384	47692
SYNTE2.IKI	16	1807
SYMPOL.OPJ	67	12844
PARSEP.SRC	112	13966
TABLES.SRT	73	3604
.SCAN	.IKI	4	251
TABLES.IKI	3	156
SYNCL.IKI	11	1264
The second disk consists solely of listing files due to their large size:

```plaintext
DIRECTORY OF :F1:LIST1
NAME       TXT  BLKS  LENGTH  ATTR
TED         180   16951 W
TRANZ.03J   2     70 W
COPY        62    6642 W
ATTRIB      35    422 W
DINCEI      28    2399 W
DINCAT      37    4025 W
IF          46    5733 W
FLIT        56    6569 W
FORMT       49    6060 W
RECONV       35    4261 W
ITSK        50    6261 W
LIB         92    17227 W
LINK        114   14295 W
LOCATE      168   13565 W
OBJEX       27    7234 W
PENAM       21    2457 W
SUBMIT      38    4628 W
PLMB0       172   21625 W
TYPE        5     492 W
UPRF        82    10216 W
PLMB0 .LIB  45    5615 W
PLME0 .OVL  152   16731 W
PLMAK .CV1  232   29122 W
PLMAK .OV2  62    8150 W
PLMAK .OVT  169   23725 W
PLME0 .CV4  72    6522 W
SYSTEM .LIB 24    2846 W
LINK .OVL   27    3981 W
```

The ISIS-II system disk used during the development of the VPS-PASCAL compiler contains the following:

```plaintext
DIRECTORY OF :F2:ISIS
NAME       TXT  BLKS  LENGTH  ATTR
TED         136   16951 W
TRANZ.03J   2     70 W
COPY        62    6642 W
ATTRIB      35    422 W
DINCEI      28    2399 W
DINCAT      37    4025 W
IF          46    5733 W
FLIT        56    6569 W
FORMT       49    6060 W
RECONV       35    4261 W
ITSK        50    6261 W
LIB         92    17227 W
LINK        114   14295 W
LOCATE      168   13565 W
OBJEX       27    7234 W
PENAM       21    2457 W
SUBMIT      38    4628 W
PLMB0       172   21625 W
TYPE        5     492 W
UPRF        82    10216 W
PLMB0 .LIB  45    5615 W
PLME0 .OVL  152   16731 W
PLMAK .CV1  232   29122 W
PLMAK .OV2  62    8150 W
PLMAK .OVT  169   23725 W
PLME0 .CV4  72    6522 W
SYSTEM .LIB 24    2846 W
LINK .OVL   27    3981 W
```
SYSRTS.SPC

$PATHWIDTH(9?) TITLE("SYSRTS - SYSTEM SUPPORT ROUTINES")

SYSSYSES:IFC

;/* CPM INTERFACE ROUTINES */

DECLARE LIT LITERALLY "LITERALLY",
    EXT LIT 'EXTERNAL',
    CR LIT '13',
    LF LIT '0AH',
    LCL LIT 'DECLARE',
    PROC LIT 'PROCEDURE',
    TRUE LIT '1',
    ADR LIT 'ADDRESS',
    FALSE LIT '0',
    FILEEOF LIT '1',
    FOREVER LIT 'WHILE TRUE';

DCL
    EOLCHAR LIT '0D', /* END OF SOURCE LINE CHARACTERS */
    TAB LIT '09',
    SOURCEFSIZE LIT '123', /* SIZE OF SOURCE FILE */
    REPORT */
    INTFSIZE LIT '128', /* INTERMEDIATE FILE RECORD SIZE */
    CONCUFFSIZE LIT '62', /* SIZE OF CONSOLE BUFFER */
    CONCPFILLER LIT '1AH'; /* CP for last record on file */

/*
   **********************************************
   /**** GLOBAL VARIABLES ****/
   **********************************************
   LCL

    /* COMPILER TOGGLES */

    LISTSOURCE BYTE EXT,
    NOINTFILE BYTE EXT,

    /* FXT VARIABLES */

    PRODUCTION BYTE EXT,
    TOKEN BYTE EXT,
    ACCUM(32) BYTE EXT,
    NEXTCHAR BYTE EXT,
    LAST$SET$SEL ADDR EXT,
    /* COUNTERS */
    EOFC LITERALLY '25', /* EOF */
    PARM'S ADR PUBLIC INITIAL(0),
    ERRCOUNT ADR PUBLIC INITIAL(0),
    CODESIZE ADR PUBLIC INITIAL(0),
    DECII(4) ADR INITIAL(1000,100,12,1);
**SYSTEM DEPENDENT ROUTINES AND VARIABLES**

**THE FOLLOWING ROUTINES ARE USED BY THE COMPILER**

**TO ACCESS DISK FILES AND THE CONSOL.**

**ROUTINES ASSUMPT THE USE OF THE CP/M OPERATING**

**SYSTEM.**

**THE SCE'S ARE USED BY THE SYSTEM TO MAINTAIN**

**INFORMATION ON OPEN FILES. THEY ARE ONLY USE BY**

**PROCEDURES IN THIS SECTION. THE BUFFERS AND POINTERS**

**TO THE BUFFERS ARE USED BY THE REMAINDER OF THE**

**PROGRAM, BUT THEIR SIZE MAY BE VARIED TO SUIT THE DISK**

**OPERATING SYSTEM BEING USED.**

*/ DCL
/* NOTE: CP/M PROVIDES SCE AS SCE AREA AND SCE AS A
BUFFER FOR* /

PROGRAM USE */
RFCBADDR ADDR INITIAL(SCE),
PCBR BASED RFCBADDR(33) BYTE, /* SOURCE FOR */
WFCE(33) BYTE /* INTERMEDIATE FILE SCE */
INITIAL (0, "PIN", 0, 0, 0).
SYCB(33) BYTE /* SYMBOL TABLE FILE SCE */
INITIAL (0, "SY", 0, 0, 0).
SRLOC ADDR INITIAL(S0H),
SOURCEBUFF BASED SBLOC(SOURCERESIZE) BYTE, /*
SOURCE BUFFER */
SOURCEPTR BYTE INITIAL (SOURCERESIZE), /* BUFFER INDEX */
DISKOUTBUFF(INCRESIZE) BYTE,
SYMCUTBUFF(INCRESIZE) BYTE,
BUFFPTR BYTE INITIAL(255), /* BUFFER INDEX */
SYMUFFPTR BYTE INITIAL(255), /* SET EUFF INDEX */
/*
LINEBUFF(CONBUFFESIZE) BYTE, /* CONSOLE OUT BUFFER */
LINEPTR BYTE INITIAL(0), /* BUFFER INDEX */
BDOS ADDR PUBLIC INITIAL(5H), /* JMP TO C/S ENTRY */
BOOT ADDR INITIAL(0H), /* REPO ENTRY */
LINENO ADDR, /* CURRENT LINE NUMBER */
STARTDOS ADDR PUBLIC INITIAL(CH); /* PTR TO START OF

END* /

*******************************************************************************/
/** G L O B A L P R O C E D U R E S **/
*******************************************************************************/

MOVE: PROC (SOURCE, DESTIN,L) PUBLIC;
/*MOVES FROM SOURCE TO DESTIN FOR L BYTES */
DCL (SOURCE, DESTIN) ADDR, /* L < 255 BYTES */
(SCHAR BASED SOURCE, DCHAR BASED DESTIN,L) BYTE;
TO WRITE (\$ = \$ + 1') <> 255:
    SOURCE = SOURCE + 1;
END MOVE;

FILL:  PROC (A,CHAR,N) PUBLIC; /* MOVE CHAR TO A N TIMES */
    DCL A ADDR,(CHAR,N,DIST BASED A) BYTE;
    DO WHILE (N := N-1) <> 255;
        DEST = CHAR;
        A = A + 1;
    END;
END FILL;

/* DEVICE ROUTINES */
MON1:PROC(FUUC,INFO) EXT;
    DCL FUNC BYTE,
        INFO ADDR;
END MON1;

MON2:PROC(FUUC,INFO) BYTE EXT;
    DCL FUNC BYTE,
        INFO ADDR;
END MON2;

MON3:PROC PUBLIC;
    CALL FOOT;
END MON3;

/* I/O ROUTINES */
PRINTCHAR:PROC(P) PUBLIC;
    /* END THE ASCII CHARACTER P TO THE CONSOLE */
    DCL B BYTE;
    CALL MON1(2,B);
END PRINTCHAR;

PRINT:PROC(A) PUBLIC;
    /* PRINT THE BUFFER STARTING AT ADDRESS A UNTIL \$ */
    DCL A ADDR;
    CALL MON1(9,A);
END PRINT;

READ:PROC(A) PUBLIC;
    /* READ CONSOLE CHAR'S INTO BUFFER A */
    DCL A ADDR;
    CALL MON1(10,A);
END READ;

CPLF:PROC PUBLIC;
    /* END CARIAGT-RETURN-LINE-FEET TO THE CONSOLE */
    CALL PRINTCFAR(CE);
CALL PRINTCHAR(LF);
END CPLF;

PRINTDEC: PROC(VALUE) PUBLIC;
CALL VALUE ADDR, 1 BYTE, COUNT BYTE;
CALL FLAG BYTE;
FLAG = FALSE;
DO I = 3 TO 3;
COUNT = 30;
DO WHILE VALUE >= DEC(I);
VALUE = VALUE - DEC(I);
FLAG = TRUE;
COUNT = COUNT + 1;
END;
IF FLAG OR (I>= 3) THEN
CALL PRINTDEC(COUNT);
ELSE
CALL PRINTCHAR( ));
END;
END PRINTDEC;

PRINTSTOKEN: PROC PUBLIC;
CALL PRINT(, ("TOKEN = ");
CALL PRINTDEC(TOKEN);
CALL PRINT(, (" ");
END PRINTSTOKEN;

PRINTPROD: PROC PUBLIC;
CALL PRINT(, ("PROD = ");
CALL PRINTDEC(PRODUCTION);
CALL PRINT(, (" ");
END PRINTPROD;

PRINTERROR: PROC PUBLIC;
CALL CPLF;
CALL PRINTDEC(ERRORCOUNT);
CALL PRINT(, ("ERROR(S) DETECTED", CR, LF, " ");
END PRINTERROR;

ERROR: PROC (ERRORCOLE) PUBLIC;
DCL ERRORCODE 'ADD',
  I BYTE;
ERRORCOUNT=ERRORCOUNT+1;
CALL CPLF;
CALL PRINT(, ("ERROR ");
CALL PRINTDEC(LINENO);
CALL PRINT(, (" ");
CALL PRINTCHAR(HIGH(ERRORCODE));
CALL PRINTCHAR(LOW(ERRORCODE));
CALL PRINT(, (" ");
DO I=1 TO ERRORCOUNT;
  CALL PRINTCHAR( ");
END:

NPS-PASCAL: A MICROCOMPUTER-BASED IMPLEMENTATION OF THE PASCAL ETC (U)
MAR 80 K. S. TINNIS
UNCLASSIFIED
CALL PRINT1("CR,LF,'AT FRCR S'");
CALL PRINTSSTORE;
CALL PRINTSPROC;
IF TOKEN=FOFC THEN DO;
   CALL PRINTSERROR;
   CALL MON3;
ENDD;
END ERROR;

DISKERR:PROC;
   CALL ERROR('DE');
   CALL MON3;
END DISKERR;

OPEN$SRC$FILE: PROC PUBLIC;
   CALL MOVE1('PAS',RFCEADDR+9,3);
   RFCE(22),RFCE(12) = 0;
   IF MON2(15,RFCEADDR) = 255 THEN DO:
      CALL FRPCP('WS');
      CALL MON3;
   ENDD;
END OPEN$SRC$FILE;

READ$SRC$FILE:PROC BYTE;
   DCL DCNT BYTE;
   IF (DCNT=MON2(20,RFCEADDR)) > FILEFCF THEN CALL DISKERR;
   RETURN DCNT;
END READ$SRC$FILE;

SETUP$INTS$FILE:PROC PUBLIC;
   IF NCINTFILE THEN /*ONLY MAKE FILE IF TOGGLE OFF*/
      RETURN;
   CALL MOVE('RFCE,WFCE,9);
   CALL MON1(19,WFCE);
   IF MON2(22,WFCEADDR)=255 THEN CALL DISKERR;
   /*SET UP SYMEOL TABLE FILE */
   CALL MOVE('RFCE,RFCE,9);
   RFCE(32)=0;
   WFCE(22) = 0;
   CALL MON1(19,RFCE);
   IF MON2(22,RFCEADDR)=255 THEN CALL DISKERR;
END SETUP$INTS$FILE;

WRIT$INTS$FILE:PROC PUBLIC;
   IF NCINTFILE THEN RETURN;
   CALL MON1(26,DISFOUTBUFF);
   IF MON2(21,WFCE)<<0 THEN CALL DISKERR;
CALL MCN1(26,82H); /* PE377 DMA ADDR */
END WRIT$INT$FILE;

EMIT: PROC(OBJCODE) PUBLIC;
DCL OBJCODE BYTE;
IF (?BUFFPTR := BUFFPTR+1) >= INRECSIZE THEN
/*WRITE TO DISK*/
DO:
   CALL WRIT$SYM$FILE;
   BUFFPTR=0;
END:
DOSKOUTBUFF(BUFFPTR)=OBJCODE;
END EMIT;

GEN$FLATT:PROC(OBJCODE) PUBLIC;
DCL OBJCODE BYTE;
GCODESIZE=CODESIZE+1;
CALL EMIT(OBJCODE);
END GENERATE;

GEN$ADDR:PROC(A,E) PUBLIC;
DCL A BYTE, E ADDR;
CALL GENERATE(A);
CALL GENERATE(LCN(E));
CALL GENERATE(HIGH(E));
END GENERATE;

WRIT$SYM$FILE: PROC PUBLIC;
IF ($INT$FILE THEN
   RETURN;
   CALL MCN1(26,SYMOUTBUFF);
   IF MCN2(21,SFC3)<<0 THEN
      CALL DISKERR;
      CALL MCN1(26,80E); /*RESP DMA ADDR*/
   END WRIT$SYM$FILE;

GEN$SYM$TEL:PROC(OBJCODE) PUBLIC;
DCL OBJCODE BYTE;
IF (SYM$BUFFPTR:=SYM$BUFFPTR+1) >= INRECSIZE THEN
/*WRITE TO DISK*/
DO:
   CALL WRIT$SYM$FILE;
   SYM$BUFFPTR=0;
END:
SYM$OUTBUFF(SYM$BUFFPTR)=OBJCODE;
END GEN$SYM$TEL;

MOV$SET$TEL:PROC PUBLIC;
DCL SYMPTR ADDR;
DCL VALUE BASED SYMPTR BYTE;
TO SYMPTR=.M$MEMORY TO (LAST$SET$TEL$1-1);
   CALL GEN$SYM$TEL(VALUE);
END;
CALL GEN$SYM$TEL(6);

96
CALL GENSYMTE(0);
CALL GENSYMTE(1);
CALL GENSYMTE(2);
CALL GENSYMTE(ZCFILLER);
CALL GENSYMTE(ZCFILLER);
CALL WRITESYM$FILE;
END "OVE$SETEL"

CLOSESINT$FIL: PROC PUBLIC;
/*CLOSE INT CODE FILE AND SYM TABLE FILE*/
IF NOINTFILE THEN
   RETURN;
IF MONZ(16, .WCF)=255 THEN
   CALL I$KERR:
IF MONZ(16, SFCB)=255 THEN
   CALL DISFERR;
END CLOSESINT$FIL:

CLEAR$LN$BUFF: PROC PUBLIC;
   CALL FILL(.LINEBUFF, "", XON$BUFFSIZE);
END CLEAR$LN$BUFF:

LISTLINE: PROC LENGTH);
   DCL (LENGTH, I) BYTE;
   CALL CRLF;
   CALL PRINT$DEC(LINENC);
   CALL PRINT$CHAR(" ");
   TO I = 0 TO LENGTH;
      CALL PRINT$CHAR(LINEBUFF(I));
   END;
   CALL CRLF;
END LISTLINE:
/* SCANNER INTERFACE */

GETCHAR: PROC BYTE PUBLIC;
   NXT$SRC$CHAR: PROC BYTE;
   RETURN SOURCEBUFF(SOURCEPTR);
END NXT$SRC$CHAR;

CHECKFILE: PROC BYTE;
   TO FOREVER;
   IF (SOURCEPTR=SOURCEPTR+1)>SOURCEFILE THEN 
      DO;
      SOURCEPTR=0;
      IF READ$SRC$FILE=FILEEOF THEN
         RETURN TRUE;
      ENI;
      IF (NEXTCHAR=NXT$SRC$CHAR)<LF THEN
         RETURN FALSE;
      ENI;
   END CHECKFILE;

97
IF CHECKFILE OF (NEXTCHAR = EOF) THEN
  IF FOR REACHED */
     CALL MOVE(('EOH', EOFCHAR, LF), SRCLOC, E);
     SOURCEPTR = 0;
     NEXTCHAR = XT$SRC$CHAR;
   END;
LINEBUFF(LINEPTR := LINEPTR + 1) := NEXTCHAR; /* OUTPUT LINE */
IF NEXTCHAR = HOFCHAR THEN
  DC:
  LINENO = LINENO + 1;
  IF LISTSOURCE THEN
     CALL LISTLINE(LINEPTR-1);
  END;
  CALL CLEANBUFF;
  END; IF NEXTCHAR = TAB THEN
  CALL GETCHAR;
END GETCHAR;
END SYS$Routines;
DECLARE LIT LITERALLY "LITERALLY",
SCANN: LG:
  ECL LIT "DECLARE",
  PROC LIT "PROCEDURE",
  EXT LIT "EXTERNAL",
  TRUE LIT '1',
  ADDR LIT "ADDRSS",
  FALSE LIT '0',
  COMMENT LIT "REM",
  UNCOMMENT LIT "FHC",
  FOPEN LIT "FILE TRUE";

ECL IDENTSIZE LIT '32', /* MAX IDENTIFIER SIZE + 1 */
  FOLCHAP LIT '0EH', /* END OF SOURCE LINE CHARACTER */
  HASHMASK LIT '127', /* HASH TABLE SIZE +1 */
  STRINGDELIM LIT '27E', /* CHAR USED TO DELIMIT STRINGS */

/* NUMBER TYPES */
  INTEGER TYPE LIT '1',
  UINTEGER EXPON LIT '3',
  REAL TYPE LIT '2',
  SIGNEDEXPON LIT '4';
/* GLOBAL VARIABLES */

DCL LIST$TOKEN BYTE PUBLIC INITIAL(FALSE),
  LIST$PROE BYTE PUBLIC INITIAL(FALSE),
  LIST$SOURCE BYTE PUBLIC INITIAL(FALSE),
  ERROR$LIN BYTE PUBLIC INITIAL(FALSE),
  NOTINT$FILE BYTE PUBLIC INITIAL(FALSE),
/* GLOBAL VARIABLES USED BY THE SCANNER */
  TOKEN BYTE EXT, /* TYPE OF TOKEN JUST SCANNED */
  FASCODE BYTE EXT, /* FAS VALUE OF CURRENT TOKEN */
  NEXTCHAR BYTE PUBLIC, /* CURRENT CHARACTER FROM 
GETCHAR */
  CNUM BYTE EXT, /* INDICATES FULL ACCUM--STILL MORE */
  ACCUM(IDENTSZ) BYTE EXT, /* HOLDS CURRENT TOKEN */

  NUMBERTC LIT '54', /* NUMBER */
  STRINGC LIT '55', /* STRING */
  IDENTC LIT '59'; /* IDENTIFIER */
/* LOCAL VARIABLES */

DCL LOCK$ BYTE, /* TRUE WHEN GETCHAR HAS ALREADY RETURNED A
chap*/
  tempchar1 byte, /* holds previously scanned chap */
  tempchar2 byte; tcl parm list(?) byte initial('');
  declare voca(170) byte initial
    (0, 1, 2, <, ' ', ' ( ', ' ) ', ' - ', ' + ', ' = ', ' ? ', ' ! ', ' , ', ' ! ', ' = ', ' ; ',
    ' cp ', ' to ', ' top ', ' div ', ' pad ', ' for ', ' mod ', ' nil ', ' not ', ' set ',
    ' var ', ' case ', ' else ', ' file ', ' goto ', ' then ', ' type ', ' with ', ' array ',
    ' begin ', ' const ', ' label ', ' until ', ' while ', ' count ', ' packed ', ' recode ',
    ' repeat ', ' program ', ' function ', ' procedure ');
  dcl vloc(10) byte
  initial(3,1,17,33,63,91,121,145,157,162);
  dcl vnum(10) byte initial(6,1,17,25,35,42,46,53,56,59);
  dcl count(10) byte initial(0,1,7,9,6,5,7,0,0,0);
  /* global procedures */
  declare parnum addr external,
  typenum byte external;
  move: proc (source,destin,l) external;
  dcl (source,destin) addr,
    l byte;
  end move;
  error: proc (errcode) external;
  dcl (errcode) addr;
  end error;
  open$srcreadfile: proc external;
  end open$srcreadfile;
  clear$ln$buff: proc external;
  end clear$ln$buff;
  /***********************************************************************************/
  * scanner procedures *
  ***********************************************************************************/
  getchar: proc byte external;
  end getchar;
  getnoblank: proc;
  do while((getchar = '') or (nextchar = eolchar'));
  end;
  end getnoblank;
INIT$SCANNER: PROC PUBLIC;
  DECL COUNT BYTE,
     I BYTE;
  I=0;
  CALL MOVE(PARM1,PARMLIST,E);
  IF PARMLIST(I)='$' THEN
    DC WHILE (COUNT:PARMLIST(I:=I-1)'<>'');
    IF (COUNT:=COUNT-'A')<=4 THEN
      TO CASE COUNT;
      LISTSOURCE = TRUE; /* A */
      LISTLSTF = TRUE; /* B */
      NOSMIPF = TRUE; /* C */
      LISTTCT = TRUE; /* D */
      DEPUG = TRUE; /* E */
    END; /* OF CASE */
  END;
  CALL OPEN$SPC$FILE;
  CALL CLEAR$BUFF;
  CALL GETNOBLANK;
END INIT$SCANNER;

/* SCANER */

SCANNER: PROC PUBLIC;

PUTINACCUM: PROC;
  IF NOT COUNT THEN
    IC;
    ACCUM(ACCUM(I) := ACCUM(I) + 1) = NEXTCHAR;
    HASHOFF = (HASHOFF+NEXTCHAR) AND HASHMASK;
    IF ACCUM(I) = 31 THEN CONT = TRUE;
  END;
END PUTINACCUM;

PUTANDGET: PROC;
  CALL PUTINACCUM;
  CALL GETNOBLANK;
END PUTANDGET;

PUTANDCHR: PROC;
  CALL PUTINACCUM;
  NEXTCHAR = GETCHAR;
END PUTANDCHR;

NUMERIC: PROC BYTE;
  RETURN(NEXTCHAR = '0') <= 9;
END NUMERIC;

LOWERCASE: PROC BYTE;
  RETURN (NEXTCHAR = 61H) AND (NEXTCHAR = 7AH);
END LOWER$CASE;
DECIMALPT: PROC BYTE;
   RETURN NEXTCHAR='.';
END DECIMALPT;

CONV$TOSTUPPER: PROC;
   IF LOWERCASE THEN
      NEXTCHAR=NEXTCHAR AND 5TH;
   END CONV$TOSTUPPER;

LETTER: PROC BYTE;
   CALL CONV$TOSTUPPER;
   RETURN (((NEXTCHAR - 'a') <= 25));
END LETTER;

ALPHANUM: PROC BYTE;
   RETURN NUMERIC OR LETTER;
END ALPHANUM;

SPOOLNUMERIC: PROC;
   DO WHILE NUMERIC;
      CALL PUTANALCHAR;
   END;
END SPOOLNUMERIC;

SET$NEXT$CALL: PROC;
   IF (NEXTCHAR = '.') OR (NEXTCHAR=EOCHAR) THEN
      CALL GETNOBLANK;
      CONT = FALSE;
   END SET$NEXT$CALL;

LOOKUP: PROC BYTE;

  DCL MAXRWLNG LIT 'O';
  DCL PTP ADD. (FIELD BASED PTP) (? BYTE;
  DCL I BYTE;

  COMPARE: PROC BYTE;
     DCL I BYTE;
     I = 0;
     DO WHILE (FIELD(I) = ACCUM(I := I + 1)) AND I <= ACCUM(0);
     END;
     RETURN I > ACCUM(0);
END COMPARE;

IF ACCUM(0) > MAXRWLNG THEN
   RETURN FALSE;
   PTR=VLOC(ACCUM(0))+.VOCAP;
IF I=VNUM(ACCUM(0)) TO
   (VNUM(ACCUM(0))=COUNT(ACCUM(0)));
   IF COMPARE THEN
      I:
      TOKEN=I;

132
RETURN TRUE;
END;
PTR=PTR+ACCUM(&);
END;
RETURN FALSE;
END LOCKUP;

CHECK$EXP: PROC;
/* THIS TAKES CARE OF EXPON. FORM */
IF NEXTCHAR = 'E' THEN
DO:
TYPENUM = UNSIGN$EXPO;
CALL PUTANICCHAR;
IF NEXTCHAR = '+' OR NEXTCHAR = '-' THEN
DO:
CALL PUTANICCHAR;
TYPENUM = SIGNED$EXPO;
END;
CALL SPOOLNUMRIC;
END;
END CHECK$EXP;

SCANNER - MAIN CODE /***/

DO FOREVER:
ACCUM(0), HASHCODE, TOKEN = 0;
IF (NEXTCHAR = STRINGDELIM) OR CONT THEN
DO: /* FOUND STRING */
TOKEN = STRING;
CONT = FALSE;
DO FOREVER;
DO WHILE GETCHAR <> STRINGDELIM;
CALL PUTINACCUM;
IF CONT THEN RETURN;
END;
CALL GETOPBLANK;
IF NEXTCHAR <> STRINGDELIM THEN
RETURN;
CALL PUT$IN$ACCUM;
END; /* OF DO FOREVER */
END; /* OF RECOGNIZING A STRING */
ELSEIF NUMERIC THEN
DO: /* HAVE DIGIT */
TOKEN = NUMERIC;
TYPENUM = INTEGER$TYPE;
DO WHILE NEXTCHAR = '0'; /* ELIMINATE LEADING ZEROS */
NEXTCHAR = GETCHAR;
END;
CALL SPOOLNUMRIC;
IF DECIMALALT THEN
DO;
TEMPCHAR1 = NEXTCHAR;
NEXTCHAR = GETCHAR;
END; /* OF DECIMALALT */
DO
  LOCKED=TRUE; /*HANDLE ... */
  RETURN;
END;
ELSE
DO:
  IF *CHAR2 = NEXTCHAR;
  NEXTCHAR = TEMPCHAR1;
  CALL PUTINSACCUM;
  NEXTCHAR=TEMPCHAR2;
  TYPEPROP = REALSTYPE;
  CALL SPCOLUMNPIC;
END;
END:
CALL CHECKEXP;
IF ACCUM(2) = 0 THEN
  BASECOL,ACCUM(ACCUM(2):1) = '2';
  CALL SP$NEXTSCALL;
  RETURN;
END; /* OF RECOGNIZING NUMERIC CONSTANT */
ELSE IF LETTER THEN
DO; /* HAVE A LETTER */
  DO WHILE ALPHANUM;
    CALL PUTANDCHAR;
  END;
  IF NOT LOCKUP THEN
    TOKEN = IDENTI;
    CALL SP$NEXTSCALL;
    RETURN;
END; /* OF RECOGNIZING FW OR IDENT */
ELSEF DO; /* SPECIAL CHARACTER */
  IF NEXTCHAR = COMMENT THEN
    DO;
      NEXTCHAR = GETCHAR;
      DO WHILE NEXTCHAR <> UNCOMMENT;
        NEXTCHAR = GETCHAR;
      END;
      CALL GET$MOSELANK;
    END;
    ELSEF DO;
      IF NEXTCHAR = ':' THEN
        DO;
          CALL PUTANDCHAR;
          IF NEXTCHAR = '=' THEN
            CALL PUTANDSET;
          END;
        END; /* IF LOCKED THEN */
        DO;
          LOCKED=FALSE;
        END;
104
CALL PUTSIN$ACCM;
NEXTCHAR ='.,'
END;
ELSE
CALL PUTANDCHECK;
IF NEXTCHAR =':' THEN
CALL PUTANTIGET;
ELSE
IF NUMERIC THEN
TO;
TOKEN = NUMBERS;
TYPENUM = REAL$TYPE;
CALL SPOOLNUMERIC;
CALL CHECK$EXP;
CALL SET$NEXT$CALL;
RETURN;
END;
ELSE CALL PUTANDGET;
IF NOT LOOKUP THEN
CALL ERROR("NC");
CALL SET$NEXT$CALL;
RETURN;
END:
END; /* OF RECOGNIZING SPECIAL CHAR */
END; /* OF DO FOREVER */
END SCANNER;
FND SCAN:
DECLARE LIT LITERALLY 'LITERALLY',
DCL LIT 'DECLARE', PUB LIT 'PUBLIC', EXT LIT
'EXTERNAL',
PROC LIT 'PROCEDURE',
TRUE LIT '1',
ALERT LIT 'ALERT',
FALSE LIT '0',
FOREVER LIT 'WHILE TRUE',
STATESIZE LIT 'ALERT',
INDEXSIZE LIT 'ADDRESS', DCL
ITEMSIZE LIT '32', /* MAX IDENTIFIER SIZE = 1 */
VARIABSIZE LIT '128', /* SIZE OF 'ARG STACK' */
PSTACKSIZE LIT '48', /* SIZE OF DAPF STACKS */
HINSTSIZE LIT '128', /* SIZE OF 'HINSTABLE' */
PERSIST LIT '2', /* BYTES USED FOR DCL VALUES */
/* MAXLEVEL LIT '4' */
/* MAXARRAYSIZEM LIT '5' */
/* MAX ARRAY DIMENSIONS */
/* MANY OF THE FOLLOWING VARIABLS CAN BE REPLACED */
/* BY */
/* MAKING USE OF THE PARALLEL DAPF STACKS */
CONSTTYPE BYTE PUB INITIAL (0),
/* TYPE OF CONSTANT */
FORM BYTE PUB INITIAL (0),
PYRONE BYTE PUB INITIAL (0),
VECDPP BYTE PUB INITIAL (0),
MEMMUM BYTE PUB INITIAL (0),
CONST$PTF BYTE PUB INITIAL (0),
TYPES$ADDR ADDR PUB INITIAL (0),
TYPES$LOCT ADDR PUB INITIAL (0),
VAR$PTF BYTE PUB INITIAL (0),
VAR$PTF$PTF BYTE PUB INITIAL (0),
ALLOC$ATIC BYTE PUB INITIAL (0),
APR$COTY (MAXARRAY$DIM) ADDR PUB INITIAL (0),
VAR$BASE (16) ADDR PUB INITIAL (0),
VAR$BASE1(16) ADDR PUB INITIAL (0),
ALLOC$COTY ADDR PUB INITIAL (0),
TYPES$ORDS$NUM BYTE PUB INITIAL (0),
PARENT$TYPE ADDR PUB INITIAL (0),
CONST$INDX BYTE PUB INITIAL (0),
LOOKUP$ADDR ADDR PUB INITIAL (0),
CONST$VEC (4) BYTE PUB INITIAL (0),
CONST$VALUE (16) BYTE PUB INITIAL (0),
CONST$PS$HEAP (4) BYTE PUB INITIAL (0),
CONST$PS$PTR BYTE PUB INITIAL (0),
CONST$SIZEM (4) BYTE PUB INITIAL (0),
INTFGEN$DIFF ADDR PUB INITIAL (0),
SUERS$VAL (2) ADDR PUB INITIAL (0),
LIST$TOKEN BYTE EXT.
COMPILED BYTE INITIAL (0).
/ * COUNTERS */
LARLCOUNT ADDP PUB INITIAL (0), / * COUNTS NUMBER OF LABELS */
ALLOC$AdDP ADDR PUB INITIAL (0), / * COUNTS PRT ENTRIES */
/ * FLAGS USEFUL DURING CODE GENERATION */
CASE$STMT BYTE PUB INITIAL (0), / * IN CASE STATEMENT */
WRITE$STMT BYTE PUB INITIAL (0), / * IN WRITE STATEMENT */
READ$STMT BYTE PUB INITIAL (0), / * IN READ STATEMENT */
NEW$STMT BYTE PUB INITIAL (0), / * GETS NEW RECORD */
DISPOSE$STMT BYTE PUB INITIAL (0), / * DISPOSES OF RECORID */
ALLOCATE BYTE PUB INITIAL (0), / * PRT LOCATION ASSIGNMENT */
VAPPARM BYTE PUB INITIAL (0), / * COMMON PARAMETER IS VARIABLE */
TYPE */
  PARAMS BYTE PUB INITIAL (?), /* DECODING ACTUAL PARAMETERS */
  PRESENT BYTE PUB INITIAL (?), /* IDENTIFIER IS IN SYMBOL TABLE */
  NOTLOOK BYTE INITIAL (?), /* CONTROLS CALLS TO SCANNER */
  SIGNFLAG BYTE PUB INITIAL (?), /* SET WHEN SIGN PRECEDES IT */
  /* GLOBAL VARIABLES USED BY THE SCANNER */
  TOKEN BYTE PUB INITIAL (?), /* TYPE OF TOKEN JUST SCANNED */
  HASHCODE BYTE PUB INITIAL (?), /* HASH VALUE OF CURRENT TOKEN */
  CONT BYTE PUB INITIAL (?), /* INDICATES FULL ACCUM--STILL MORE */
  ACCUM(IDENSIZE) BYTE PUB INITIAL (?), /* HOLDS CURRENT TOKEN */
  /* GLOBAL VARIABLES USED IN SYMBOL TABLE OPERATIONS */
  BASE ADDR PUB INITIAL (?), /* BASE LOCATION OF ENTRY */
  FASHTABLE(FASHTBLSIZE) ADDR PUB INITIAL (?), /* FASHTABLE */
  APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIAL (?), /* APTRAD ADDR PUB INITIA
ECL STATE STATESIZE INITIAL (0),
VAR(PSTACKSIZE) BYTE PUB INITIAL (0),
HAS(PSSTACKSIZE) BYTE PUB INITIAL (0),
STATESTACK(PSSTACKSIZE) STATESIZE INITIAL (0),
PARAMLOC(PSSTACKSIZE) BYTE PUB INITIAL (0), /*
MAINTAINS NUMBER OF PARAMETERS ASSOCIATED WITH A
SUBROUTINE */
LABELSTACK(PSSTACKSIZE) ADDP PUB INITIAL (0), /*
TRACKS STATEMENT LABELS */
PARAMNUMLOC(PSSTACKSIZE) ADDP PUB INITIAL (0), /*
MAINTAINS THE LOCATION IN SYMBOL TPL WHERE PARAMETER
INFO STORED */
BASELOC(PSSTACKSIZE) ADDP PUB INITIAL (0), /* STORES
THE SYMBOL TABLE ADDRESS OF THE PERTINATE ENTRY

109
**/ FORMSFILE(*STACKSIZE) BYTE PUB INITIAL ('z'). **
STOPS THE FORM FILE C**
SCANNED IDENTIFIERS **/
**/ TYPEFILE(*STACKSIZE) BYTE PUB INITIAL ('z'). **
A VARIABLE'S TYPE **/
**/ EXPRES$FILE(*STACKSIZE) BYTE PUB INITIAL ('z'). **
CONTAINS THE TYPES OF THE **
**/ EXPRESSION COMPONENTS
**/

AN IDENTIFIER'S FRT *

LOCATION **/

VARC(VARCOR$IZE) BYTE PUB INITIAL ('z'),
VARINDEX BYTE PUB INITIAL ('7'),
PARAMNUM BYTE PUB INITIAL ('z'),
(SP, WP, WPP1) BYTE PUB INITIAL ('z');

/* NAMEMONICS FOR PASCAL-85 MACHINE **/
DCL MAXNO LIT '165' /*MAX READ COUNT*/, MAXNO LIT '242' /*MAX LOCK COUNT*/, MAXNO LIT '266' /*MAX PUSH COUNT*/;
STARTS LIT '1' /*START STATE*/;

DECLARE READ1(1) BYTE EXT, READ2(1) ADDR EXT, INDEX1(1)
ADDR EXT, INDEX2(1) BYTE EXT, APPLY1(1) BYTE EXT, APPLY2(1)
ADDR EXT, LOCK1(1) BYTE EXT, LOCK2(1) ADDR EXT;

SETUP$PTCL$FIL: PROC EXT; END SETUP$PTCL$FIL;

INIT$SCANNER: PROC EXT; END INIT$SCANNER;

INIT$SYNTH: PROC EXT; END INIT$SYNTH;

ERROR: PROC(ERRCODE) EXT; DECLARE $RECODE ADDR; END PROC;

SCANNER: PROC EXT; END SCANNER;

PRINT$TOKEN: PROC EXT; END PRINT$TOKEN;

SYNTH$SIZE: PROC EXT; END SYNTH$SIZE;

PRINT: PROC(A) EXT;
  TCL A ADDR;
  END PRINT;

CPLF:PROC EXT;
  END CPLF;

TITLE:PROC:
  CALL CPLF;
  CALL PRINT(.('NPS-PASCAL VERS 0.0 7-MAR-85 $'));
  CALL CPLF;
  END TITLE;

NOCONFLICT: PROC (CSTATE) BYTE;
  DCL CSTATE STATE$SIZE, (I, J, X) INDEX$SIZE;

110
$T = \text{INDEX1}(C\text{STATE})$

$K = J + \text{INDEX2}(C\text{STATE}) - 1$

DO $I = J$ TO $K$

IF READ1($I$) = TOKEN THEN RETURN TRUE;

END;

RETURN FALSE;

END NOCONFLICT;

RECOVER: PROC STATESIZE;

DCL TSP BYTE, RSTATE STATESIZE;

DO FOREVER;

TSP = SP;

IF TSP ⟨> 255;

IF NOCONFLICT(RSTATE := STATESTACK(TSP)) THEN

DO; /* STATE WILL READ TOKEN */

IF SP ⟨> TSP THEN SP = TSP - 1;

RETURN RSTATE;

END;

TSP = TSP - 1;

END;

CALL SCANNER;

END;

END RECOVER;

DO: /* BLOCK FOR DECLARATIONS */

DCL (I, J, K) INDEXSIZE, INDEX BYTE;

INITIALIZE: PROC;

CALL INITScanner;

CALL INITSYNTAX;

CALL TITLE;

END INITIALIZE;

GETIN1: PROC INDEXSIZE;

RETURN INDEX1(STATE);

END GETIN1;

GETIN2: PROC INDEXSIZE;

RETURN INDEX2(STATE);

END GETIN2;

INCSP: PROC;

IF (SP := SP + 1) = LENGTH(STATESTACK) THEN

CALL ERROR("SO");

END INCSP;

LOOKAHEAD: PROC:

IF NOTLOOK THEN DO;

CALL SCANNER;

NOTLOOK = FALSE;

IF LISTTOKEN THEN CALL PRINTTOKEN;

END;

END LOOKAHEAD;

SETVARS: PROC(I): /* SET VARS AND INCREMENT VARIINDEX */

*/

111
DCL I BYTE:
VARC(VARINDEX)=I;
IF (VARINDEX=VARINDEX-1) > LENGTH(VARC) THEN
   CALL FRFCP("NC");
END SETSVARCSI;

$SELECT /********************* / *
*/ /* PARSER: EXECUTION BEGINS HERE */ /* */
/*****************************/

CALL SETUPSTFIL; /* CREATES OUTPUT FILE FOR GENERATED
CODE */ CALL INITIALIZE; COMPILING,NOLOCK=TRUE;
STATE=SSTARTS; SP=255; VARINDEX,VAP(0) = 0; DO WHILE
COMPILING;
   IF STATE=MAXPNC THEN /* READ STATE */
      DO:
         CALL INCSP:
         STACKTOPV(SP)=STATE:
         I=GETIN1;
         CALL LOOKAHEAD;
         J=I+GETIN2-1;
         DO I=1 TO J;
            IF READ1(I)=TOKEN THEN /* SAME TOKEN */
               DO; /* COPY ACCUM TO PROPER POSITION */
                  VAP(SP)=VARINDEX;
                  DO INLPX = 2 TO ACCUM(2);
                  CALL SETSVARCSI(ACCUM(INDEX));
               END;
               HASH(SP) = HASHCODE;
               STATE=REAL2/I):
               NOLOCK=TRUE;
               I=J;
               END;
            ELSE
               IF I=J THEN
               DO;
                  CALL FRFCP("NP");
                  IF (STATE := RECOVER)=2 THEN
                     COMPILING = FALSE;
                  END;
               END;
            END:
         END;
      END;
      IF STATE=MAXPNC THEN /* APPLY PRODUCTION STATE */
      DO:
         MP=SP-GETIN2;
         MMP1=MP+1;
         PRODUCTION = STATE-MAXPNC;
         CALL SYNTHESIZE;
         SP=MP;
         I=GETIN1;
         VARINDEX=VAR(SP);
      END;
      CALL SETUPSTFIL; /* CREATES OUTPUT FILE FOR GENERATED
CODE */ CALL INITIALIZE; COMPILING,NOLOCK=TRUE;
STATE=SSTARTS; SP=255; VARINDEX,VAP(0) = 0; DO WHILE
COMPILING;
   IF STATE=MAXPNC THEN /* READ STATE */
      DO:
         CALL INCSP:
         STACKTOPV(SP)=STATE:
         I=GETIN1;
         CALL LOOKAHEAD;
         J=I+GETIN2-1;
         DO I=1 TO J;
            IF READ1(I)=TOKEN THEN /* SAME TOKEN */
               DO; /* COPY ACCUM TO PROPER POSITION */
                  VAP(SP)=VARINDEX;
                  DO INLPX = 2 TO ACCUM(2);
                  CALL SETSVARCSI(ACCUM(INDEX));
               END;
               HASH(SP) = HASHCODE;
               STATE=REAL2/I):
               NOLOCK=TRUE;
               I=J;
               END;
            ELSE
               IF I=J THEN
               DO;
                  CALL FRFCP("NP");
                  IF (STATE := RECOVER)=2 THEN
                     COMPILING = FALSE;
                  END;
               END;
            END:
         END;
      END;
      IF STATE=MAXPNC THEN /* APPLY PRODUCTION STATE */
      DO:
         MP=SP-GETIN2;
         MMP1=MP+1;
         PRODUCTION = STATE-MAXPNC;
         CALL SYNTHESIZE;
         SP=MP;
         I=GETIN1;
         VARINDEX=VAR(SP);
      END;
```plaintext
I = STATESTACK(SP);
DO WHILE (Z = APPLY1(I)) < 3 AND J < 3;
    I = I + 1;
END;
IF (STATE = APPLY2(I)) = 3 THEN
    COMPILING = FALSE;
END;
ELSE
    IF STATE = MAXINO THEN /* LOOKAHEAD STATE */
        DO:
            I = GETINI;
            CALL LOOKAHEAD;
        END;
        IF WEIIF (Z = LOCK1(I)) < 3 AND TOKEN < 3;
            I = I + 1;
        END;
        STATE = LOCK2(I);
        END;
    ELSE
        DO: /* PUSH STATE */
            CALL INCSP;
            STATESTACK(SP) = GETINI;
            STATE = GETINI;
        END; /* OF WHILE COMPILING */
    END; /* OF BLOCK FOR PARSER */
END PARSER;
```
<table>
<thead>
<tr>
<th>TABLES:</th>
<th>DCL</th>
<th>$PAGEWIDTH(80) TITLE('TABLES - LALP/1: PAGE OF TABLES')</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DCL</td>
<td>LIT LITERALLY 'LITERALLY'.</td>
</tr>
<tr>
<td></td>
<td>ADD</td>
<td>LIT 'ADDRESS',</td>
</tr>
<tr>
<td></td>
<td>DCL</td>
<td>LIT 'DECLASS',</td>
</tr>
<tr>
<td></td>
<td>PUB</td>
<td>LIT 'PUBLIC';</td>
</tr>
<tr>
<td></td>
<td>DCL</td>
<td>DUMMY (3) BYTE DATA (2, 2.3); /* DUMMY FILLER TO FORCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BUILTSINSTABLE TO 1265 */</td>
</tr>
<tr>
<td></td>
<td>DCL</td>
<td>BUILTSINSTABL (<em>): BYTE P'3 /</em> AT (1062) */ DATA (</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3, 3.2, 42F, 147, 'I', 'N', 'T', 'E', 'R',</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3, 0.1E, 0DE, 445, 567, 9, 'T', 'A',</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.0, 0.31F, 167, 52H, 32.4, 'C', 'F', 'A',</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3, 0.1E, 1EP, 5AE, 0, 7, 'E', 'O', 'C', 'L', 'E', 'T', 'A',</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.0, 0.1E, 24F, 628, 69.4, 'T', 'X', 'Y', 'A',</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.2, 31F, 36P, 05F, 16.5, 'I', 'N', 'P', 'T',</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.1F, 16E, 113.6, 'C', 'T', 'P',</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.1F, 14H, 0TT, 3D.3, 'A', 'S', 'D', 'E',</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3, 0.1F, 06E, 116.3, 'S', 'C', 'O', 'P', 113F, 1.13H,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3, 0.1F, 0AF, 0DE, 166.3, 'S', 'T', 'P', 23F, 1.13F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3, 0.1F, 0EF, 0DE, 121.3, 'C', 'G', 'S', 'C', 43F, 1.13F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3, 0.1F, 06F, 0DE, 57.6, 'A', 'P', 'C', 'F', 'A', 'N', 43F, 1.13H,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.1F, 04F, 0DE, 146.3, 'E', 'P', 53F, 1.13F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.1F, 245F, 0DE, 26.2, 'L', 'W', 'E', 'H', 1.13F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.2, 31F, 03F, 0DE, 74.4, 'S', 'C', 'R', 'T', 73F, 1.13F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.1F, 0DE, 67.3, 'C', 'T', 'P', 2.5E, 1E,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.1, 0.1F, 0CF, 0DE, 46.4, 'F', 'O', 'L', 'N', 53E, 1.067,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.2, 31F, 03D, 0DE, 92.3, 'T', 'O', 'F', 13.5F, 1.06F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.1F, 0CH, 0DE, 12.5, 'T', 'R', 'U', 'N', 'C', 11.1F, 1.13F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.7, 0.1F, 0AF, 0DE, 8.5, 'B', 'O', 'U', 'N', 'D', 1P, 13.113F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0CH, 01F, 02F, 0DE, 121.3, 'O', 'R', 'D', 13.1H, 1.12F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0F, 0.02F, 1AF, 0DE, 93.2, 'C', 'P', 'R', 14, 2D, 1.12F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0DDF, 01F, 02F, 0DE, 46.4, 'S', 'U', 'C', 'O', 'C', '1E', 073F, 1.073F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3, 0.32F, 45F, 0DE, 43.4, 'P', 'O', 'F', 'E', 16, 073F, 1.073F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.02F, 45F, 0CE, 121.3, 'P', 'U', 'T', 17, 17F, 06F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.02F, 54F, 0CE, 95.1, 'C', 'E', 'T', 18, 10F, 06F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3, 0.32F, 61F, 0CE, 33.5, 'R', 'T', 'S', 'T', 'T', 19, 17F, 06F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.02F, 6FE, 0CE, 34.7, 'R', 'S', 'W', 'R', 'I', 'T', 'E', 'F', 2D.10F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3, 0.23F, 7DF, 0CH, 28.4, 'P', 'A', 'S', '21, 17H, 06F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>07E, 0.01F, 02F, 0EH, 06F, 10F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.02F, 0CE, 0CH, 22.7, 'D', 'N', 'T', 'S', 'P', 'O', 'S', 'E', 27, 0FFH,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.0, 0.32F, 3AE, 09H, 64.4, 'T', 'E', 'U', 'S', 'T', 0, 3, 0, 3,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.02F, 0DE, 107.5, 'F', 'A', 'L', 'S', 'E', 0.0, 1, 0,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.02F, 0CE, 0CH, 29.4, 'P', 'E', 'A', 'T', 'D', 24, 0FFH,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.02F, 0CE, 11.5, 'W', 'R', 'I', 'T', 'E', 'F', 26, 0FFH,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0, 0.02F, 0CE, 37.7, 'W', 'B', 'I', 'T', 'E', 'L', 'O', 27, 0FFH,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.9, 0.23H, 01E, 0CH, 40.4, 'S', 'E', 'K', 2D, 2, 0FF, 01F,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>114</td>
</tr>
</tbody>
</table>
DCL INDEX1(*) ADR PUB INITIAL

SYMBOL.MCL

$SPACEWITH(80) TITLE("SYMBOL - SYMBOL TABLE ROUTINES")

SYMBOL.MCL

DECLARE LIT LITERALLY 'LITERALLY',
DCL LIT 'DECLARE',
POS LIT '0',
NEG LIT '1', PUB LIT 'PUBLIC', VAT LIT 'EXTERNAL',
PROC LIT 'PROCEDURE',
TRUE LIT '1',
ADDL LIT 'ADDRESS',
FALSE LIT '0',
QUIT SIZE FUNC LIT 'QBE'; DCL
IDENTSIZE LIT '32', /* MAX IDENTIFIER SIZE - 1 */
VARGSIZE LIT '10', /* SIZE OF VARG STACK */
FSTACKSIZE LIT '40', /* SIZE OF PARSER STACK */
HASPTRSIZE LIT '128', /* SIZE OF HASHABLE */
HASHMASK LIT '127', /* HASH TABLE SIZE - 1 */
MAXINT LIT '2767', /* MAX INTEGER */
BCOSIZE LIT '8', /* BYTES USED FOR ECT VALUES */
MAXNEST LIT '3', /* MAX LEVEL OF NESTS FOR TYPES */
MAXARRY DIM LIT '5', /* MAX ARRAY DIMENSIONS */
FORMASK LIT '7', /* USED TO DETERMINE FORM TYPE */
/* FORM ENTRIES */
CONSTENTRY LIT '1',
TYPEENTRY LIT '2',
VARENTRY LIT '3',
FUNCTIONENTRY LIT '5', /* NUMBER TYPES */
INTEGERSTYPE LIT '1',
SIGNEXPCN LIT '4',
REALTYPE LIT '2',
PARAM LIT '6',
LOCAT LIT '78',
PARAM LIT '69';
/* MANY OF THE FOLLOWING VARIABLES CAN BE REPLACED
BY MAXING USE OF THE PARALLEL PARSER STACKS */
DCL
FORM BYTE EXT,
EXPCN BYTE EXT,
VFPTR BYTE EXT,
TYPEENTRY BYTE EXT,
CONSTPTL BYTE EXT,
STARTDOS ADDR EXT, /* ADDR OF PTR TO TOP OF PRODS */
TYPESLOT ADDR EXT,
VARPTR BYTE EXT,
VARSIZE(10) ADDR EXT,
ALLOC AT ADDR EXT,
CONSTINDEX BYTE EXT,
LOOKUPSADDR ADDR EXT,
CONSTVALUE(16) BYTE EXT,
CONSTPN$PHASE(4) BYTE EXT,
CONSTPN$PTR BYTE EXT,
CONST NULL SIZE(4) BYTE EXT,
CURR(SET (MAX OFFSET)) ADDR EXT,
/* CASE STATEMENT VARIABLES */
CONST NUM STUFF(4) BYTE EXT; /* GLOBAL VARIABLES */

I C L
E C T ( E C T S I Z E ) B Y T E E X T ,
SCOPE ( 10 ) ADDR EXT,
SCOPES NUM BYTE EXT,
TEMP BYTE EXT,
TEMP ADDR ADDR EXT,
TEMP ADDR ADDR EXT,
PREVIOUS ENTRY ADDR EXT;

E C L
/* COMPILE TOGGLES */

/* COUNTERS */
LAPL COUNT ADDR EXT, /* COUNTS NUMBER OF LABELS */
ALLOC ALL ADDR EXT, /* COUNTS PAT ENTRIES */

/* FLATS USED DURING CODE GENERATION */
REAPARMS BYTE EXT, /* READING ACTUAL PARAMETERS */
PRESENT BYTE EXT, /* IDENTIFIER IS IN SYMBOL TABLE */

/* SIGN FLAT BYTE EXT, /* SET WHEN SIGN PRECEDES ID */

/* GLOBAL VARIABLES USED BY THE SCANNER */
HASH FLAT BYTE EXT, /* HASH VALUE OF CURRENT TOKEN */

/* GLOBAL VARIABLES USED IN SYMBOL TABLE */
OPERATIONS /*
BASE ADDR EXT, /* BASE LOCATION OF ENTRY */
HASH TABLE (HASH TABLE SIZE) ADDR EXT, /* HASH TABLE ARRAY */

/* SETUP */
SETPTOP ADDR EXT, /* HIGHEST LOCATION OF SYMBOL */
SSTBP ADDR EXT, /* CURRENT TOP OF SYMBOL TABLE */
APTRADDR ADDR EXT, /* UTILITY VARIABLE TO ACCESS */

SETUP */
ADDR PTR BASED APTRADDR ADDR, /* CURRENT 8 BYTES */
POINTER AT /*
(ADDRPTR EASILY APTRADDR) (1) BYTE, /* CURRENT BYTE */
POINTER AT /*
PRINT NAME ADDR EXT, /* SET PRIOR TO LOOKUP OR ENTER */
SYMPHAT BYTE EXT, /* HASH VALUE OF AN IDENTIFIER */
LASTSHTBL%d ADDR EXT, /* HOLD PREVIOUS BASE */

LOCATION */
PARAM NUM LOC ADDR EXT, /* STORES POINTER TO PARAM */
LISTING */
SPTBLSCAPE ADDR EXT, /* BASE OF LAST ENTRY IN */
PREVIOUS BLOCK */
SP BYTE EXT,
MP BYTE EXT.

120
PARXVIMLOC (PSTACKSIZE) ADDR EXT,
PARAMARN EYTE EXT,
POSTADDH (PSTACKSIZE) ADDR EXT,
EXPRESS EYTE (PSTACKSIZE) BYTE EXT,
FORMFIELD (PSTACKSIZE) BYTE EXT,
VAR (PSTACKSIZE) BYTE EXT,
VARC (VARCRESIZE) BYTE EXT,
HAEK (PSTACKSIZE) BYTE EXT:

/*/ DECLARE EXTERNAL PROCEDURES, FOUND IN SYSPTS */

GENERATE: PROC (OBJCODE) EXT;
 LCL OBJCODE BYTE;
END GENERATE;

ERROR: PROC (ERRCODE) EXT;
 LCL ERROCODE ADDR;
END ERROR;

MOVE: PROC (SOURCE, DESTIN, L) EXT;
 LCL (SOURCE, DESTIN) ADDR;
 DCL L BYTE;
END MOVE;

MCN: PROC EXT;
END MCN;

GENADDR: PROC (A, B) EXT;
 LCL A BYTE, B ADDR;
END GENADDR;

/****** SET ADDR $P1 - THIS PROCEDURE SETS A **********
 * POINTER TO A SPECIFIC LOCATION IN THE *
 * SYMBOL TABLE. *
*****************************/

SFTADDRPTR: PROC (OFFSET) PUB;
 LCL OFFSET BYTE;
 APTRADDR = BASE + OFFSET;
END SFTADDRPTR;

/****** SET$PAST$PRINAME - THIS PROCEDURE SETS ******
 * APTRADDR TO A LOCATION IN A SYMBOL TABLE *
 * ENTRY THAT IS PAST THE ENTRY'S PRINAME *
 * (WHICH IS OF VARIABLE LENGTH). *******/

SFT$PAST$PRINAME: PROC (OFFSET) PUB;
 LCL OFFSET BYTE;
 CALL SFTADDRPTR (6);
 CALL SFTADDRPRT (BYTEPTR 'O') + OFFSET);
END SFT$PAST$PRINAME;

/****** CALC$VAPC - THIS PROCEDURE DETERMINES THE ******
 * LOCATION OF AN IDENTIFIER PRINTNAME. *******/

121
CALCVARC: PROC(A) ADDR PUB;
   COL A BYTF;
   RETURN. VAR(A) + .VARC;
END CALVCVARC;

SETLOCKUP - THIS PROCEDURE IS UTILIZED TO *
   FIND THE HASH VALUE OF AN IDENTIFIER. *
   MODELS:
   P NAME "CLC V A C L";

SETLOCKUP: PROC(A) PUB;
   COL A BYTF;
   PRINTNAME = CALCSVARC(A);
   SYMBALE = HASH(A); /" HASHCODE OF PU */
END SETLOCKUP;

ENTERSLINKS - THIS PROCEDURE ENTERS IN THE */
   NEXT FOUR BYTES OF THE SYMBOL TABLE THE */
   COLLISION FIELD AND THE PREVIOUS SYMBOL */
   TABLE ENTRY ADDRESS FIELD FOR THE NEXT */
   SYMBOL TABLE ENTRY. ( BOTH IN ADDRESS VAR ) */
   MODELS:
   P NAME "CLC V A C L";

ENTERSLINKS: PROC PUB;
   BASE, APTRADD, SETBLSOPE = SBTPL;
   SSCOPE(SCOPSCNUM) = SBTPL;
   ADDRPR = TABLESPEL(SYMBALE);
   CALL SETADDPRTR(A);
   ADDRPR = PRVSSPENTRY;
   PRVSSPENTRY = SETBL;
   EASETABLE(SYMBALE) = BASE;
END ENTERSLINKS;

CHECKPRINTNAME - THIS PROCEDURE DOES A */
   CHARACTER TO CHARACTER COMPARISON BETWEEN */
   THE CURRENTLY RECOGNIZED IDENTIFIER AND */
   SYMBOL TABLE ENTRIES OF THE SAME BASE VALUE. */
   MODELS:
   P NAME "CLC V A C L";

CHECKPRINTNAME: PROC(A) BYTE PUB;
   /* A IS OFFSET FROM BASE TO PRINTNAME */
   DCL(N BASED PRINTNAME)(1) BYTF;
   ECL (LEN,A) BYTE;
   CALL SETADDPRTR(A);
   IF ( LEN := BYTFPTR(0) ) = N(0) THEN
   DO WHILE (BYTEPTR(LEN)=N(LEN));
     IF ( LEN := LEN-1 ) = 0 THEN
       RETURN TRUE;
   END;
   RETURN FALSE;
END CHECKPRINTNAME;

LOOKUPPRINTNAME'SIDENTITY - THIS PROCEDURE */
   IS PASSED THE LOCATION OF AN IDENTIFIER IN */
   THE PRODUCTION RULE, AND ITS TARGET ENTRY */
   TYPE. IF THE IDENTIFIER IS FOUND WITH THE */
   CORRECT TYPE THE PROCEDURE RETURN TRUE; */

122
IY
IS
RB
.......
~
'kLL
S-
IoKUP(.A);
2
A JA
S-JTA?Ll:
SYt'1T ASL ~
c
k
TL
SB?-.LD~r'7(4);
!F
((
-YFPT.:z(Z0)
A~n
710.,M'kSy
I%!7NT? -
IF CFK PRTeNPIF(e)
N
u'
((BASE < SCCPt?%'))
Cz
O3AS-
>=
OR
((IrSZN'?Y
= TYPESEPJ7RY)
A':'!ST
SCOPE(SCcE$,JT)
T-47N
DO;
L-OKUF $1.
=E-ASF;
RETL
T
PN TRUF;
CALL
SETAr:RpF:B(e';
3 AS7
ADDR?TD;
RETURN
F-ALSE;
TEND
LOQTU? ?NID
T~
LI V1IS
-T I S
R,)
C
I
UP
p
23JVS
C~
Sv3L
T
_~
EN~z
v AB3CUT TO
~
7RL~
/* AVAILABLE
SYV
10L
TABmLE A.ZRESSE.
/*/L.
T.
PAAEE
S
TE
3YTFCOUN7
OF E
/*/FNTRY
TO
BEENTERED.

LI
M
lITS : P?OC (COTN
T)
PUB;
DCL COUNT
BYTE;
I' SETBLTOP <= (SBTEL + COUNT) T7EN
DO;
CALL FPOR('TO');
CALL MON3;
END;
END LIMITS;
/*/ ENTP$PINNAMESIDENTITY - THIS PROCEDURE */
/*/ LOADS THE SYMBOL TABLE WITH THE FOLLOWING: */
/*/ 1. COLLISION FIELD */
/*/ 2. PREVIOUS SYMBOL TABLE ENTRY ADDRESS */
/*/ 3. FORM OF ENTRY ( PRESET BYTE "FORM" ) */
/*/ 4. THE LENGTH OF THE PRINTNAME IN ONE BYTE*/
/*/ 5. THE PRINTNAME CHARACTERS */
/*/ PARAMETER: PRINTNAME IS SET PRIOR TO CALL. */
/*/ ENTP$PINNID:PROC PUB;
DCL I BYTE;
DCL (N BASED PRINTNAME)(1) BYTE;
CALL LIMITS(I:=N(Ø)-7);
CALL ENTP$PINNID;
CALL SETADDRPTR(4);
BYTEPTR(2) = ERROR;
CALL SETADDRPTR(4);
BYTEPTR(2) = SYMFILE;
CALL SETADDRPTR(4);
BYTEPTR(2) = N(2);
CALL MOVE(PRINTNAME-1, S3TBL+7, N(2));
LAST$SETELSID = SETEL;
S3TBL = S3TBL + 1;
END ENTER$PNSID;
/* ******************************************** /
/* ENTER$PNSID - THIS PROCEDURE  */
/* CALLS ENTER$PNSID TO LOAD THE SYMFILE TABLE */
/* ENTRY CURRENTLY BEING SCANNED. IT ALSO */
/* GENERATES THE ENTRY'S "FORM" BY PERFORMING */
/* A BOOLEAN 'OR' OPERATION ON THE ID$ENTRY */
/* AND THE PARAMETER 'A'. */
/* ******************************************** /
ENTRY$VAR$ID: PROC(A,B, ID$ENTRY) PUB;
ENTRY (A,B, ID$ENTRY) BYTE:
ENTRY@PNSID(B, ID$ENTRY) THEN
DO;
PRESENT = TRUE;
RETURN;
END;
/* ELSE ENTRY VAR NAME */
PRESENT = FALSE;
FORM = A OR ID$ENTRY;
CALL ENTRY$PNSID;
IF ID$ENTRY = VAR$ENTRY THEN
DO;
CALL LIMITS(4);
VAR$BASEL(VAR$PTR) = S3TBL;
S3TBL = S3TBL + 4;
END;
END ENTRY$VAR$ID;
/* ******************************************** /
/* SET$LABEL - THIS PROCEDURE  */
/* TO THE CURRENT DECLARED LABEL AND INCREMENT */
/* THE LABELCOUNT ( NEXT TO ASSIGN ). */
/* ******************************************** /
SET$LABEL: PROC PUB:
ADDPTR = LABELCOUNT;
LABELCOUNT = LABELCOUNT + 1;
END SET$LABEL;
/* ******************************************** /
/* ENTER$LABEL - THIS PROCEDURE  */
/* ENTRY INTO THE SYMBOL TABLE. SYMFILE AND */
/* PRINTNAME MUST BE SET PRIOR TO CALLING */
/* ******************************************** /
ENTRY$LABEL: PROC PUB:
CALL LIMITS(2);
APTRADER = SETEL;
CALL SET$LABEL;
124
ENTP$PRT$LOC: PROC PUB;
DCL (I,P) BYTE;
CALL SET$PAST$PNT(7);
P = BYTEPTR(Q);
PARAM$NUMLOC = APTRADDR;
DO I = 1 TO P;
   CALL ST$PAST$PNT(P);
   APTRADDR = APTRADDR + 1;
   ADDPTR = ALLOCADDR;
   APTRADDR = TEMPADDR;
   APTRADDR = APTRADDR + 6;
   APTRADDR = APTRADDR + 1 + BYTEPTR(Q);
   ADDPTR = ALLOCADDR;
   ALLOCADDR = ALLOCADDR + ALLC$HTY;
   TEMPADDR = APTRADDR + 4;
END;
END ALTEPS$PRT$LOC;

ENTP$SUB$PNT: PROC (A,P,IT$ENTRY) PUB;
DCL (A,P,ID$ENTRY) BYTE;
CALL ENTP$VAR$ID(Q, SP, ID$ENTRY);
IF NOT PRESENT THEN
   DO;
      CALL LIMITS(4);
      PARAM$NUMLOC = SETL;
      SETL = SETL + 3;
      CALL SET$PAST$PNT(10);
      ADDPTR = ALLOCADDR; ALLOCADDR = ALLOCADDR + 2;
      CALL SET$PAST$PNT(14);
      ADDPTR = LABLCOUNT;
      LABLCOUNT = LABLCOUNT + 2;
      SETL = SETL + 6;
      IF ID$ENTRY = Func$ENTRY THEN
         125
DO;
    S'TBL = S'TBL + 1;
END;
ELSE DO; /* FORWARD FUNCTION */
    CALL SET$PAST$PN(14);
    IF ID$ENTRY = FUNC$ENTRY THEN TEMPADDR1 = APTRADDR - 3;
    ELSE TEMPADDR1 = APTRADDR + 2;
    CALL SET$PAST$PN(17);
    ADDR = ALLOCADDR;
    ALLOCADDR = ALLOCADDR + 2;
    CALL ALTER$PTR$LOG;
END;
PAR$NUMLOC(MP) = BASE;
SCOPT(SCOPT$NUM := SCOPT$NUM + 1) = S'TBL;
END FNTP$SUBR;

LOOK$ONLY: PROC(A) BYTE PUB;
DCL A BYTE;
    CALL SETLOOKUP(A);
    BASE = EAS TABLE(SYMPHAS);
    DO WHILE BASE <> 3;
        IF CFK$PPTY$NAME(6) THEN DO;
            LOOKUP$ADDR = FAST;
            RETURN TRUE;
        END;
        ELSE DO;
            CALL SETADDRPTR(0);
            BASE = ADDRPTR;
        END;
    END;
    RETURN FALSE;
END LOOK$ONLY;

CONV$BCD: PROC(A, T) PUB; /* A=SP/MP/MPP1, B=PCS/NEG */
DCL (I,J,DFLAG,FLAG,SFLAG,A,T) BYTE;
DCL (IN BASED PRINTNAME)(1) BYTE;
DCL (EXPO.NLOOP,EXPSIGNLOOP) LABEL;
CALL SETUPLOOKUP(A);
/* INITIALIZE VARIABLES */
SFLAG = FALSE; EFLAG = TRUE; DFLAG = TRUE; I = 1;
DO J = 0 TO 7; BC$INUM(J) = 0; END;
H(i~ T~V'AIJNC GZ PCS IF I=Z+/ THEN &OO XPON :LOOP;
P" 
LCPAD BCDNUM .ITF SIGNIFICANT r CITIS ~
PC w/FLF (N (I)-' <= 9 CIR N(I) ~
IF 'J(I) THEN DO; FFLAG=FA 
I7 I=N(o')
179 I=N(o) TEEN FLA' EXPONLnoC?;
IF TBCPsizF-1 ) THIFN Co 
FY:XC.4LCC?;
IF EFLAG THEN FXPON=EXC,4+l; 
DO; j = j + 1;
1 = I
DO; j = j + 1;
1 = I
IF i=(N(c1+i) THEN 
FEr; IF i=(N(c1+i) THEN 
FN
J=2; EXPON=64; /* P+CE */
/* H expert leading ZEPCS /*
DO WHILE ((N(I) - "0") = 0);
I=I+1;
IF I=(N(2)+1) THEN GOTO EXPONLOOP;
END;
/* load BCDNUM with significant digits */
PC WHILE ((N(I) - "0") <= 0 OR N(I) = ".";
IF N(I) = "." THEN
DO; EFLAG=FALSE;
IF I=N(0) THEN GOTO EXPONLOOP;
I = I + 1;
END;
ELSE
DO;
WHILE J = 0 AND EFLAG AND (N'I) - "0") = 0;
EXPON = EXPON-1;
IF I = N(0) THEN GOTO EXPONLOOP;
I = I + 1;
END;
IF J = ( BCDSIZE-1 ) THEN GOTO EXPONLOOP;
IF EFLAG THEN /* first ECT pair */
DO:
BCDNUM(J)=$LL((N(I)-"Z"),4);
EFLAG=FALSE; I= I+1;
IF EFLAG THEN EXPON=EXPON+1;
END;
ELSE
DO:
BCDNUM(J)=BCDNUM(J)+(N(I)-"3");
J = J + 1; I = I + 1;
EFLAG=TRUE; IF EFLAG THEN EXPON=EXPON-1;
END;
IF I=(N(6)+1) THEN GOTO EXPONLOOP;
END;
EXPONLOOP:
IF N(I) = "." THEN EFLAG = FALSE;
IF I = (N(2)+1) THEN GOTO EXPSIGNLOOP;
IF EFLAG THEN DO;
DO WHILE N(I) <> ".";
EXPON = EXPCK + I;
I = I + 1;
END;
I = I + 1;
END;
DO WHILE I < (N(3)+1) AND (N(I)-"3") <= 9 ;
I = I + 1;
END;
IF TYPENUM = REALTYPE THEN GOTO EXPSIGNLOOP;
/* N(I) = "Z */ I = I+1;
IF TYPENUM = SIGNED$XPON THEN DO;
IF \( N(I) = \text{ZER} \) THEN SFLAG = \text{TRUE};
I = I + 1;
END;
IF I = N(J)+1 THEN
DO:
  CALL ERROR('IF');
  RETURN;
END;
DFLAG = \( \Phi \);
DO J = I TO N(J);
  DFLAG = (DFLAG*10)+(N(J)'\text{'}\Phi');
END:
IF SFLAG THEN /* EXPONENT CALCULATION */
  EXPON = EXPON-DFLAG;
ELSIF EXPON = EXPON + DFLAG:
ENDSIGNLOOP:
  ECDNUM(ECDSIZE-1)=RCL(2,7); /* SIGN OF NUMBER */
  IF EXPON > 127 THEN
    DO:
      CALL ERROR('ZE');
      RETURN;
    END;
  ELSE ECDNUM(ECDSIZE-1)=ECDNUM(ECDSIZE-1)+EXPON;
  END
END

******************************************************************************
/* CONVPTTI - THIS PROCEDURE IS PASSED "A", "B" */
/* LOCATION OF A CONSTANT IN THE PRODUCTION */
/* AND "B" THE 'SIGN' OF THE INTEGER. THE */
/* FUNCTION GENERATES A SIGNED 16 BIT REPRESENTATION OF THE NUMBER AND RETURNS IT IN */
/* AN ADDRESS VARIABLE. */
******************************************************************************
CONVPTTI: PROC(A, B) ADDRESS PUB;
  DCL (I,A, B) BYTE;
  DCL (N FASD PRINTNAME)(1) BYTE;
  DCL NUM ADDR;
  CALL SETLOOKUP(A); NUM=0;
  DO I=1 TO N(\( \Phi \));
    IF (MAXINT/10) \( \geq \) NUM THEN
      DO:
        IF (MAXINT/10) = NUM AND (N(I)'\text{'}\Phi') \( > \) ? THEN
          DO;
            CALL ERROR('IE');
            RETURN NUM;
          END;
          NUM=(NUM*10)+(N(I)'\text{'}\Phi');
        END;
        ELSE DO;
          CALL ERROR('IE');
          RETURN NUM;
        END;
      END;
    IF B = \text{POS} THEN RETURN NUM;
  IF NUM = MAXINT THEN
    128
DO:
   CALL PROB( 'IF' );
   RETURN NUM;
END;
   RETURN ( - NUM );
END CONVERT;

CONVRT$CONST: PROC (A) PUB; /* A=POS,NEG */
DCL A BYTE, INT$ADDR ADDR;
IF TYPE$NUM = INTEGER$TYPE THEN
   DO:
      INT$ADDR = CONVERTI (SP, A);
      CONST$NUM$TYPE (CONST$PTR) = INTEGER$TYPE;
      CONST$PTR = CONST$PTR+1;
      CALL MOVE (.INT$ADDR, .CONST$VALUE (CONST$INDX), 0);
      CONST$INDX = CONST$INDX - 2;
   END;
ELSE DO:
   CALL CONVRTBCI (SP, A);
   CONST$NUM$TYPE (CONST$PTR) = REAL$TYPE;
   CONST$PTR = CONST$PTR+1;
   CALL MOVE (.B$NUM,..., .CONST$VALUE (CONST$INDX), PCT$SIZE);
   CONST$INDX = CONST$INDX + PCT$SIZE;
END;
END CONVRT$CONST;

ENTRY$CONST$NUMBER: PROC PUB; /* A=POS,NEG */
CONVRT$PTR = ENTRY$PTR - 1;
IF ENTRY$NUM$TYPE (ENTRY$PTR) = INTEGER$TYPE THEN
   DO:
      CALL SETADDRPTR (4); BYTEPTR (2) = & OR CONVN$ENTRY;
      CALL LIMITS (2); CONST$INDEX = CONST$INDEX - 2;
      CALL MOVE (.CONST$VALUE (CONST$INDX), S$TBL, 2);
      S$TBL = S$TBL + 2;
   END;
ELSE DO:
   CALL SETADDRPTR (4); BYTEPTR (2) = 128 OR CONVN$ENTRY;
   CALL LIMITS (BCDSIZE); CONST$INDEX = CONST$INDEX + BCDSIZE;
   CALL MOVE (.CONST$VALUE (CONST$INDX), S$TBL, BCDSIZE);
   S$TBL = S$TBL + BCDSIZE;
END;
ENTE$STRING: PROC(A) PUB;
DCL (N BASED PRINTNAME)(1) BYTE;
DCL A BYTE;
CALL SFTLOOKUP(A);
CALL LIMITS(N(2)+1);
CALL MOVE(PRINTNAME,SBT3L,(N(2)+1));
SBTBL=SBTBL+(N(2)+1);
END ENTERSTRING;

ENTE$CONS$NUPM: PROC(A,B,C) PUB; /* A=POS/NEG , B=MP/USP1/SP */
DCL (A,B,C) BYTE;
C=POL(A,C);
CALL SETADDRPTR(4); BYTEPTR(2)=C OR CONS$ENTRY;
CALL ENTERSTRING(SB);
CONST$PN$PTR=CONST$PN$PTR-1;
CONST$INX=CONST$INX-CONST$PN$SIZE(CONST$PN$PTR);
END ENTER$CONS$N;

ENTE$CONS$NTRY: PROC PUB;
VEC$PTR=VEC$PTR-1;
DO CASE EXPRESS$STK(SP);
/* CASE CONSTANT NUMBER */
CALL ENTER$CONS$NUM;
/* CASE IDENTIFIER CONSTANT */
CALL ENTER$CONS$ID(POS,SP);
/* CASE SIGNED IDENTIFIER CONSTANT */
CALL ENTER$CONS$ID(NEG,SP);
/* CASE CONSTANT STRING */
DO;
CALL SETADDRPTR(4); BYTEPTR(2)=1SH OR CONS$ENTRY;
CALL ENTERSTRING(SB);
CONST$PN$PTR=CONST$PN$PTR-1;
CONST$INX=CONST$INX-CONST$PN$SIZE(CONST$PN$PTR);
END;
END ENTER$CONS$NTRY;
ENTPS$CPLX$STYP: PROC(A) PUB;
LCL A BYTE;
CALL LIMITS($);
3ASP$APTRADD$=SBT$L;
ADRPTR=$2$O$H;
CALL SETADJ$0$RPTR($);
ADFPTR=PY$V$SBT$ENTRY$;
PS$SBT$ENTRY$=BASE;
CALL SETADJ$0$RPTR($);
RTF$PTR(0)=$;
SBTR$=SBTR$+5;
END ENTSP$CPLX$STYP;

ENTSP$STR$STYP: PROC(A) PUB;
LCL A BYTE;
CALL ENTSP$CPLX$STYP(A);
CALL LIMITS($);
CALL SETADJ$0$RPTR($);
ADFPTR=TYPE$L$OCC$;
SBTR$=SBTR$+2$;
TYPE$L$OCC$=BASE;
END ENTSP$STR$STYP;

ENTPS$PRM$STYP: PROC PUB;
APTP$AR$ = PARAM$UM$LOC $ 1$;
ADRPTR = SBT$L$;
SBTR$ = SBT$L$ + 2$*PARAM$UM$ $ 7$;
BASE = LAST$S$PT$SID$;
DO WHILE PARAM$UM$ <> $0$;
CALL SST$AD$RPTR($);
TEMP$YTE = BYTE$PTR(0);

END
APTRADDR = SETPL;
ETEPTR(7) = TEMPLPE;
SPTRL = SPTRL + 1;
CALL SET$PADDPR(7);
TEMPLPE = AT$PADDR;
APTRADDR = SP$TRPL;
ACCEPTR = TEMPL$ADDPR;
SPTRL = SPTRL - 4;
CALL SET$PADDPR(7);
BASE = ACCEPTR;
PARAM$NUM = PARAM$NUM - 1;
END;
AP$PADDR = PARAM$NUMLOC;
SPTRL = SPTRL + 3*(BYTEPTR(0)+1);
END}

PAR$BYTES: PROC(LCT);
CALL LOC BYTE; IF LOC=39 THEN
CALL GENERATE(J2H); ELSE
IF LOC=133 THEN
CALL GENERATE(J6H);
ELSE
CALL GENERATE(J1E);
END PAR$BYTES;

RUL$IN$P$PAR$HEADER: PROC PUT;
APTRADDR = PARAM$NUMLOC(SP);
BASE = AP$TRADDR;
IF BYTEPTR(0) = 13H THEN
DO; /* CHECK FOR INTEGER OR REAL INPUT */
IF NOT(((SEL((BYTEPTR(0) AND FORM$MASK),3) OR
VAR$ENTRY)=
(FORM$FIELD(SP) AND 7FH))
OR ((ROL((BYTEPTR(2) AND 7F),1) OR VAR$ENTRY=
(FORM$FIELD(SP) AND 7FH))) THEN
CALL ERROR("IP");
ELSE
DO;
CALL GEN$ADMR(PARM$PRT$ADDR(SP));
call PARA$BYTES(BYTEPTR(0));
END;
END;

132
ELSE DO;
  IF BYTEPTR(0) = &= THEN
    DO:
      IF SHR(FMS$FIELD(SP), 3) = 23H THEN /* CAN'T RF */
        CALL ERROR("IP");
      ELSE
        DO;
          CALL GEN$ADDR(PARM, PRT$ADDR(SP));
          CALL PARM$BYTES(BYTEPTR(0));
        END;
        END;
    END;
  END;
END:
END;
PARM$NUMLOC(SP+2) = PARM$NUMLOC(SP) + 1;
IF SHR(FMS$FIELD(SP), 7) THEN CALL GENERATE(LCDI);
END BUILT$INS$PARM;
/*****************************/
* ASSIGN$PARAMETERS - THIS PROCEDURE ENSURES */
* A PROPER MATCH UP BETWEEN THE SUBROUTINE'S */
* FORMAL PARAMETERS AND THE CALLING ACTUAL */
* PARAMETERS. */
*******************************************************************************
ASSIGN$PARAMETERS: PROC PUR;
  IF SIGN$FLAG THEN
    DO:
      IF FMS$FIELD(MP-2) = BUILT$INSFUNC THEN
        CALL BUILT$INS$PARM;
      END;
    ELSE IF FMS$FIELD(MP-2) = BUILT$INSFUNC THEN
      CALL BUILT$INS$PARM;
    ELSE DO;
      APTRADDR = PARM$NUMLOC(SP);
      BASE = APTRADDR;
      IF SHR(BYTEPTR(0), 7) THEN
        DO:
          IF (BYTEPTR(0) AND 7FH) = FMS$FIELD(SP) THEN
            /* THIS IS A VARIABLE PARAMETER */
            CALL GEN$ADDR(PARM, PRT$ADDR(SP));
          ELSE CALL ERROR("IP");
        END;
      ELSE DO; /* THIS IS A VALUE PARAMETER */
        IF (BYTEPTR(0) = FMS$FIELD(SP))
          OR (BYTEPTR(0) = (FMS$FIELD(SP) AND 7FH)) THEN
        DO;
          /* */
        END;
      END;
    END;
  ELSE /* */
CALL GENADDR(PARM, PRTADDR(SP));
CALL PARMBYTES(BYTEPR(2));
END;
ELSE CALL ERROR('IF');
END;
PARMNUMLOC(SP+2) = PARMNUMLOC(SP) - 3;
PARMS = TRUE;
END:
END ASSIGNPARMS;

/* --------------------------------------------------------*/
/* LOOKUP$IDENTIFIER - THIS PROCEDURE IS CALLED */
/* WITH 'SYMHASH' AND PRINTNAME SET. IT WILL */
/* RETURN TRUE IF THE IDENTIFIER CAN BE FOUND */
/* --------------------------------------------------------*/
LOOKUP$IDENTIFIER: PROC BYTE PUS;
BASE=BASEPTE(SYMHASH);
DO WHILE (BASE <> 0) AND (BASE > 3) PRINT(SYM$HASHNUM);
IF ORK$PRTNAME(S) THEN
DO;
LOOKUPSALDR=BASE;
RETURN TRUE;
END;
ELSE DO;
CALL SETADDRTR(S);
BASE=ADD$PTETR;
END;
END;
RETURN FALSE;
END LOOKUP$IDENTIFIER;

/* --------------------------------------------------------*/
/* LOOKUP$PRINTNAME$ONLY - THIS PROCEDURE SETS */
/* THE 'SYMHASH' AND CALLS LOOKUP$IDENTIFIER*/
/* LOOKUP$NAME IF THE ENTRY IS IN THE SYM$HASH */
/* TABLE. THE ADDRESS OF THE PRINTNAME IS */
/* PASSED AS A PARAMETER. IF THE ENTRY IS */
/* FOUND, 'TRUE' IS RETURNED. */
/* --------------------------------------------------------*/
LOOKUP$NAME: PROC(A) BYTE PUS;
I=1 A ADDR; /* ADDR OF PRINT-NAME */
I=1 BYTE; /* BASED A)(1) BYTE; */
HASHCODE=0;
TO I=1 TO N(C);
HASHCODE=(HASHCODE+N(D)) AND HASHPASH;
END;
SYMHASH=HASHCOF;
PRINTNAME=A;
RETURN LOOKUP$IDENTIFIER;
END LOOKUP$NAME;

/* --------------------------------------------------------*/
/* STORE$CONSTANT IDENTIFIER - THIS ROUTINE IS */
/* CALLED WITH PRINTNAME SET TO LOAD AN */
/* IDENTIFIER IN THE 'CONSTANT' 'ALU$' VARIABLE */
/* --------------------------------------------------------*/
STORE$CONST: PROC PUS;

174
CALL NOUT(PRINTNAME, CONSTVALUE(CONSTINIT), ...);  
CONSTINIT=CONSTINIT+(Y(i)-1);  
CONSPNSTATE(CONSPNSPTR)=SYNCH;  
CONSPNSIZE(CONSPNSPTR)=N(i)+1;  
CONSPNSPTR=CONSPNSPTR-1;  
END STORESCONST;  
END SYMBOL;
SYNT1.SRC

$PAGEWIDTH(62) TITLE('SYNT1 - PRODUCTION PROCEDURES')
SYNT1; INC;

DECLARE LIT LITERALLY 'LITERALLY',
DCL LIT 'DECLARE',
EXT LIT 'EXTERNAL',
POS LIT '0',
NEG LIT '1',
PROC LIT 'PROCEDURE',
TRUE LIT '1',
ADD LIT 'ADD',
FALSE LIT '0',
STATELIT LIT 'ADDRESS',
BUILTINS LIT 'ADHOC'; DCL
PSTACKSIZE LIT '48', /* SIZE OF PARSER STACK */
HASHTABLESIZE LIT '128', /* SIZE OF HASHTABLE */
POSTSIZE LIT '8', /* BYTES USED FOR PCD VALUES */
MAXNEST LIT '4', /* MAX LEVEL OF NESTS FOR TYPES */
MAXAPPDIM LIT '6', /* MAX ARRAY DIMENSIONS */
FORMMASK LIT '8', /* USED TO DETERMINE FORM TYPE */
/* FOR ENTERS */
LAPLENTRY LIT '2',
CONSTENTRY LIT '1',
TYPEENTRY LIT '2',
APPENTRY LIT '3',
FUCNENTRY LIT '5',
TYPEDCLUE LIT '7',

/* NUMBR TYPES */
INTTYPE LIT '0',
INTERROTYPE LIT '1',
CHARTYPE LIT '2',
UNSIGNEDTYPCH LIT '3',
SIGNPOSTYPCH LIT '4',
POSTYPELIT '5',
REALTYPE LIT '6',
COMPLEXTYPE LIT '7',
STRINGTYPE LIT '8'; $FILE
/* MANY OF THE FOLLOWING VARIABLES CAN BE REPLACED BY */

MAKING USE OF THE PARALLEL PARSER STACKS */
DCL
ARYSTIM$LOWVAL(25) ADDR EXT,
ARYSTIM$IVAL(25) ADDR EXT,
DISPSTIM(25) ADDR EXT,
CONSTSTIM ADDR EXT,
CONSTSTRIM ADDR EXT,
TYPESTRIM ADDR EXT,
STARTPOS ADDR EXT, /* ADDR OF PTR TO TOP OF FLOS */
MAXBASED STARTDUS ADDR,
TYPP$LOGT ADDR EXT.

136
VAR$#P$ BYTE EXT,
*LOCAL$IC$87 BYTE EXT,
APR$&$T$ (MAX$ARRAY$) ADDR EXT,
VAR$&$P$ (16) ADDR EXT,
VAR$BASE$ (12) ADDR EXT,
ALLOC$#T$ ADDR EXT,
CHANNEL$ADDR ADDR EXT,
CONST$INDEX BYTE EXT,
LOOKUP$ADDR ADDR EXT,
CONST$VALUE (16) BYTE EXT,
CONST$SIZE$BASE (4) BYTE EXT,
CONST$SIZE$P$ BYTE EXT,
CONST$SIZE$ SIZE (4) BYTE EXT,
INTER$SIZE$ ADDR EXT,
SUB$S$P$ (2) ADDR EXT,
SUPER$TYPE(2) BYTE EXT,
SUPER$P$ BYTE EXT,
SUPER$P$ ADDR (1) ADDR EXT,
SUPER$P$ ADDR EXT,
APR$P$ STRP BYTE EXT,
APR$P$ PTR BYTE EXT,
APR$DIM$P$P$ BYTE EXT,
PTR$PTR$ BYTE EXT,
RPC$VAR$S$PTR (MAX$WEST) BYTE EXT,
REC$S$ BYTE,
VARIANT$P$ (MAX$WEST) BYTE EXT,
RPC$ARRAY$ (MAX$ARRAY$) BYTE EXT,
APR$DIM$ (26) ADDR EXT,
CONST$SIZE$ ADDR EXT,
APR$DIM$ ADDR$PTR BYTE EXT;

DCL REGNUM (REGSIZE $4) BYTE EXT,
SCOPE (10) ADDR EXT,
SCOPE$NUM BYTE EXT,
TEMPBYTE BYTE EXT,
TEMPBYTE1 BYTE EXT,
TEMP$P$ ADDR EXT,
TEMPADDR1 ADDR EXT;

DCL

/** COUNTERS **/
CODE$SIZE ADDR EXT, /* COUNTS NUMBER OF LABELS */
ERROR$COUNT ADDR EXT, /* COUNTS NUMBER OF ERRORS */
ALLOC$#R$ ADDR EXT, /* COUNTS PCT ERRORS */
/** FLAGS USED DURING CODE GENERATION */
WRITE$STM$ BYTE EXT, /* IN WRITE STATEMENT */
REAL$STM$ BYTE EXT, /* IN REAL STATEMENT */
NEW$STM$ BYTE EXT, /* GETS NEW RECORD */
DISPOSE$STM$ BYTE EXT, /* DISPOSES OF RECORD */
ALLOCATE$STM$ BYTE EXT, /* PCT LOCATION ASSIGNED */
VAR$P$VAR$ BYTE EXT, /* PARAMETER IS VARIABLE */

READ$PARm$ BYTE EXT, /* READING ACTUAL PARAMETERS */

137
PRESENT BYTE EXT, /* IDENTIFIER IS IN SYMBOOL TABLE */

/* GLOBAL VARIABLES USED BY THE SCANNER */
TOKEN BYTE EXT, /* TYPE OF TOKEN JUST SCANNED */
/* GLOBAL VARIABLES USED IN SYMBOL TABLE */

OPERATIONS */
BASE ADDR EXT, /* BASE LOCATION OF ENTRY */
HASHTABLE(hashtablesize) ADDR EXT, /* HASHTABLE ARRAY */

TABLE */
SETTL TOP ADDR EXT, /* HIGHEST LOCATION OF SYMBOL */

SETPL ADDR EXT, /* CURRENT TOP OF SYMBOL TABLE */
/* UTILITY VARIABLE TO ACCESS */

SETPL */
ADDRTTP BASED APTRADDR ADDR, /* CURRENT 2 BYTES */
POINTER AT */
BYTE(J) RETURNED FROM APTRADDR(1) BYTE, /* CURRENT BYTE */
POINTER AT */
PRINTNAME ADDR EXT, /* SET PRIOR TO LOOKUP OR ENTER */

SYSPAGE BYTE EXT, /* BASE VALUE OF AN IDENTIFIER */
LAST$SETB$LOAD ADDR EXT, /* HOLD PREVIOUS PAST */

LOCATION */
PARAMNUM LOC ADDR EXT, /* STORES POINTER TO PARAM */

LISTING */
SETTIME Scope ADDR EXT; /* PAST OF LAST ENTRY IN */

PREVIOUS $LOOK*/

DCL PRT$IN$STRL(12) BYTE EXT;

/******* PARSE VARIABLES ********/

DCL PARAMNUM(PSTACKSIZE) BYTE EXT, /* MAINTAINS NUMBER OF */
PARAMETERS ASSOCIATED WITH A SUBROUTINE */
LABELSTACK(PSTACKSIZE) ADDR EXT, /* TRACKS STATEMENT */

LABELS */
PARAMNUM LOC(PSTACKSIZE) ADDR EXT, /* MAINTAINS THE */
LOCATION IN SYMBOL TBL WHERE PARAMETER ING STORED */
BASE$LOC(PSTACKSIZE) ADDR EXT, /* STORES THE SYMBOL */
TABLE ADDRESS OF THE PERTINATE ENTRY */
FORM$FIELD(PSTACKSIZE) BYTE EXT, /* STORES THE FORM */
FIELD OF SCANNED IDENTIFIERS */
TYPE$STACK(PSTACKSIZE) BYTE EXT, /* HOLDS A VARIABLE'S */

TYPE */
EXPRESSION$STK(PSTACKSIZE) BYTE EXT, /* CONTAINS THE */
TYPES OF THE EXPRESSION COMPONENTS */
PPT$ADDR(PSTACKSIZE) ADDR EXT, /* STORES AN */
IDENTIFIER'S PPT LOCATION */
PARAMNUM BYTE EXT,
(SP, MP, MPP1) BYTE EXT;

$EJECT
/* MEMONICS FOR PASCAL-SY MACHINE */
LIT '4', EWP LIT '1', LEO LIT '2', LIT LIT '3',
LIT LIT '4', PEO LIT '5', LIV LIT '6', LIT LIT '7',
LIT '8', CVF LIT '9', CNV LIT '10', ALL LIT '11',
LIT '12', ALE LIT '13', ALI LIT '14', SLE LIT '15',
SPE LIT '16', SUM LIT '17', LLI LIT '18', DEU LIT '19',
LIV LIT '20', CIV LIT '21', FET LIT '22', NFS LIT '23',
LFC LIT '24', FCI LIT '25', LSS LIT '26', CCI LIT '27',
LIT '28', LCP LIT '29', TNC LIT '30', TNC LIT '31',
LIT '32', LSS B LIT '33', CR75 LIT '34', AMS LIT '35',
NFC S LIT '36', LEC S LIT '37', SCFS LIT '38', LSS S LIT '39',
SCS LIT '40', RES T LIT '41', NFO S LIT '42', INCL,
LIT '43',
INCL LIT '44', NFG LIT '45',
NFG LIT '46', COM LIT '47', COM LIT '48', NOT X LIT '49',
ANY LIT '50', FOR LIT '51', STC LIT '52', STC LIT '53',
SFC LIT '54', SDR LIT '55', STD LIT '56', STD LIT '57',
UNION LIT '58', STUF LIT '59', ISA LIT '60', CNV LIT '61',
BR LIT '62', LSC LIT '63', CNV LIT '64', WSET LIT '65',
XCR LIT '66', PAM LIT '67', PAM7 LIT '68', PAAM LIT '69',
INC LIT '70', DFC LIT '71', DFL LIT '72', DFL LIT '73',
SFL LIT '74', LPSI LIT '75', KAS LIT '76', LCL LIT '77',
LDR LIT '78', LODI LIT '79', CRE LIT '80', PEO LIT '81',
PFS LIT '82', VTR LIT '83', KFL LIT '84', KFLS LIT '85',
XMP LIT '86', AES LIT '87', SCR LIT '88', SIN LIT '89',
COS LIT '90', APS T LIT '91', EXP LIT '92', LN LIT '93',
SCR LIT '94', ORP LIT '95', FOL LIT '96', FXB LIT '97',
TRUNC LIT '98', ROUND LIT '99', OFT LIT '100', CER LIT '101',
SFC L LIT '102', PRED LIT '103', SFX LIT '104', PUT LIT '105',
GET LIT '106', RESET LIT '107', RER T LIT '108', PAGE LIT '109',
NEW LIT '110', TISPZ LIT '111', FWE LIT '112', XTRE LIT '113',
REV LIT '114',
$EJECT ERROR=PROC((EBP CODE)) EXTERNAL;
DCL FORCODE ADDS;
END PROP;

LOOKUPONLY=PROC (A) BYTE EXTERNAL;
DCL A BYTE;
END LOOKUPONLY;

MOVE=PROC (SOURCE, DESTIN, L) EXTERNAL;
DCL (SOURCE, DESTIN) ADDR,
L BYTE;
END MOVE;

SFT ADDR PTR=PROC (OFFSET) EXTERNAL;
DCL OFFSET BYTE;
END SETADDR PTR;

MON3=PROC EXTERNAL;
END MCNZ;
LIMITS:PROC(COUNT) EXTERNAL;
 DCL COUNT BYTE;
END LIMITS;
ENTRY$CLXX$YP: PROC (A) EXTERNAL;
 DCL A BYTE;
END ENTRY$CLXX$YP;
SET$PAST$PN:PROC(OFFSET) EXTERNAL;
 DCL OFFSET BYTE;
END SET$PAST$PN;
LOOKUP$PNAME:PROC(A) BYTE EXTERNAL;
 DCL A ADDR;
END LOOKUP$PNAME;
GENERATE:PROC OBJCODE) EXTERNAL;
 DCL OBJCODE BYTE;
END GENERATE;
GEN$ADDR:PROC(A,B) EXTERNAL;
 DCL A BYTE, B ADDR;
END GEN$ADDR;
ASSIGN$PARGS:PROC EXTERNAL;
END ASSIGN$PARGS;
ENTRY$VAR$ID:PROC (A,B,ID$ENTRY) EXTERNAL;
 DCL (A,B,ID$ENTRY) BYTE;
END ENTRY$VAR$ID;
ENTRY$LABEL:PROC EXTERNAL;
END ENTRY$LABEL;
ENTRY$PRM$STYP:PROC EXTERNAL;
END ENTRY$PRM$STYP;
PRINT$CHAR:PROC (CHAR) EXTERNAL;
 DCL CHAR BYTE;
END PRINT$CHAR;
CRLF:PROC EXTERNAL;
END CRLF;
PRINT$ERROR:PROC EXTERNAL;
END PRINT$ERROR;
WRITE$INT$FILE:PROC EXTERNAL;
END WRITE$INT$FILE;
MOV$SET$TEL:PROC EXTERNAL;
END MOV$SET$TEL;
CLOSE$INT$FIL:PRCC EXTERNAL;
END CLOSE$INT$FIL;

PRINT:PRCC(A) EXTERNAL;
DCL A ADDR;
END PRINT;

LOOKUP$IDENT:PROC BYTE EXTERNAL;
END LOOKUP$IDENT;

SELECT

INIT$SYNTH: PROC PUBLIC;
CODESIZE = 3;
SETBLOCP=MAX-2;
VFCTR=0;
CONST$PTR=3;
CONST$INIX=0;
CONST$P$PTR=0;
SUBR$PTR=0;
ARY$ADR$PTR=-1;
ARY$PTR=-1;
VARIANT$PART(D)=FALSE;
ARYQTY(0)=0;
ALLOC$ADDR=0;
END INIT$SYNTH;

/***************************************************************************/
* SUBRANGE$ERROR - THIS PROCEDURE IS CALLED *
* IN THE EVENT OF AN IMPROPER VALUE IN A *
* SUBRANGE. *
*******************************************************************************/
SUBR$ERROR: PROC;
CALL ERROR(‘IS’);
SUBR$TYPTY(SUBR$PTR)=INTEGR$TYPE;
SUBR$VAL(SUBR$PTR)=0000E;
END SUBR$ERROR;

/***************************************************************************/
* OPDSpecialLOW$CHECK - THIS PROCEDURE IS *
* CALLED TO ENSURE THE SECOND SUBRANGE VALUE *
* IS GREATER THAN THE FIRST. *
*******************************************************************************/
OPDSpecialLOW$CHECK: PROC PUBLIC;
IF SUBR$PTR=6 THEN RETURN;
IF SUBR$TYPTY(0)=SUBR$TYPTY(1) THEN
    IF SUBR$VAL(0) < SUBR$VAL(1) THEN RETURN;
    CALL ERROR(‘IS’);
END OPDSpecialLOW$CHECK;
SUBRANGE$CHECK: PROC;
   IF SUB$PTR=0 THEN RETURN;
   IF SUB$TYPE(0) <> SUB$TYPE(1) THEN
      DO;
         CALL SUB$ERROR;
         RETURN;
      END;
   IF SUB$VAL(0) < 32768 AND SUB$VAL(1) > 32767 THEN DO;
      INTEGER$DIFF = SUB$VAL(2) + (-SUB$VAL(1))^1;
      RETURN;
   END;
   IF SUB$VAL(0) > 32767 AND SUB$VAL(1) < 32768 THEN DO;
      CALL SUB$ERROR;
      RETURN;
   END;
   IF SUB$VAL(0) < 32768 THEN /* BOTH POSITIVE */ DO;
      IF (SUB$VAL(0)-(SUB$VAL(1)+1)) < 32768 THEN DO;
         INTEGER$DIFF=SUB$VAL(0)-(SUB$VAL(1)+1);
         RETURN;
      END;
      CALL SUB$ERROR;
      RETUR
   END;
   ELSE /* BOTH NEGATIVE */
      IF ( - SUB$VAL(1)-(-SUB$VAL(2)+1)) < 32768 THEN DO;
         INTEGER$DIFF=(- SUB$VAL(1))-( - SUB$VAL(2))+1;
         RETURN;
      END;
      CALL SUB$ERROR;
      RETURN;
   END;
END SUBRANGE$CHECK;

SUBRANGE$IDENTIFIERS:PROCEDURE — THIS ROUTINE */
/* IS CALLED TO DETERMINE THE OFFSET (NUMBER */
/* OF ENTRIES IN A SUBRANGE ) AND THE TYPE OF */
/* SUBRANGE, GIVEN THAT THE SUBRANGE TYPE IS */
/* A NAMED IDENTIFIER. */
SUB$SPEC: PROC;
  CONST$PTR=CONST$PTR-1;
  CONST$INDEX=CONST$INDEX-CONST$SIZE('CONST$PTR');
  PRINTNAME=CONST$VALUE(CONST$INDEX);
  SYM$HASH=CONST$HASH(CONST$PTR);
  IF NOT LOOKUP$SENTENT THEN CALL SUB$ERROR;
  ELSE DO; /* FOUND CONSTANT IDENTIFIER */
    PAST=LOOKUP$ADDR;
    CALL SETADDRPTR(4); /* POINTS TO FMT(EXT$PTR) */
    SUB$FORM=BYTEPTR(0);
    IF SUB$FORM <> 274 AND (SUB$FORM AND FORM$MASK) <> CON$ENTRY
      THEN CALL SUB$ERROR;
    ELSE DO;
      IF SUB$FORM = 274 THEN
        DO;
          SUB$TYPE(SUB$PTR)=OF$TYPE;
          CALL SETADDRPTR(6);
          SUB$FORM=BYTEPTR(0); /* LENGTH OF P.NAME */
          CALL SETADDRPTR(7-SUB$FORM);
          SUB$VAL(SUB$PTR)=DOUBLE(EXT$PTR(0));
          CALL SETADDRPTR(7-SUB$FORM);
          SUB$TYPE$ADDR(SUB$PTR)=ADDPTR;
          CALL ORT$ISLCW$SCEK;
        END;
      ELSE DO;
        DO WHILE !(SUB$FORM,2) = 2:
          IF SUB$FORM,0 = NOT THEN
            IF SIGN$VAL=POS THEN SIGN$VAL=NEG;
            ELSE SIGN$VAL=POS;
            CALL SETADDRPTR(6);
            SUB$FORM=BYTEPTR(0);
            CALL SETADDRPTR(7-SUB$FORM);
            IF NOT LOOKUP$ONLY(APTR$ADDR) THEN
              DO;
                CALL SUB$ERROR;
                SUB$PTR=SUB$PTR+1;
                RETURN;
              END;
            ELSE DO;
              BASE=LOOKUP$ADDR;
              CALL SETADDRPTR(4);
              SUB$FORM=BYTEPTR(0);
            END;
          END;
          IF (SUB$FORM,3) AND 27 = 2 THEN
            DO;
              CALL SUB$ERROR;
              SUB$PTR=SUB$PTR+1;
              RETURN;
            END;
        END;
        /* HERE WE HAVE EITHER AN INTEGER OR C$5 */
        IF (SUB$FORM,3) AND 27 = 1 THEN
      END;
  END;
/* SUBR$CASE - THIS PROCEDURE IS USED TO DETERMINE THE NUMBER OF ENTRIES IN A SUBRANGE */

SUBR$CASE: PPROC(A);
DCL A BYTE;
SIGNVALU=0;
DO CASE EXPRESS$STA(A);
  /* CASE CON$ILI NUMBER */
  PC; CON$TPTR=CON$TPTR+1;
  IF CON$T$STYPF(CON$TPTR)=REAL$STYPF THEN DO;
    CALL SUB$ERROR;
    CON$T$IN$X=CON$T$IN$X-PC$SIZE;
  END;
ELSE
IC: /* INTEGER TYPE */
    CONSTANT=CONSTANT-1;
    CALL
    VCPTR(CONSTANT),SUBRPR'P$R'P'),&:
    SUBRPR'TYPE(SUBRPR'TP)=CHAPRPR'TYPE;
    CALL SUBRSP$P$R;L$CPK;
END;
SUBRPR'TP=SUBRPR'TP+1; /* NEXT TO FILL */
END;
/* CASE IDENT CONSTANT */
CALL SUBRSP$PROC;
/* CASE SIGNED IDENT CONSTANT */
IC: 
    SIGNED=-1;
    CALL SUBRSP$PROC;
END;
/* CASE CONSTANT STRING */
DO;
    CONSTANT=CONSTANT-1;
    CONSTANT=CONSTANT-CONSTANT$SIZE(CONSTANT$PTR); 
    PRINTNAME=.CONSTANT$VALUE(CONSTANT$INDEX);
    IF CONSTANT$SIZE(CONSTANT$PTR) < 2 THEN
        CALL SUBRSP$ERROR;
    ELSE
        DO;
            CASE=PRINTNAME;
            CALL SETADDRP$R(1);
            IF CASEPTR(0) < #17 OR BYTEPTR(0) < 5 THEN
                CALL SUBRSP$ERROR;
            ELSE
                DO;
                    SUBRSP$PTR=SUBRPR$P$R+1;
                    CALL SUBRSP$PROC;
                END;
            END;
        END;
    END;
SUBRPR$TP=SUBRPR$TP+1;
END;
END: /* OF CASE FXPRESS$STY(1P) */
END SUBRSP$CASE;

******************************************************************************************
/* ENTER$SUBRANGE$PRP$Y - THIS PROCEDURE IS */
/* USE TO ENTER A SUBRANGE TYPE ENTRY INTO */
/* THE SYMBOL TABLE. THIS SYMBOL TABLE ENTRY */
/* HAS NO PRINTNAME ASSOCIATED WITH IT. */
******************************************************************************************
NTD$SUBRPR$: PROC PUBLIC;
    TYPE$S$CT=SUBTPL;
    CALL LIMITS(12);
    V$CPR$R=V$CPR$R-1;
    CALL SUBRSP$CASE(SF);
YP$ERRPROP: PROC;
  ALLOCAT=FALSE;
  CALL YP$ERRPROP( 'IT' );
END TYPE$ERROR;

ALLOC$OFFSET: PROC(A) PUBLIC; /* TYPE$LOCCT */
  IN A ADDR:
  DCL (ALLOC$FORM,3) BYTE;
  BASE=A;
  CALL SETADDRPTR(4); /* POINTS TO FORM OF TYPE */
  ALLOC$FORM = BYTEPTR(0) AND FORM$BASE;
  IF ALLOC$FORM <> TYPE$ENTRY AND ALLOC$FORM <> TYPE$CLE
THEN
  DO;
  CALL TYPE$ERROR;
  ALLOC$STY=1;
  ALLOC$ASIC$TY=0;
END.
RETURN;
END;
END:
/FULL EXISTS EITHER A FASCIC TYPE OF A TYPE DECLARATION */
IF ALLOCSFORM = TYPEENTRY THEN
END: /* FASCIC TYPE */
DO CASE (SRR(BYTPTR(2),3) AND FORMASK);
/* INTEGER */
DC:
ALLOCSCTY=2;
ALOCEASICTYP=INTEGER$TYPE;
END:
/* BCD REAL */
DC:
ALLOCSCTY=3;
ALOCEASICTYP=UNSIGN$EXPON;
END:
/* CHARACTER */
DC:
ALLOCSCTY=1;
ALOCEASICTYP=CHAR$TYPE;
END:
/* BOOLEAN */
DC:
ALLOCSCTY=1;
ALOCEASICTYP=BOOLEANTYPE;
END:
/* TEXT */
DC:
ALLOCSCTY = 2;
ALOCEASICTYP = STRING$TYPE;
END;
END: /* OF CASE */
ALLOCATE=TRUE;
RETURN;
END: /* FULL EXISTS A TYPE DECLARATION */
TIMPPYTE1,ALLOCSFORM=(SRR(BYTPTR(2),3) AND FORMASK);
IF ALLOCSFORM = 0 THEN
DO: /* SCALAR */
ALLOCATE=TRUE;
ALLOCSCTY=DOUBLE(ALLOCSFORM+1);
ALOCEASICTYP=ORD$TYPE; RETURN;

IF ALLOCFORM=1 THEN
   DO; /* SURFACE */
      ALLOCATE=TRUE;
      ALLOCATIC=COMPLEXTYPE;
      E=SER(BYTEPTR(3),E);
      IF E = 1 THEN ALLOCCTY=DUPLE(ALLOCFORM-1);
      ELSE ALLOCCTY=TCUPLE(ALLOCFORM); RETURN;
   END;
IF ALLOCFORM=2 THEN
   DO; /* ARRAY */
      ALLOCATE=TRUE;
      ALLOCATIC=COMPLEXTYPE;
      CALL SETADDRPTR(1);
      ALLOCCTY=ADDPTR; RETURN;
   END;
E=2;
/* ALL OTHER CASES ALLOCATE AN ADDRESS FIELD */
ALLOCCTY=DOUBLE(2);
ALLOCATIC=COMPLEXTYPE;
ALLOCATE=TRUE;
END ALLOCOFFSET;

/*****************************/
/* ALLOC_OFFSET - THIS PROCEDURE IS CALLED */
/* TO DETERMINE THE NUMBER OF BYTES REQUIRED */
/* BY AN ARRAY TO STORE THE ARRAY'S COMPONENTS */
/* TYPE=LOC IS SET PRIOR TO CALLING THIS */
/* ROUTINE, AN ADDRESS VARIABLE CONTAINING THE */
/* TYPE COUNT IS RETURNED. */
/* ALLOC_OFFSET: PROC ALTYP PUBLIC:
DCL A ADDP,3 BYTT;
   A,BASE=TYPELOC;
   CALL SETADDRPTR(4);
   IF (SER(BYTEPTR(3),3) AND FORMASK) = 7 AND
      (BYTEPTR(2) AND FORMASK) = TYPEENTRY
      CALL SETADRPTR(7);
      BASE=ADDRPTR; CALL SETADDRPTR(4);
      TYPELOC=BASE;
      END;
   /* IF WE HAVE EITHER A SCALAR,SURFACE,POCLEAN, OR Clone */
   TYPE */
   E=SER(BYTEPTR(3),3) AND FORMASK;
   IF (BYTEPTR(2) AND FORMASK) = TYPEENTRY THEN
      DO;
      IF E = 0 OR E = 1 THEN
         DO;
            CALL ERROR('IA');
            E=2;
            RETURN DOUBLE(3);
         END;
IF \( p = 0 \) THEN /* CHARACTER SUPPASET */

FOR
\( p = 2; \)
    PECSVARSTYP(PECSNST) = CHARSTYP;
RETURN DOUBLF(\( p \));
END;

/* I/O CLEAN */
PECSVARSTYP(PECSNST) = ROCLEANSTYP;
\( p = 2; \) RETURN DOUBLF(\( p \));

END;

/* COMPLEX TYPE */
IF (( BYTEPTR(\( 0 \)) AND FORMMASK)) \( \neq \) TYPESCALE C =
(( \( p \) \( \neq \) \( 2 \)) AND ( \( p \) \( \neq \) \( 1 \))) \{ THEN
DO;
    CALL ERROR('IA');
    \( p = 2; \) RETURN ECULE(\( p \));
END;

IF \( p = 0 \) THEN
TO; /* SCALAR TYPE */
    PECSVARSTYP(PECSNST) = COMPLEXSTYP;
    CALL SETSCMPSTPN(\( p \));
RETURN DOUBLF(BYTEPTR(\( p \)) + 1);
END:

/* SUPPANCE TYPE */
PECSVARSTYP(PECSNST) = GRSTYP;
CALL SETADDPTR(11);
RETURN ADDPTR;

END ALNEXS$OFFSET;

***************************************************************
" ALLOCATESVARIABLES - THIS PROCEDURE IS "
* CALLED TO ASSIGN PTR LOCATIONS FOR EACH *
* OF THE PROGRAM VARIABLE. "
***************************************************************

ALLOCATESVARS: PROC PUBLIC;
TEMPBYTE = \( \delta \);
CALL ALLOCOFFSET(TYPE$LOCST);
TEMPBYTE = VAR$PTR;
DO VAR$PTR = \( \delta \) TO TEMPBYTE;
    PAST = VAR$BASE(VAR$PTR);
    CALL SETADDPTR(4);
IF ERR(BYTEPTR(0),7) THEN
DO;
    BYTEPTR(\( 0 \)) = (BYTEPTR(\( 2 \)) OR (SFL(ALLOCBASICSTYP,2) OR
VAR$ENTRY);
    A$PTRDER = VAR$BASE1(VAR$PTR);
    ADDPTR = ALLOC$ADDR;
    ALLOC$ADDR = ALLOC$ADDR + 2;
END;
ELSE DO;
    BYTEPTR(\( 0 \)) = (SFL(ALLOCBASICSTYP,2) OR VAR$ENTRY;
IF (BYTEPTR(\( 2 \)) = 2CH) AND (TEMPBYTE1 = 2) THEN
149
/* CASE $PTRPTR - THIS PROCEDURE IS CALLED TO */
/* SET ALL VARIABLES APPROPRIATE TYPE. */

CASE $PTRPTR: PROC(A) PUBLIC;

CALL A BYTE:
  TO CASE 4:
  /* CASE 3 ORD VARIABLE */
  ID:
    PTRPTR = $10E;
    CALL SET$PAST$SPN(Id);
    PAST$LOC(SP) = ADRPTR; /* ALL OF PARENT */
  END:
  /* CASE 1 INTGEB VARIABLE */
  PTRPTR = $09E;
  /* CASE 2 CHAR VARIABLE */
  PTRPTR = $39E;
  /* CASE 3 REAL VARIABLE */
  PTRPTR = $0AE;
  /* CASE 4 COMPLEX VARIABLE */
  ID: /* ARRAY, SUBRANGE, USER DEFINED TYPES */
    TEMPADD = BASE; /* STORE VARIABLE START LOCATION */
    CALL SET$PAST$SPN(Id);
    BASE = ADRPTR;
    CALL SET$ADD$PTR(4);
    IF BYTEPTR($0) = 17 THEN /* ARRAY */
      APTRADDR = APTRADDR + 6;
      TEMPBYTE1 = BYTEPTR(3);
    ELSE IF (BYTEPTR($0) AND PTR) = $FF THEN /* SUBRANGE */
TEMPBYTE1 = BYTEPTR(3). = ;
ELSE IF BYTEPTR(2) = 'FA' THEN
PC; /* USER DEFINED TYPE */
TEMPBYTE1 = 0;
CALL SET$PAST$PN(7);
PAST = ADDPTR;
CALL SET$ADDRPTR(4);
IF BYTEPTR(2) <> '7H' THEN CALL ERCP('ns'); /* THIS IS A SET TYPE */
CALL SET$ADDRPTR(5);
PAST$LOC(SP) = ADDPTR; /* ADDR OF PARENT */
END;
ELSE IF BYTEPTR(0) = '7H' THEN
PC; /* POINTER */
CALL SET$ADDRPTR(6);
PAST$LOC(SP) = ADDPTR; /* ALL OF PARENT */
END;
ELSE IF BYTEPTR(0) = '3H' THEN
PC; /* TRAP */
CALL SET$ADDRPTR(7);
PAST$LOC(SP) = ADDPTR; /* OF PARENT */
END;
END;
/* OF CASE */
BASE = TEMPADDR; /* RESTORE ORIGINAL BASE LOCATION */
/*
END;
/* CASE 5 BOOLEAN VARIABLE */
PC = '6H';
END; /* OF VARIABLE CASE */
END CASE$PTPTR;

.KeyPresses$^ 740

/* SET$VARIABLE$TYPE - THIS PROCEDURE IS CALLED */
/* TO SET THE VARIABLE TYPE, VARIABLE SIGN, AND */
/* ADDRESS OF THE BASIC TYPE GIVEN. THE ADDRESS */
/* VARIABLE 'LOOKUP$ADDR' IS SET PRIOR TO TRAP */
/* CALL. */

SET$VARIABLE$TYPE: PROC PUBLIC;
SET$TYPE$LOC: PPOC(A,E,C);
PCL (A, E, C) BYTE;
CALL SET$PAST$PN(A);
IF (P=34H) OR (P=35H) OR (P=36H) OR (P=11H) THEN
PTADDR(SP) = APTRADD;
ELSE PTTABLE(SP) = ADDPTR;
TYPE$STACK(SP) = (P OR P$LOC(C, 7));
END SET$TYPE$LOC;
BASE = Lookup$ADDR;
CALL SMSgettext(4);
SBSTR(SP) = BYTEPTR(8);
TO CASE (FORM$FIELD(SP) AND FORM$MASK);
;
/* CONSTANT ENTRY */
EC:
  SIGNSVALU = POS;
  DO CASE (SHR(BYTEPTR(2),3) AND 255);
  /* FIND OUT WHAT KIND OF CONSTANT IT IS */
  LO WHILE (SHR(BYTEPTR(2),3) AND 255) = 0;
  IF (SHR(BYTEPTR(2),5) AND 014) = 014 THEN
    IF SIGNSVALU THEN SIGNSVALU = NEG;
    ELSE SIGNSVALU = POS;
    CALL SETADDRPTR(6);
  IF NOT LOOKUP$NAME(APTR1) THEN
    DC:
    CALL FPROP('IC');
    RETURN:
END;
CALL SETADDRPTR(4);
IF (BYTEPTR(2) AND FORM$MASK) <> CON$ENTRY THEN
DO:
  CALL FPROP('IC');
  RETURN;
END;
/* INTEGER OR BOOLEAN CONSTA NT */
IF BASE < TBCX ENTRY THEN /* BOOLEAN */
  CALL SET$TYPES$LOC(9,4,POS);
ELSE /* INTEGER */
  CALL SET$TYPES$LOC(7,6,SIGNSVALU);
/* REAL CONSTANT */
CALL SET$TYPES$LOC(7,6,IGNSVALU);
/* STRING CONSTANT */
CALL SET$TYPES$LOC(7,7F,3);
END; /* OF CASE */
/
/* TYPE ENTRY */
;
/* VARIABLE ENTRY */
DO:
  IF SHR(FORM$FIELD(SP),7) THEN VARPARK = TRUE;
  PTRPTR = (SEP(FORM$FIELD(SP),3) AND FORM$MASK);
  BASE$LOC(SP) = PAST; /* SYMBOL TABLE LOCATION OF VARIABLE */
  CALL CASEPTRPTR(PTRPTR);
  CALL SET$TYPES$LOC(7,PTRPTR,2);
END;
/* PROCEDURE ENTRY */
; /* NO SUCH THING EXISTS IN PASCAL */
/* FUNCTION ENTRY */
DO:
  IF FORM$FIELD(SP) = BUILDS$FUNC THEN /* BUILT IN FUNCTION */

152
CALL SETSPASTSP(0);
IF BYTEPTR(3) <> 15 THEN
  IF BYTEPTR(0) <> 12 THEN
    DC:
    CALL CASEPTRPTR(BYTEPTR(2));
    TYPESSTACK(SP) = PTPTR:
    FND;
    APTRADEC = APTRADEC + 1;
    PARVMUM(SP) = BYTEPTR(0);
    PARVMUMLOC(SP) = APTRADEC + 1;
    FND;
  ELSE DC:
    CALL SETSPASTSP(16);
    CALL CASPEPTRTR(SRF(BYTEPTR(3),3) AND FORMASK);
    CALL SETTYPESLOC(10, PTRPTR, 0);
    CALL SETSPASTSP(7);
    PARVMUM(SP) = BYTEPTR(3);
    CALL SETSPASTSPN(6);
    PARVMUMLOC(SP) = ADDPTR;
    CALL SETSPASTSPN(14);
    APTVARPLOC(SP) = ADDPTR;
    FND;
    IF TOYEN <> 18 THEN READPARMS = TPUP;
    /* OTHERWISE, THIS WILL BE A FUNCTION ASSIGNMENT */
    CASE ENTRY */
    PARVMUMLOC(SP + 2) = PARVMUMLOC(SP);
    ENI:
    /* FILE ENTRY */
    ;
    /* SCALAR ENTRY */
    DC:
    CALL SETTYPESLOC(7, 11H, 3);
    APTRADEC = APTRADEC + 1;
    BASELOC(SP) = ADDPTR:
    FND;
    ENI: /* OF CASE */
    FND SETSVAR$TYPE:

*****************************************************************************
/* LOADSVARI - THIS PROCEDURE GENERATES THE */
/* INTERMEDIATE CODE TO LOAD THE NEXT VARIABLE */
/* ON THE EXECUTION STACK OF THE OBJECT FILE */
*****************************************************************************
LOADSVARI: PROC(PT) PUBLIC;
TCL PT BYTE; /* PT REPRESENTS A STACK POINTER */
FYPSTACK: PROC ;
TCL A BYTE;
TO CASE (TYPESSTACK(PT) AND 0FF):
  A = OPDS$TYPE;
  A = ORD$TYPE;
;
/* A */

/* EXP STACK */

FNL EXP$STACK;

LOAD: PROC(A, B, C);

DCL (A, B, C) BYTE;

/* CHECK IF LOADING A FUNCTION VALUE */

IF (FORM$FIELD(PT) AND 7FF) <> FUNCTION THEN

DC:

CALL GENERATE(A);

CALL GENERATE(B);

IF SFR(TYPE$STACK(PT), 6) THEN /* ACCESSING ARRAY */

    CALL GENERATE(SUB);

ELSE CALL GENERATE(C);

IF A = LDI THEN /* LOAD REST OF BCD NUMBER */

TO PTRPTR = 2 TO (ECDNUM(0)/2);

APTRADDR = APTRADDR - 2;

CALL GENERATE($BYTEPTR(2));

CALL GENERATE($HIGH(AHPTRPTR));

END:

IF SFR(FORM$FIELD(PT), 7) THEN /* VARIABLY PARAMETER */

    CALL GENERATE(LDI);

END:

ELSE CALL Generator(PRO, LABELSTACK(MP));

CALL EXP$STACK;

END LOAD;

IF READSTVT THEN RETURN: /* GOING TO READ THIS VALUE */

IF READPARMS THEN

DO: /* READING A SUBROUTINE'S PARAMETERS */

    IF (TOKEN <> 12) AND (TOKEN <> 6) THEN READPARMS =

FALSE:

/* THIS MEANS THIS PARAMETER IS AN EXPRESSION THAT
MUST BE EVALUATED. AFTER EVALUATION, READPARMS WILL BE SET
TO TRUE. */

    ELSE DO:

        CALL ASSIGNPARMS;

        CALL EXP$STACK;

        RETURN:

    END:

/* IF LOADING A FUNCTION VALUE, GO TO THE CASE STATEMENT */

IF (FORM$FIELD(MP) AND 7FF) <> FUNCTION THEN
FO;
IF ('TYPE$STACK(PT) > 3EH' AND 'TYPE$STACK(PT) < 11F') OR
('TYPE$STACK(PT) AND 42H) = 42H THEN /* IN CASE OF ARRAYS */
CALL GENERATE(LITA); /* GOING TO LOAD A PTR AFT */
ELSE APTRADR = PTRADDR(PT); /* GOING TO LOAD A CONSTANT */
END /*
TO CASE ('TYPE$STACK(PT) AND 0FH'); /* /* COND VARIABLE */
CALL LOAD(LOW(PTRADDR(PT)),HIGH(PTRADDR(PT)),LOI);
/*1*/ /* CPD CONSTANT */
; /*2*/ ; /*3*/ ; /*4*/ /* BOOLEAN CONSTANT */
/* */ CALL LOAD(III, BYTEPTR(0), NOP); /*5*/ /*6*/ /*7*/
INTEGER CONSTANT */
/*8*/ CALL LOAD(LEI, BYTEPTR(C), HIGH(ADEPTR(PT));
/*9*/ IF TYPE$STACK(PT) = 3EH THEN CALL GENERATE(W31);
/* */ CALL LOAD(LIII, BYTEPTR(0), HIGH(ADEPTR(PT));
/*10*/ IF TYPE$STACK(PT) = 3EH THEN CALL GENERATE(W32);
/* */ CALL GENPAT(LEI); /* LENGTH OF STRING */
/* */ TEMPBYTE = BYTEPTR(0);
/* */ DO PTEPTR = 3 TO TEMPPTEP;
/* */ CALL GENERATE(ARBAII + PTEPTR);
END;
/* */ BOOLEAN VARIABLE */
/* */ CALL LOAD(LOW(PTRADDR(PT)), HIGH(PTRADDR(PT)), LOI);
/* */ INTEGER VARIABLE */
/* */ CALL LOAD(LOW(PTRADDR(PT)), HIGH(PTRADDR(PT)), LOF);
/* */ REAL VARIABLE */
/* */ CALL LOAD(LOW(PTRADDR(PT)), HIGH(PTRADDR(PT)), LOF);
/* */ CHARACTER VARIABLE */
/* */ CALL LOAD(LOW(PTRADDR(PT)), HIGH(PTRADDR(PT)), LOF);
END; /* OF CASE */
END LOAD$VAR;

******************************************************************************/
/* ASSIGN$VAR - THIS PROCEDURE GENERATES THE */
/* INTERMEDIATE CODE TO LOAD THE LEFT SIDE OF */
/* AN ASSIGNMENT STATEMENT ON THE EXECUTION */
/* STACK AND STORES A RESULT AT THAT LOCATION. */
*******************************************************************************/
ASSIGN$VAR: PROC(LE, STORAGE) PUBLIC;
DCL LE BYTE; /* LE IS THE LEFT SIDE OF ASSN STATE */
DCL (A. B. storage) BYTE; /* STORAGE INDICATES */
/* */ TO DELETE OR LEAVE THE CURRENT VALUE AT THE TOP OF THE STACK */
/* */ IF ('TYPE$STACK(LE) AND 42H) = 42H THEN
DO;
TYPE$STACK(LS) = (TYPE$STACK('LS')) AND (TYPE$STACK('IC'))
CALL GENERATE('IC');
END;
ELSE CALL GENERATE('MATRIX(LS)');
IF SFR(FIELD('LS'),7) THEN /* CHECK FOR VAR PARAMETER */
   CALL GENERATE('LORI');
END CASE
CASE 0
   CALL ERROR('AT');
ELSE A = 2;
/* CASE 1 - INTEGER TYPE */
IF TYPE$STACK('IS') = 0 THEN
   A = 1;
ELSE DO;
   IF TYPE$STACK('IS') = 0 THEN
      CALL GENERATE('CNAI');
      A = 3;
   END;
   ELSE CALL ERROR('AT');
END;
/* CASE 2 - CHAR TYPE */
IF TYPE$STACK('IS') = 0 THEN
   A = 2;
ELSE CALL ERROR('AT');
/* CASE 3 - REAL TYPE */
IF TYPE$STACK('IS') = 0 THEN
   A = 0;
ELSE CALL ERROR('AT');
/* CASE 4 - STRING TYPE */
A = 2;
/* CASE 5 - BOOLEAN TYPE */
IF TYPE$STACK('IS') = 0 THEN
   A = 2;
ELSE CALL ERROR('AT');
END; /* OF CASE */
IF STORE$TYPE THEN A = 1 + 3;
DO CASE 4:
   B = STD;
   S = STL;
   F = STF;
   P = STOP;
   B = STO;
   A = STG;
END; /* OF CASE */
CALL GENERATE('E');
END ASSIGN$VARI;
/* THIS PROCEDURE CHECKS THE TOP TWO */
/* VARIABLES ON THE EXECUTION STACK */
/* FOR PROPER TYPE. */

BYTE PUBLIC;
    IF (EXPRESSION$STK(SP) = EXPRESSION$STK(MP)) AND
    EXPRESSION$STK(SP) <> @H
    THEN RETURN TRUE;
    IF EXPRESSION$STK(SP) = 1H THEN
        DO:
            IF EXPRESSION$STK(MP) = 3P THEN
                DO:
                    CALL GENERATE(CVVI); /* CONVERT INT TO ECI */
                    EXPRESSION$STK(SP) = @H;
                    RETURN TRUE;
                END;
                ELSE RETURN FALSE;
            END:
            IF EXPRESSION$STK(SP) = 3H THEN
                DO:
                    CALL GENERATE(CN21); /* CONVERT SECOND INT TO ECI */
                    EXPRESSION$STK(MP) = 3F;
                    RETURN TRUE;
                END;
                ELSE RETURN FALSE;
            END:
            IF EXPRESSION$STK(SP) = 2H THEN
                DO:
                    IF EXPRESSION$STK(MP) <> @H THEN
                        RETURN FALSE;
                    ELSE DO:
                        IF BOUND$LOC(SP)=BOUND$LOC(MP) THEN
                            RETURN TRUE;
                        END:
                    END;
                END:
                RETURN FALSE;
            END;
            RETURN FALSE;
        END CHK$EXPR$TYPE;

COPY$STACKS:
PROC(A, B) PUBLIC;
    DCL (A, B) BYTE;
    TYP$STACK(A) = TYP$STACK(B);
    PTR$ADD(A) = PTR$ADD(B);
    EXPRESSION$STK(A) = EXPRESSION$STK(B);
    FORM$FIELD(A) = FORM$FIELD(B);
    BASE$LOC(A) = BASE$LOC(B);
PROC;
APTRADR = PARAMUMLOG(MP) - 2;
IF (BYTEPTR(2) = 177) OR (BYTEPTR(2) = 010) GOTO
CALL COPYSTACKS(MP, SP-1);
ELSE EXPRESSTOP(MP) = BYTEPTR(2);
/* GENERATE THE NUMERIC CODE FOR THE BUILT-IN FUNCTION */
*/
APTRADR = APTRADR - 1;
DO CASE BYTEPTR(2);
CALL GENERATE(ABS);
CALL GENERATE(SCR);
CALL GENERATE(SIN);
CALL GENERATE(COS);
CALL GENERATE(ARCTN);
CALL GENERATE(EXP);
CALL GENERATE(IN);
CALL GENERATE(SORT);
CALL GENERATE(CDD);
CALL GENERATE(FOLN);
CALL GENERATE(ESP);
CALL GENERATE(FUNS);
CALL GENERATE(FOUND);
CALL GENERATE(CDI);
CALL GENERATE(CHR);
CALL GENERATE(SUCI);
CALL GENERATE(PRED);
END; /* OF CASE */
END GENERATE;

PROC(NUMP) PUBLIC;
ECL NUM BYTE;
CALL GENERATE(ATTS);
CALL GENERATE(NUMP);
END WRITESTRING;

PROC(NUMP) PUBLIC;
ECL NUM BYTE;
CALL GENERATE(ATTS);
CALL GENERATE(NUMP);
END WRITESTRING;

PROC(NUMP) PUBLIC;
ECL NUM BYTE;
CALL GENERATE(ATTS);
CALL GENERATE(NUMP);
END WRITESTRING;

PROC(NUMP) PUBLIC;
ECL NUM BYTE;
CALL GENERATE(ATTS);
CALL GENERATE(NUMP);
END WRITESTRING;

PROC(NUMP) PUBLIC;
ECL NUM BYTE;
CALL GENERATE(ATTS);
CALL GENERATE(NUMP);
END WRITESTRING;
PROC(CHAR PUBLIC)
DCL NUM, WP; /* NUMBER OF WRITE PARMS */
IF NOT READPARS THEN
   TO CASE EXPRESS$STK(WP);
   /* ORD TYPE */
   CALL GENERATE(WP);
   /* INTEGER TYPE */
   CALL GENERATE(WP);
   /* CHAR TYPE */
   CALL GENERATE(WP);
   /* REAL TYPE */
   CALL GENERATE(WP);
   /* STRING TYPE */
   DC:
      CALL WRITE$STRING(NUM);
      RETURN;
   ENS:
   /* BOOLEAN TYPE */
   CALL GENERATE(WP);
   ENS: /* CASE EXPRESS$STK(WP) */
   CALL GENERATE(NUM);
   ENZ WRITE$VAR;

/***/
/** PPAD$VAR - THIS PROCEDURE CENTRALES */
/** THE INTERMEDIATE CODE TO PPA$ VAR A VARIABLE */
/** FROM THE CONSOLE. */
/***/
/***/
PROC PUPIC;
   IF (TYPE$STACK(SP) < 9) OR (TYPE$STACK(SP) > 9) THEN
      CALL ERROR('IP');
   ELSE DO CASE (TYPE$STACK(SP) - 9) = 0; /* CASE INTEGER */
      CALL GENERATE(REVI); CALL GEN$ATLITAPRATLR(SP)); CALL
      GENERATE(STD); ENS: /* CASE FLOAT */
      CALL GENERATE(PDVP); CALL GEN$ADDR(LITA, PRATADDR(SP)); CALL
      GENERATE(STAP); ENS: /* CASE ASCII */
      CALL GENERATE(PDVP); CALL GEN$ADDR(LITA, PRATADDR(SP)); CALL
      GENERATE(STAP); ENS: /* CASE BYTE */
      ENS: /* CASE (TYPE$STACK(SP) - 9) */
   ENS: /* CASE (TYPE$STACK(SP) - 9) */
   ENS: /* CASE (TYPE$STACK(SP) - 9) */
   ENS: /* CASE (TYPE$STACK(SP) - 9) */
   ENS: /* CASE (TYPE$STACK(SP) - 9) */
   ENS: /* CASE (TYPE$STACK(SP) - 9) */
   ENZ PPAD$VAR;

/*****/
/** P$ISPROCEDURE - THIS PROCEDURE IS CALLED */
/** UPON RECOGNITION OF A BUILT-IN PROCEDURE */
/** STATEMENT. */
/*****/
P$ISPROCEDURE: PROC
FAST = FASTLOC(YR);  
CALL SETSSTATEP;  
IF BYTEPTR(0) < 22 THEN /* FILE HANDLING PROTOCOL */  
   DO CASE (BYTEPTR(0) - 17);  
     CALL GENERATE(PC);  
     CALL GENERATE(GET);  
     CALL GENERATE(RESET);  
     CALL GENERATE(PAGE);  
     CALL GENERATE(NEL);  
   END;  
   /* OF CASE (BYTEPTR(0) - 17) */  
   ELSE DO CASE (BYTEPTR(0) - 22); /* VARIABLE NUMBER OF PARAMETERS */  
     NEWSTMT = FALSE;  
     DISPCONSTM = FALSE;  
     READSTMT = FALSE;  
     REALSTMT = FALSE;  
     WRITESTMT = FALSE;  
     DO;  
       WRITESTMT = FALSE;  
       CALL GENERATE(DUMP);  
     END;  
   END;  
   /* OF CASE (BYTEPTR(0) - 22) */  
FND /* END PROCEDURE;  

/***                        */  
** BPTAYSLINKS = THIS PROCEDURE REMOVES THE *  
** SYMBOL TABLE LOCATIONS FROM THE HASH TABLE *  
** FOR THOSE IDENTIFIERS THAT WERE LOCAL TO *  
** THE CURRENT SCOPE; AND THE SCOPE POINTER IS *  
** DECREmented BY ONE. */  
**                        **/  
BPTAYSLINKS: PROC;  
   DO WHILE SPECTSCOP > SCOPE(SCOPESNW - 1);  
     FAST = SPECTSCOP;  
     CALL SETADDPTR(4);  
     IF ( ((BYTEPTR(0) AND FORMMASK) = 7) ) THEN  
       DO;  
         CALL SETADDPTR(2);  
         SPECTSCOP, FAST = ADDPTR;  
       END;  
       ELSE DO;  
         CALL SETADDPTR(5);  
         SYMPHASE = BYTEPTR(4);  
         CALL SETADDPTR(6);  
         HASTABLE(SYMPHASE) = ADDPTR;  
         CALL SETADDPTR(2);  
         SPECTSCOP, FAST = ADDPTR;  
       END;  
     END;  
   SPECTSCOP = SCOPE(SCOPESNW - 1);  
FND /* END PROCEDURE;
procedure generates the intermediate code that permits branching.

SCOPEFRANCE: PROC PUBLIC;
IF SCOPENUM > 1 THEN
DO;
  APTRADD = PARAMNUMLOC + 1;
  CALL GENSAIRED(ERL, (APTRADD-1));
  CALL GENSAIRED(LRL, APTRADD-1);
END;
END SCOPEFRANCE;

procedure generates all labels in the symbol table.

LABELSMAKER: PROC PUBLIC;
IF TYPENUM = INTEGER THEN
DO;
  CALL ENTREMAPS(C, SP, LABELSMAP);
  CALL ENTRELABEL;
END;
END LABELSMAKER;

This procedure permits the placement of user defined types in the symbol table.

USFSTYPE: PROC(A) PUBLIC;
ECL A BYTE;
TYPELOC=SETBL;
IF LOCKEDONLY(SP) THEN
  CALL ERROR("ETC");
  CALL ENTRMAPS(0, SP, TYPELOC);
IF NOT PRESENT THEN
DO;
  CALL LIMITS(C);
  APTRADD=SP+1;
  BYTEPC(R3)=A;
  APTRADD=APTRADD+1;
  APTRADD=PARENT'STYPE;
  STTL=SP+3;
END;
END USFSTYPE;
COUNTSPAREBYTES - THIS PROCEDURE IS USED TO DETERMINE THE NUMBER OF BYTES ASSOCIATED WITH THE PARAMETERS OF A SUBROUTINE CALL. THIS INFORMATION IS NEEDED TO ALLOW PARAMETER MAPPING FROM THE EXECUTION STACK INTO THE PROTHEM SP OPERAATION.

COUNTSPAREBYTES: PROC (NUMSPAREBYTES, ATTP)
DCI TEMPVAL AD0R.
   (NUMSPAREBYTES, I) BYTF;
   TEMPVAL=0;
   DO I=1 TO NUMSPAREBYTES;
      CALL SETSPASR"(I);
      ATPTRPT=ADPTRT + ((I-1)*3);
      IF PTRPTR"(I)=OFF THEN ALLCSTY=0;
      ELSE
         IF PTRPTR"(I)=1EE THEN
            ALLCSTY=8;
            ELSE
            ALLCSTY=11;
            TEMPVAL=TEMPVAL + ALLCSTY;
         END;
      RETURN TEMVAL;
   END;
   RETURN COUNTSPAREBYTES;

/*************************************************************************/
"GENSFCNCHERSIZE - THIS PROCEDURE IS USED TO DETERMINE THE NUMBER OF BYTES ALLOCATED IN THE DPT FOR A FUNCTION NAME.
GENSFCNCHERSIZE: PROC CLR;
   CALL SETSPAS"SIZE(I16);
   IF BYTEPTR(0) = 7FF THEN
      RETURN 01H;
   ELSE
      IF BYTEPTR(0) = 1FF THEN
         RETURN 0FF;
      ELSE
         RETURN 01H;
   END;
   RETURN 01H;
   END GENSFCNCHERSIZE;

***************************************************************************/
PROSTCNBYTESIZE - THIS PROCEDURE RETURNS THE NUMBER OF BYTES ALLOCATED IN THE DPT FOR A PROVIDER OR FUNCTION DECLARATION. THIS DATA IS REQUIRED TO ALLOW PARAMETER MAPPING INTO THE DPT BY A SAVE OPERAATION.
PROSTCNBYTESIZE: PROC ADDR;
   CALL SETADPTRT(4);
   IF BYTEPTR(2) = 2FF THEN
SETSAVES.getBlock: PROC PUBLIC
    ILIL BytesCounter ater,
    COUNT RATE;
    BYTEsCOUNTER = J;
    LASTsSFILESIL = SBF1;
    IS SCOPE sNUM > 1 THEN
        ELSE = ParamNumLoc(SP - 5);
        CALL SStPASTSPN(12);
        ALTCPTR = AllocsADER; /* SEP */
        ALLOCsADER = ALLOCsADER - 2;
        CALL SStPASTSPN(7);
        TEMPATER = RTRPTR(2) AND SRFV1;
        CALL CENSADER(LDII, TEMATER);
        CALL GNSADER(lDII, PROPSNsPSTSIL7);
        BYTESCOUNTER=COUNTS PARASyTES (COUNT);
        CALL CENSADER(LDII, ByTESCOUNTER);
        CALL SStPASTSPN(2);
        CALL GNSADER(LDII, ALTCPTR);
        CALL SStPASTSPN(12);
        CALL BYTESADDER(LDII, ALTCPTR);
        CALL GENERATE(SAVP);
    END;
END SETSAVES.getBlock;

/* PPDASnSsRoCK — UPON RECOGNITION OF A */
/* SUBROUTINE'S PADING AND BLOCK, THIS */
/* PROGRAM IS CALLED TO GENERATE RECURS */
/* CODE FOR UNSAVING THE SUBROUTINE'S */
/* PARAMETERS IN THE EVENT OF RECURSIVE CALLS. */

PPDASnSsRoCK: PROC PUBLIC
    ELSE = ParamNumLoc(MP);
    CALL SStPASTSPN(12);
    CALL GNSADER(LDII, ALTCPTR);
    CALL SStPASTSPN(12);
    CALL GNSADER(LDII, ALTCPTR);
    CALL GENERATE(UNSfP);

163
CALL GETWILINES;
BASE = PARAMLOC(MPI);
SCOPENUM = SCOPENUM - 1;
CALL GENERATE(FTH);
CALL GETWILINES(14);
CALL GNSADD(LBC, (ADEPTR + 1));
TEMPADD = P03;
CALL GNSADD(LDII, TEMPADD);
CALL GETWILINES(12);
CALL GNSADD(LITA, ADEPTR);
CALL GENERATE(STI);
END READLY;

/ *
FORWARSSUBROUTINE - IN THE EVENT OF A *
FORWARD DEFINED SUBROUTINE, THE ALLOCATED *
SPACES IN THE RPT FOR THE ROUTINE AND ITS *
ASSOCIATED PARAMETERS ARE CP-ALLOCATED AND *
WILL BE REALLOCATED AT THE POINT OF THE *
SUBROUTINE'S DEFINITION, *
***********************************************************************/

FWRSSUPRT: PROC PUBLIC;
SCOPENUM = SCOPENUM - 1;
APTRRAIPT = PARAMNUMLOC + 3;
ALLOCADD = ADEPTRP;
END FWRSSUPRT;

/ *
COSSPARAMETERS - THIS PROCEDURE IS CALLED *
ONE CALL ALL A SUBROUTINE'S PARAMETERS HAVE *
BEEN RECOGNIZED AND ENTERED IN THE SYMBOL *
TABLE, THE NUMBER OF PARAMETERS AND THEIR *
ASSOCIATED TYPE ARE THEN STORED IN THE *
SYMBOL TABLE. *
***********************************************************************/

COTTSPARMS: PROC PUBLIC;
APTRADD = PARAMNUMLOC;
BYTEPTR($) = PARAMNUM;
CALL ENTRSPRMTYP;
END COTTSPARMS;

/ *
SETSPOTYPE - THIS PROCEDURE IS CALLED TO *
LOAD THE TYPE OF OPERATOR USED IN AN EX- *
PRESSION. *
***********************************************************************/

 SETSPOTYPE: PROC(A) PUBLIC:
 ECL A BYTE;
CALL$AT$PROC: PROC(A) PUBLIC;
  IF A BYTE: /* TRUE OR FALSE */
  P$ADP$AVS = FALSE;
  IF A THEN /* THE SUBROUTINE HAS PARAMETERS */
  DO:
    IF PARM#NUM(MP) <> PARM#NUM(SP-1) THEN
      CALL ERROR("PN");
  END:
  IF SHR($FORM$FIELD(MP),3) THEN
  DO;
    IF $FORM$FIELD(MP) = 0 THEN
      CALL $GEN$BUITLSIN;
      ELSE CALL $P$ IS$PROCEDURE;
  END:
  ELSE DO;
    IF $FORM$FIELD(MP) = $FUN$ENTRY THEN
      CALL LOCAL$VARI(MP);
    ELSE DO;
      CALL $GEN$ADDP(PRO, LABELSTACK(MP));
      CALL GENERATE(DSL);
    END;
  END;
  END CALL$AT$PROC;

GOT$FUNCTION$TYPE: PROC PUBLIC;
  BASE=PAR#NUM$LOC(MP);
  CALL SET$PAST$PN(1E);
  BYTE$TPR(3)=SHL(ALOC$BASIC$TYP,3) OR VAR$ENTRY;
  CALL SET$PAST$PN(1E);
  $ALOC$ADDR = ADDP$PTR;
  $ALOC$ADDR = $ALOC$ADDR + $ALC$QTY;
END GOT$FUNCTION$TYPE;
/*************************************************************************************/
/* ENDPGRAM: - THIS PROCEDURE IS CALLED UPON */
/* RECOGNITION OF THE END OF A PROGRAM. IT */
/* POINTS OUT THE ERROR COUNT, CLOSES THE */
/* INTERMEDIATE FILE, WRITES THE SYMBOL TABLE */
/* FILL, AND INFORMS THE PROGRAMMER OF PROGRAM */
/* COMPIATION. */
/* *************************************************************************************/
ENDPROGRAM: PROC PUBLIC;
    CALL PRINTERROR;
    CALL PRINTCHAR( " ");
    CALL CRLF;
    IF NOT (ERRORCOUNT > 0) THEN
        CALL GENADDR(ALL,ALLOCADDR);
        CALL GENERATE(ENDP);
    END;
    CALL WRITEINTFILE;
    CALL MOVESTRBL;
    CALL CLOSEINTFILE;
    CALL PRINT( (." COMPIATION COMPLETE." ));
    CALL MOCN;
END ENDPROGRAM;

/*******************************************************************************/
/* ARRAYDECLARE: - THIS PROCEDURE DETERMINES */
/* AND STORES SYMBOL TABLE INFO ON ARRAYS. */
/* THIS PROCEDURE FAILS TO MAKE USE OF THE */
/* PARALLEL PARSE STACKS. */
/* *******************************************************************************/
ARRAYDECLARE: PROC PUBLIC;
    DCL I BYTE;
    IF APPYSPTR = -1 THEN APPYSPTR = 0;
    CALL ENTP$CPSTYP(17F);
    APPYSATRSPTR = APPYSPTR + NUMSAPPYSTIM(APPYSPTR);
    APPYSASF = BASE;
    CALL LIMITS((NUMSAPPYSTIM(APPYSPTR)*4)+E);
    CALL SETADDRPTR(5); /* NUMBER OF DIMENSIONS*/
    BYTPTR(0) = NUMSAPPYSTIM(APPYSPTR);
    CALL STADDRPTR(E); /* ADDRESS OF COMPONENT TYPE*/
    ADDRSPTR = TYPE$LOCT;
    BASE = APPYSPTR;
    CALL SETADDRPTR(0); /* TOTAL STORAGE REQUIRED*/
    ADDPPT = APPYSPTR$ALLOCSCTY;
    CALL STADDRPTR(12); /* COMPONENT TYPE*/
    BYTPTR(0) = ALLOCS[type];
/*******************************************************************************
THE FOLLOWING CODE CALCULATES THE OFFSET AND DISPLACEMENT VECTORS FOR EACH ARRAY DECLARATION AS FOLLOWS:
WHERE \( V = \# \text{DIMENSIONS IN THIS ARRAY} \),
\( D(I) = \text{DISPLACEMENT VECTOR FOR IT} \)
\( U(I) = \text{UPPER BOUND OF IT ARRAY} \)
\( L(I) = \text{LOWER BOUND OF IT ARRAY} \)
\( V = \text{OFFSET FOR THIS ARRAY} \)

FOR I = \# DOWNTO 1
IF I = \# THEN D(I) = 0 ELSE
\[ D(I) = (U(I+1) - L(I+1)) / \#(I-1) \]
\[ V = V - (L(I) - D(I)) \]

\( \text{ARRAYOFFSET} = 0; /\text{INIT FOR ZERO-ORIGIN}\ / \)
\( \text{SUBOFFSET} = \# \text{ARRAYDIM(APRYPTR)}; \)
\( \text{DISPSVEC(SUBOFFSET)} = 1; \)
\( \text{ARRAYOFFSET} = \)
(\( \text{ARRAYDIMLOWVAL} (\text{ARRAYDMADRPT} + \text{SUBOFFSET}) \))
\( \text{DISPSVEC(SUBOFFSET)} ; \)
\( \text{SUBOFFSET} = \text{SUBOFFSET} - 1; \)
DO WHILE \( \text{SUBOFFSET} > 2; \)
\[ \text{DISPSVEC(SUBOFFSET)} = \]
(\( \text{ARRAYDIMHIGHVAL} (\text{ARRAYDMADRPT} + \text{SUBOFFSET} - 1) - \text{ARRAYDMALVOL} (\text{ARRAYDMADRPT} + \text{SUBOFFSET} - 1) \))
+1)
\[ \text{DISPSVEC(SUBOFFSET} + 1); \)
\( \text{ARRAYOFFSET} = \)
(\( \text{ARRAYDIMLOWVAL} (\text{ARRAYDMADRPT} + \text{SUBOFFSET}) \))
\( \text{DISPSVEC(SUBOFFSET)} ; \)
\( \text{SUBOFFSET} = \text{SUBOFFSET} - 1; \)
END; /* DO WHILE */
CALL STADERPT(11); /*OFFSET*/
ALDRPT = ARAYSADDRESS * ALLOCSTR;
CALL STADERPT(13); /*ADDRESS OF DIMENSION 1*/
DO I=1 TO NUM$ARRAY$DIM(APRYPTR);
ALDRPT = ARAYSADDRESS + APRYPTR + 1;
APRADD = APRADD + 2; /*DISP VECTOR FOR THIS
DIMENSIONS*/
ALDRPT = DISPSVEC(I) * ALLCSTR;
APRADD = APRADD + 2; /*SET-UP FOR NEXT DIMENSIONS*/
END;
TYPE$LOC=BASE;
STBLR = STBLR + (NUM$ARRAY$DIM(APRYPTR) * 4) + 8;
ARRPYPTR = APRYPTR - 1;
END ARAYSDECLARE;

/\\*
FINESHELPEL - THIS PROCEDURE DETERMINES *
WHAT MNEMONIC SHOULD BE GENERATED FOR ANY*
RELATIONAL OPERATOR. */

167
FINDSRLCP: PROC PUBLIC;
  CALL A TYPE;
  IC CASE (TYPE=STACK(MPP1)-8);
  A = ECLI;
  A = NECI;
  A = LECI;
  A = GECI;
  A = LSSI;
  A = CBTI;
  IF EXPRESSION(STK(SP)) <> ORD$TYPE THEN CALL ERROR('CE');
  ELSE A = XIN;
  END: /* CASE (TYPE=STACK(MPP1)-8) */
  DO CASE EXPRESSION(STK(SP));
  */ ORT TYPE */
  IF (A = LSSI) OR (A = GHTI) THEN CALL ERROR('CF');
  ELSE IF A <> XIN THEN A = A + 19;
  */ INTEGER TYPE */
  ; /* NO OFFSET REQUIRED */
  */ CHAR TYPE */
  ; /* NO OFFSET REQUIRED */
  */ REAL TYPE */
  A = A + 7;
  */ STRING TYPE */
  A = A + 13;
  */ BOOLEAN TYPE */
  ; /* NO OFFSET REQUIRED */
  END: /* OF CASE EXPRESSION(STK(SP)) */
  CALL GENERATE(A);
  EXPRESSION(STK(MP)) = BOOLEAN$TYPE;
  END FINDSRLCP;

END SYNTX1;
SYNTGEN.SFC

SPACEWIDTH(80) TITLE('SYNT - PRODUCTION CASE STATEMENTS')

SYNTGEN: DO;

DECLARE LIT LITERALLY 'LITERALLY', EXT LIT 'EXTERNAL'.

DCL LIT 'DECLARE',
POS LIT '0',
NEG LIT '1',
PROC LIT 'PROCEDURE',
TRUE LIT '1',
'ADDR LIT 'ADDRESS',
FALSE LIT '0',
FOREVER LIT 'WHILE TRUE',
STATE LIT 'ADDRESS',
BUILTSINPROC LIT 'OCH',
CONSSTRTYPE LIT '3',
CONSSNUMTYPE LIT '4',
CONSSIDENTTYPE LIT '1',
CONSSIDENTTYPE LIT '2';
LCL
IDENTSIZE LIT '32', /* MAX IDENTIFIER SIZE - 1 */
VARC_SIZE LIT '128', /* SIZE OF VARP STACK */
STACKSIZE LIT '46', /* SIZE OF PARSE STACKS */
HASTABLESIZE LIT '128', /* SIZE OF HASTABLE */
PERSIST LIT '2', /* TYPES USED FOR PTR VALUES */
MAXNEST LIT '3', /* MAXLEVEL OF NESTS FOR TYPES */
MAXNEST LIT '4', /* MAX NESTED LEVEL FOR ARRAYS */

MAX$ARRYSIM LIT '2', /* MAX ARRY DIMENSIONS */

/** FORM ENTRIES */

LABELENTRY LIT '0',
CONSENTRY LIT '1',
TYPEENTRY LIT '2',
VARSENTRY LIT '3',
PROCENTRY LIT '4',
FUNCENTRY LIT '5',
FILEENTRY LIT '6',
TYPE$DCLE LIT '7',

/** NUMBER TYPES */

ORD$TYPE LIT '0',
INTERTYPE LIT '1',
UNSIGN$EXPO LIT '3',
STRING$TYPE LIT '4',
BOOLEAN$TYPE LIT '5';

/** MANY OF THE FOLLOWING VARIABLES CAN BE REPLACED BY MAKING USE OF THE PARALLEL PARSE STACKS */

DCL NUMS$PMTS(25) ADDR, ARrys$DIM$LOWVAL(25) ADDR
PUBLIC, ARrys$DIM$HIGHVAL(25) ADDR PUBLIC, TEMPS$BASE ADDR
PUBLIC, EXPSCTR BYTE, EXPSCTR BYTE, DISP$VEC(25) ADDR PUBLIC,
APRYS$OFFSET ADDR PUBLIC,
SIGNTYPE BYTE EXT,
VECPTR BYTE EXT,
TYPE$EXT BYTE EXT,
CONST$PTR BYTE EXT,
STARTEOS ADDR /* ALR OF PTR TO TOP OF FCNS */
$ADD ADDR EXT,
$LOC TADDR ADDR EXT,
VAR$PTR BYTE EXT,
VAR$PART$PTR BYTE EXT,
ALOC$EXT BYTE EXT,
APRYSQTY(MAX$ARRAY$LIMIT) ADDR EXT,
VAR$BASE(1C) ADDR EXT,
ALLOCQTY ADDR EXT,
$ORD$NUM BYTE EXT,
PARENT$TYPE ADDR EXT,
CONST$INDEX BYTE EXT,
LOCKUP$ADDR ADDR EXT,
CONST$TR$PTR BYTE EXT,
APRYS$PTR BYTE EXT,
APRYS$P$PTR BYTE EXT,
PTR$PTR BYTE EXT,
TAG$ED(MAX$NEST) BYTE EXT,
VAR$SCAN$IP(MAX$NEST) ADDR EXT,
VAR$SCAN$VAL(MAX$NEST) ADDR EXT,
EC$NEST BYTE EXT,
RECORD$SPTR BYTE EXT,
RECSADDR(1D) ADDR EXT,
RECS$P$ADDR(MAX$NEST) ADDR EXT,
VARIANT$PART(MAX$NEST) BYTE EXT,
VAP$OST$BST(MAX$NEST) ADDR EXT,
VAP$OST$BSF(MAX$NEST) ADDR EXT,
CUR$OST$(MAX$NEST) ADDR EXT,
NUMSARRAY$DIM(MAX$ARRAY$DIM) BYTE EXT,
APRYS$DIM$(25) ADDR EXT,
ARY$DM$ADDR$PTR BYTE EXT,
/* CASE STATE$MENT VARIABLES */
CASE$STK(12) BYTE EXT; /* NUMBER OF SCMTS IN CURRENT CASE */
CASE */
CASE$COUNT BYTE EXT; /* LEVEL OF CASE SCMTS */
/* GLOBAL VARIABLES */
DCL SCOM$NUM$(ECDSIZE) BYTE EXT.
SCOPE(10) ADDR EXT,
SCOPE$NUM BYTE EXT,
TEMPBYTE BYTE EXT,
PRODUCTION BYTE EXT,
PPV$S$ENTRY ADDR EXT;
/* COMPILER TOGGL ES */
DCL LIST$PROD BYTE EXT,
DI$EGU$LN BYTE EXT;
/* COUNTERS */
/* FLAGSC COUNTS NUMBER OF LABELS */
/* USE IF CODE GENERATION */
/* CASE STATEMENT */
/* WRITE STATEMENT */
/* READ STATEMENT */
/* WRITE STATEMENT */
/* GETS NEW RECORD */
/* DISPOSE STATEMENT */
/* DISPOSES OF RECORDED */
/* VARIABLE IS PARM */

/* READPARAMS BYTE EXT, */
/* PARM NUM BYTE EXT, */
/* IDENTIFIER IS IN SYMBOL TABLE */
/* SET WHEN SIGN PRECEDES ID */
/* CONT BYTE EXT, */
/* INDICATES FULL ACCUM -- STILL MORE */
/* HOLD CURRENT TOKEN */
/* GLOBAL VARIABLES USED IN SYMBOL TABLE OPERATIONS */
/* BUILTINS (12) BYTE EXT, */
/* BASE ADDR EXT, */
/* BASE LOC OF ENTRY */
/* TEMPAL ADR EXT, */
/* CURRENT TOP OF SYMBOL TABLE */
/* APTADDR ADDR EXT, */
/* UTILITY VARIABLE TO ACCESS */
/* ADDRPTR */
/* CURRENT 2 BYTES */
/* BYTEPTR BASED APTADDR BYTE */
/* CURRENT BYTE POINTER */

/* PARSER VARIABLES */
DCL
/* MAINTAINS NUMBER OF PARAMETERS ASSOCIATED WITH A SUBROUTINE */
LABELS */
PARAMLOC(PSTACKSIZE) ADDR EXT, */
LOCATION IN SYMBOL TABLE WHERE PARAMETER INPC STORED */
BASELOC(PSTACKSIZE) ADDR EXT, */
STORES THE SYMBOL TABLE ADDRESS OF THE PERTINENT ENTRY */
FORMSFIELD(PSTACKSIZE) BYTE EXT, */
STORES THE FORMS FIELD OF SCANNED IDENTIFIERS */
TYPESTACK(PSTACKSIZE) BYTE EXT, */
HOLDS A VARIABLE'S TYPE */
EXPRESSIONS(PSTACKSIZE) BYTE EXT, */
CONTAINS THE TYPES OF THE EXPRESSION COMPONENTS */
FINDADD(PSTACKSIZE) ADDR EXT, */
STORES AN IDENTIFIER'S PTR LOCATION */
PARAMNUM EXT BIT, */
(SP, MP, MP1) BYTE EXT;
/* MNEMONICS FOR PASCAL-SM MACHINE */

DCL
NOP LIT '0', FND LIT '1', LBL LIT '2', LDI LIT '3',
LIT '4', PRO LIT '5', RDN LIT '6', SAVE LIT '7',
INT '8', CNV LIT '9', CVI LIT '10', ALL LIT '11',
LIT '12', ADI LIT '13', ADDI LIT '14', SUEF LIT '15',
SUE LIT '16', MUL LIT '17', MULI LIT '18', FIVE LIT '19',
DIV LIT '20', MODX LIT '21', FCL LIT '22', NEG LIT '23',
LI H LIT '24', GCE LIT '25', LSI LIT '26', HT LIT '27',
YIV LIT '28', ECL LIT '29', NECE LIT '30', LCE LIT '31',
FCD LIT '32', LSS LIT '33', JRP LIT '34', ECLS LIT '35',
NEOS LIT '36', LEC LIT '37', GFS LIT '38', LSSS LIT '39',
GRTS LIT '40', EFSFT LIT '41', NEST LIT '42', INCI LIT '43',
INC LIT '44', NED LIT '45',
NEG LIT '46', COM LIT '47', COI LIT '48', ACTX LIT '49',
ANDY LIT '50', BI LIT '51', SICE LIT '52', SCI LIT '53',
STC LIT '54', STD LIT '55', STDI LIT '56', STE LIT '57',
UNION LIT '58', UDIF LIT '59', ISEC LIT '60', DM LIT '61',
PRL LIT '62', ELC LIT '63', CN2I LIT '64', WSET LIT '65',
XCHG LIT '66', PAPM LIT '67', PAPM LIT '68', WAPF LIT '69',
INC LIT '70', EGC LIT '71', DEL LIT '72', WRT LIT '73',
SCL LIT '74', LSI LIT '75', KASE LIT '76', LG LIT '77',
LOC LIT '78', LIDI LIT '79', RDX LIT '80', EDI LIT '81',
RDVS LIT '82', WRPT LIT '83', ARI LIT '84', WPTS LIT '85',
TUMP LIT '86', ABS LIT '87', SCR LIT '88', SIN LIT '89',
COS LIT '90', ARCTN LIT '91', EXP LIT '92', LN LIT '93',
SCRT LIT '94', ODD LIT '95', FOUT LIT '96', EXF LIT '97',
TRUNC LIT '98', ROUN LIT '99', ORL LIT '100', OR LIT '101',
SUC LIT '102', PRET LIT '103', SFFX LIT '104', PUT LIT '125',
GET LIT '126', RESET LIT '127', REPT LIT '128', PAGF LIT '129',
NEW LIT '110', ISPZ LIT '111', PFE LIT '112', XDR LIT '113',
REF LIT '114';

SCANNER: PPC EC XT;
END SCANNER;

PRINT$PROD: PPC EC XT;
END PRINT$PROD;

ERROR: PPC (ERROR) EC XT; DCL ERROR LER;
END ERROR;

/* EXTERNAL PROCEDURES FROM SYMBOL.SRC */
GENERATE: PPC (BEGIN) EC XT; DCL OGOLF BYTE;
END GENERATE;

GNS$ADD: PPC (A, B) EC XT; DCL A BYTE; DCL B ALDP;
END GNS$ADD;

SETADRPTR: PPC (OFFSET) EC XT;
DCL OFFSET BYTE;
END SETADRPTR;

SFT$PAST$PN: PPC (OFFSET) EC XT;

172
DCL OFFSET BYTE;
END SETPARSCAN;

CPX$RT$NAME: PROC(A)$ BYTE EXT;
/* A IS OFFSET FROM BASE TO PRINTNAME */
DCL (LEN, A) BYTE;
END CPX$RT$NAME;

LOOKUP$PSII: PROC(A,ID ENTRY) BYTE EXT;
DCL (A, ID ENTRY) BYTE;
END LOOKUP$PSII;

LIMITS: PROC(COUNT) EXT;
DCL COUNT BYTE;
END LIMITS;

ENTERSVAR$II: PROC(A,B,ID ENTRY) EXT;
DCL (A,B,ID ENTRY) BYTE;
END ENTERSVAR$II;

ALTER$PR$LOC: PROC EXT;
DCL (I,P) BYTE;
END ALTER$PR$LOC;

ENTERSQRTN: PROC(A,B,ID ENTRY) EXT;
DCL (A,B,ID ENTRY) BYTE;
END ENTERSQRTN;

LOOKUP$ONLY: PROC(A) BYTE EXT;
DCL A BYTE;
END LOOKUP$ONLY;

CONVRTBCD: PROC(A,B) EXT; /* A=SP/MP/MPP1, B=POS/NFG */
DCL (I,J,DFLAG,EFLAG,SFLAG,A,B,N BASED PRINTNAME) BYTE;
END CONVRTBCD;

CONVRTI: PROC(A,B) ADDRESS EXT;
DCL (I,A,B,N BASED PRINTNAME) BYTE;
DCL NUM ADER;
END CONVRTI;

CONVRT$CONST: PROC(A) EXT; /* A=POS,NFG */ DCL A BYTE;
END CONVRT$CONST;

ENTP$CONS$TRY: PROC EXT;
DCL IXINDX BYTE;
END ENTP$CONS$TRY;

ENTP$STR$TYP: PROC(A) EXT;
DCL A BYTE;
END ENTP$STR$TYP;

STOP$CONST: PROC EXT;
END STORE$CO$UNT;

/* EXTERNAL PROCEDURE DECLARATIONS FROM SY$1.SET */

CRE$HI$L0WSCHX: PROC EXT;
END CRE$HI$L0WSCHX;

ENT$SUSER$TRY: PROC EXT;
END ENT$SUSER$TRY;

ALLOC$OFFSET: PROC(A) EXT; /* TYPE$LOC */
  DCL A ADDR;
  DCL (ALLOC$FORM, 3) BYTE;
END ALLOC$OFFSET;

AL$NX$OFFSET: PROC ADDR EXT;
  DCL A ADDR, 3 BYTE;
END AL$NX$OFFSET;

ALLOC$VARS: PROC EXT;
END *ALLOC$VARS;

CASE$PTPR$PTR: PROC(A) EXT;
  DCL A BYTE;
END CASE$PTPR$PTR;

SET$VAR$TYPE: PROC EXT;
END SET$VAR$TYPE;

LOAD$VAR: PROC(P) EXT;
  DCL PT BYTE; /* PT REPRESENTS A STACK POINTER */
END LOAD$VAR;

ASSIGN$VAR: PROC(LS, STGP$STYP$) EXT;
  DCL LS BYTE; /* LS IS THE LEFT SIDE OF ASSM STMT */
  DCL (A, B, STGP$STYP$) BYTE; /* STGP$STYP$ INDICATES WHETHER TO DELETE OR LEAVE THE CURRENT VALUE AT THE TOP OF THE STACK */
END ASSIGN$VAR;

CK$SEXPR$STYP$: PROC BYTE EXT;
END CK$SEXPR$STYP$;

COPY$STACKS: PROC(A, B) EXT;
  DCL (A, F) BYTE;
END COPY$STACKS;

WRITE$VAR: PROC(NUM$) EXT;
  DCL NUM$ BYTE; /* NUMBER OF WRITE PARAMS */
END WRITE$VAR;

READ$VAR: PROC EXT;
END READ$VAR;
PROCEDURE: PROC EXT;
   END PROCEDURE;

LABELS: PROC EXT;
   END LABELS;

USER$TYPE: PROC(A) EXT;
   DCL A BYTE;
   END USER$TYPE;

SET$SAVE$BLOCK: PROC EXT;
   END SET$SAVE$BLOCK;

HEAD$SAVE$BLK: PROC EXT;
   END HEAD$SAVE$BLK;

FIND$SUERTN: PROC EXT;
   END FIND$SUERTN;

GOT$PARAM$S: PROC EXT;
   END GOT$PARAM$S;

SET$CPS$TYPE: PROC(A) EXT;
   DCL A BYTE;
   END SET$CPS$TYPE;

CALL$AS$PROC: PROC(A) EXT;
   DCL A BYTE; /* TRUE OR FALSE */
   END CALL$AS$PROC;

GOT$FUNC$TYPE: PROC EXT;
   END GOT$FUNC$TYPE;

END$PROGRAM: PROC EXT;
   END END$PROGRAM;

ARRAY$DECL$ATE: PROC EXT;
   END ARRAY$DECL$ATE;

FIND$REL$OP: PROC EXT;
   END FIND$REL$OP;

SELECT SYNTHESIZE: PROC PUBLIC;

IF LIST$PROD THEN
   CALL PRINT$PROD;
END CASE PRODUCTION;

/**** P R O D U C T I O N S ******/

/* CASE 0 NOT USE */
/* 1 <PROGRAM> ::= <PROGRAM HEADING> <BLOCK> */
CALL ENDPGRAM:
/*
2  <PROCEDURE HEADING> <BLOCK> . _
CALL ENDPGRAM:
/*
3  <FUNCTION HEADING> <BLOCK> . _
CALL ENDS PROGRAM:
/*
4  <PROGRAM HEADING> ::= PROGRAM <PROG IDENT> 
/
5  <XFILE IDENT> ; 
DO;
SCOPESNUM = 6;
SCOPEN(SCOPESNUM) = SETBL;
SCOPESNUM = 1;
END;
/*
5  <XFILE IDENT> ::= <FILE IDENT> 
/
6  <FILE IDENT> ::= <PROG IDENT> , <FILE IDENT> 
/
7  <PROG IDENT> ::= <IDENTIFIER> 
/
8  <FILE IDENT> ::= <IDENTIFIER> 
CALL ENTER$VAR$SP(16,SP,FILE$ENTRY);
/
9  <BLOCK> ::= <LDP><CEP><TLK><VDP><P&D$I><STMP> 
/
10  <LDP> ::= 
CALL SCOPESBRANCH;
/*
11  _ LABEL <LABEL STRING> ; 
CALL SCOPESBRANCH;
/*
12  <LABEL STRING> ::= <LABEL> 
CALL LABELSMAKER;
/*
13  _ <LABEL STRING> , <LABEL> 
/
14  <LABEL> ::= <NUMBER> 
IF TYPENUM <> INTEGER$TYPE THEN 
CALL ERROR(\"LS\") ; 
/
15  <CDP> ::= 
/
16  _ CONST <CONST DEF> ; 
/
17  <CONST DEF> ::= <IDENT CONST DEF> 
/
18  _ <CONST DEF> ; <IDENT CONST DEF> 
/
19  <IDENT CONST DEF> ::= <IDENT CONST> = <CONSTANT> 
CALL ENTER$CONST$ENTRY;
/
20  <IDENT CONST> ::= <IDENTIFIER> 
TO; 
IF LOOKUP$ONLY(SP) THEN 
CALL ERROR(\"DC\") ; 
CALL ENTER$VAR$SP(0,SP,CONST$ENTRY); 
END;
/
21  <CONSTANT> ::= <NUMBER> 
TO; 
CALL CONV$PT$CONST(PCS); 
EXPRESS$STK(\$P)=CONST$NUM$TYPE; 
VECPTR=VECPTR+1;
/* 22 */
ENT:  
  <SIGN> <NUMBER>
  DO:
    IF SIGNTYPE=NEG THEN
      CALL CONV$CONSTM Neg;
      ELSE CALL CONV$CONSTM Pos;
      EXPRESS$STK($P)=CONS$IDENT$TYPE;
      VECPTP=VECPTP+1;
      SIGN$FLAG = FALSE;
    END:
  END:
/* 23 */
<CONSTANT IDENT>
  DO:
    EXPRESS$STK($P)=CONS$IDENT$TYPE;
    VECPTP=VECPTP+1;
    CALL STORE$CONST;
    END:
/* 24 */
<SIGN> <CONSTANT IDENT>
  DO:
    IF SIGNTYPE=NEG THEN
      EXPRESS$STK($P)=CONS$IDENT$TYPE;
      ELSE EXPRESS$STK($P)=CONS$IDENT$TYPE;
      VECPTP=VECPTP+1;
      CALL STORE$CONST;
      SIGN$FLAG = FALSE;
    END:
/* 25 */
<String>
  DO:
    EXPRESS$STK($P)=CONS$STRING$TYPE;
    VECPTP=VECPTP+1;
    CALL STORE$CONST;
    END:
/* 26 */
<CONSTANT IDENT> ::= <IDENTIFIER>
/* 27 */
<SIGN> ::= +
  DO:
    SIGN$TYPE = POS;
    SIGN$FLAG = TRUE;
  END:
/* 28 */
<NUMBER> ::= 
  CASE$STMT=FALSE;
/* 29 */
  TYPE <TYPE DEF STRING> ;
  CASE$STMT=FALSE;
/* 30 */
  <TYPE DEF STRING> ::= <TYPE ID>
/* 31 */
  <TYPE DEF STRING> ::= <TYPE ID>
/* 32 */
  <TYPE ID> ::= <TYPE ID$S> = <TYPE$
  DO;
    APTR$ADDR=TYPE$ADDR;
    177
ADTPTR = TYPESLOC;

/* 34  */  <TYPE IDENT> ::= <IDENTIFIER> /* */
DO;
  IF LOCKUPSONLY(SP) THEN
    CALL TPROM('DT');
    PARENT TYPE = SP+TLP;
    CALL ENT$VAR$ENTRY(SP, TYPENAME);
  IF NOT PRESENT THEN
    CALL LIMITS(2);
    TYPESADDR = SP+TLP;
    SP+TLP = SP+TLP + 2;
END:

/* 35  */  <TYPE> ::= <SIMPLE TYPE> /* */
/;/* 36  */  <STRUCTURED TYPE> /* */
/;/* 37  */  <POINTER TYPE> /* */
/;/* 38  */  <SIMPLE TYPE> ::= <TYPE IDENT> /* */
/;/* 39  */  ( <IDENT STRING> ) /* */
/;/* 40  */  <CONSTANT> .. <CONSTANT> /* */
CALL ENTRY$ENTRY;
/;/* 41  */  <TYPE IDENT> ::= <IDENTIFIER> /* */
IF LOCKUPSNIL(SP, TYPENAME) THEN
  TYPESLOC = LOCKUP$ADDR;
ELSE DO;
  CALL ERROR('TI');
  TYPESLOC = SILEN$ENTRY; /* INTEGER DEFAULT */
END:
/;/* 42  */  <IDENT STRING> ::= <IDENTIFIER> /* */
DO;
  TYPESRSOP$ = 0;
  CALL USER$TYP(TYPEDORD$NUM);
END;
/;/* 43  */  <IDENT STRING>, <IDENTIFIER> /* */
DO;
  TYPEDORD$ = TYPESORD$ - 1;
  CALL USER$TYP(TYPEDORD$);
END:
/* 44  */  <STRUCTURED TYPE> ::= <UNPACKED STRUCTURED TYPE> /* */
/;/* 45  */  <PACKED /* */
/;/* 45  */  <UNPACKED STRUCTURED TYPE> /* */
/;/* 46  */  <UNPACKED STRUCTURED TYPE> ::= <ARRAY TYPE> /* */
/;/* 47  */  <RECORD TYPE> /* */
/;/* 46  */  <SET TYPE> /* */

178
```c
/* 50 <FILE TYPE> */

/* 51 <ARRAY TYPE> ::= ARRAY [ <INDEX TYPE STRING> | <COMPONENT TYPE> ] */
CALL ARYP$DECLAR;
/* 51 <INDEX TYPE STRING> ::= <INDEX TYPE> */
DO:
IF ARYS$PTR=ARYS$NEST-1 THEN
DO;
CALL ERROR("AN");
ARYSDMSADR$PTR = ARYS$DM$ADR$PTR =
NUM$ARY$DM$ADR$PTR;
END;
ELSE ARYS$PTR = ARYS$PTR-1;
ARYSDMSPTR=2;
ARYSM$ADR$PTR=ARY$DM$ADR$PTR-1;
ARY$CTY(ARY$PTR) = ALS$NDX$OFFSET;
ARY$DM$EN(ARY$DM$ADR$PTR) = TYP$LOCT;
TEMPBASE=EASE;
BASE=TYPELOCT;
CALL SETADDR PTR(7);
ARY$DM$LOWVAL(ARY$DM$ADR$PTR) = ADRPTR;
CALL SETADDRPTR(3);
ARY$DM$HIVAL(ARY$DM$ADR$PTR) = ADRPTR;
CALL SETADDRPTR(11);
NUM$ARY$ELMTS(ARY$DM$ADR$PTR) = ADRPTR;
BASE=TEMPBASE;
NUM$ARY$SIM(ARY$PTR)=1;
END;
/* 52 <INDEX TYPE STRING> */
/* 52 <INDEX TYPE> */
DO:
IF ARYS$DM$PTR=MAX$ARY$DM-1 THEN
CALL ERROR("AL");
ELSE ARYS$DM$PTR=ARY$DM$PTR+1;
ARY$DM$ADR$PTR=ARY$DM$ADR$PTR+1;
ARY$CTY(ARY$PTR) = ARYS$CTY(ARY$PTR) *
ARY$DM$EN(ARY$DM$ADR$PTR) = TYP$LOCT;
TEMPBASE=EASE;
BASE=TYPELOCT;
CALL SETADDRPTR(7);
ARY$DM$LOWVAL(ARY$DM$ADR$PTR)=ADRPTR;
CALL SETADDRPTR(3);
ARY$DM$HIVAL(ARY$DM$ADR$PTR)=ADRPTR;
CALL SETADDRPTR(11);
NUM$ARY$ELMTS(ARY$DM$ADR$PTR)=ADRPTR;
BASE=TEMPBASE;
NUM$ARY$SIM(ARY$PTR)=NUM$ARY$SIM(ARY$PTR)-1;
END;
/* 53 <INDEX TYPE> ::= <SIMPLE TYPE> */
/* 54 <COMPONENT TYPE> ::= <TYPE> */
```
**Record Type**: ::= <record> <field list> END

```plaintext
/* 55 <record type> ::= <record> <field list> END */
/*
VARIANT$PART(RECS$NST) = FALSE;
BASE$TYPE$SLOC$T = BASE$PAR$SAEP(RECS$NST);
IF "VAR$CAS$VAL(RECS$NST) <> " THEN
CALL $PRCP(IV);
CALL $TADDRPTP(5);
ALTPRT = $PR$S$S$ENTRY;
RECS$NST = RECS$NST - 1;
END;

/* 56 <record> ::= RECORD */
/ *
RECS$NST = RECS$NST + 1;
APTRAADDR, RECS$PAR$SAEP(RECS$NST) = SETL;
APTPR$TR = 0; /* COLLISION ENTRY */
APTRADD$BP = APTRAADDR + 2;
ADD$PTR = PRV$SET$ENTRY;
PRV$SET$ENTRY = SBTBL;
APTRAADDR = APTRAADDR + 2;
BYTEPTR = 1; /* FORM FOR RECS$NST */
SEFE$SET = SBTBL + 6; /* ALCF# FOR REST OF ENTRY */
/ */ INITIALIZE RECORD */
/*
VARIANT$PART(RECS$NST), TAG$FD(ReC$NST) = FALSE;
FIX$OF$SET$S$E$E(RECS$NST) = 0000;
VAR$OF$SET$S$E$E(RECS$NST) = 0000;
CUR$OF$SET(RECS$NST) = 0000;
VAR$CAS$VAL(RECS$NST) = 0000;
RECORD$PTR = -1;
END;

/* 57 <field list> ::= <fixed part> */
/ *
58 <fixed part> ; <variant part> */
/ *
59 */
/ *
60 */
/ *
61 */
/ *
62 */
/ *
63 */
/ *
64 */
/* 65 <record section> ::= <field literal string> ; <type> */
/*
CALL ALLOC$OFFSET(TYPE$SLOC$T);
/* ALOC$BASE$TYP AND ALLO$QTY ARE SET */
DO PTRPTR = 2 TO RECORD$PTR;
FA SE = RECS$ALDR(PTRPTR);
CALL SET$PAS$P$N(9);
BYTEPTR = ALOC$BASE$TYP;
APTRAADDR = APTRAADDR + 1;
ALTPRT = TYPE$SLOC$T;
APTRAADDR = APTRAADDR - 1;
ALTPRT = CUR$OF$SET(RECS$NST);
```

190
CURSORST(RECONS) = CURSORST(RECONS)
END;

/* 63 */
RECORDS$TR = 1;
IF PDP$ST$SP$SE(RECONS) < CURSORST(RECONS)
THEN EXP$S$TP$ST(RECONS) = CURSORST(RECONS);
END;

/* 64 */<FIELD IDENT STRING> ::= <FIELD IDENT> /* 65 */
/* 65 */<FIELD IDENT STRING>,<FIELD IDENT> /* 66 */
/* 66 */<FIELD IDENT> ::= <IDENTIFIER>

DO;
IF RECORDS$PTR <> 10 THEN RECORDS$PTR = RECORDS$PTR + 1;
ELSE CALL ER$OP('EN');
RECS$ADR(RECORDS$PTR) = SETEL;
CALL FNTP$VAR$SID(REC$SP,TYP$STCL$);
IF NOT PRESENT THEN DO;
CALL LIMITS(7);
APTR$ADDR = SETEL;
ADDP$PTR = RECS$P$ADDR(RECONS);
SETEL = SETEL + 7;
END;
IF VARIANTS$PART(RECONS) THEN
DO;
BASE = RECS$ADDR(RECORDS$PTR);
CALL LIMITS(2);
CALL SETALL$P$PTR(4);
CY$TEPTR = CDP;
END;
END;

/* 67 */<VARIANT PART> ::= CASE<TAG FIELD><TYPE IDENT> OF /* 68 */
/* 68 */<VARIANT STRING> /* 69 */
/* 69 */<VARIANT STRING> ::= <VARIANT> /* 70 */
/* 70 */<VARIANT STRING> ; <VARIANT>/* 71 */
/* 71 */<TAG FIELD> ::= <FIELD IDENT> :
TAG$FL(RECONS) = TRUE;
/* 72 */<VARIANT> ::=<CASE LABEL LIST> : (<FIELD LIST>) /* 73 */
/* 73 */; /* 74 */
/* 74 */<CASE LABEL LIST> ::= <CASE LABEL>

DO;
LABEL$STACK(SP) = LABL$COUNT;
LABL$COUNT = LABL$COUNT + 2;
181
CALL GEN$SAVEEFP(KASF,ALLOC$CTY);
CALL GENERATE(LOW(LABE$STACK(SP)));
CALL GENERATE(HIGH(LABE$STACK(SP)));
END;

/* 76 <CASE LABEL LIST>, <CASE LABEL> */
DO:
  CALL GEN$SAVEEFP(KASF,ALLOC$CTY);
  CALL GENERATE(LOW(LABE$STACK(MP)));
  CALL GENERATE(HIGH(LABE$STACK(MP)));
END;

/* 76 <CASE LABEL> ::= <CONSTANT> */
IF CASE$STY THEN DO;
  CASE$STK(CASE$COUNT) = CASE$STK(CASE$COUNT) + 1;
  DO CASE EXP$SS$TP(SP) ;
    /* NUMBER */
    ALLOC$CTY = CONVETI(SP,PCS);
    /* IDENTIFIER */
    DO;
      IF NOT LOCK$UP$ONLY(SP) THEN CALL ERRO("TE");
      ELSE DO;
        BASE = LOOK$UP$ADD$P;
        CALL SET$PAST$PN(?);
        ALLOC$CTY = ADDR$PTR;
      END;
    END;
    /* SIGNED IDENTIFIER */
  DO;
  /* STRING TYPE */
END;
  /* END OF CASE*/
END;
ELSE DO:
  IF NOT VARI$NT$PART(REC$NST) THEN DO;
    VARI$NT$PART(REC$NST) = TRUE;
    VAR$CAS$TP(REC$NST) = TYPE$LOCT;
    VAR$CAS$VAL(REC$NST) = AL$NE$OFFSET;
    CALL ALLOC$OFFSET(TYPE$LOCT);
    IF TA$SET(REC$NST) THEN DO;
      TA$SET(REC$NST) = FALSE;
      BASE = REC$ADDR(REC$NT$PTR);
      CALL SET$ADDR$PTR(4);
      BYTE$PTR = 9FF;
      CALL SET$ADDR$PTR(8);
      CALL SET$ADDR$PTR(2*BYTE$PTR);
      ADDR$PTR = VAR$CAS$SP(REC$NST);
      AP$TR$PTR = AP$TR$PTR + 2;
      ADDR$PTR = CUR$OFFSET(REC$NST);
    END:
CURSDFST(RECONST)=CURSDFST(RECONST)-ALLTITY;
VAR$DFST(RECONST)=CURSDFST(RECONST);
EIT$DFST(RECONST)=CURSDFST(RECONST);
END;
/** CALL COMPARE$CONST$VARIANTE: /**
/** CHECKS THE CASE LABEL WITH THE VARIANTE TYPE. **/
CURSDFST(RECONST)=VAR$DFST$SP(RECONST);
VecPTR=VecPTR-1;
CONST$PTR, CONST$INDEX, CONST$PN$PTR=V;
END;
/** 75 <SET TYPE> := SET OF <PASE TYPE> */
CALL FNTR$STR$TYPE(PTH);
/** 76 <BASE TYPE> := <SIMPLE TYPE> */
/** 79 <FILE TYPE> := FILE OF <TYPE> */
CALL FNTR$STR$TYPE(STR);
/** 80 <POINTER TYPE> ::= <TYPE IDENT> */
CALL FNTR$STR$TYPE(IDE);
/** 81 <VDOP> ::= */
SCOP$SCOPE$NUM = SETIL;
/** 82 VAR <VAR DECLAR STRING> ; */
SCOP$SCOPE$NUM = SETIL;
/** 83 <VAR DECLAR STRING> ::= <VAR DECLAR> */
/** 84 <VAR Declar> ::= <IDENT VAR STRING> : <TYPE> */
DO;
CALL ALLOC$VARS;
END;
/** 86 <IDENT VAR STRING> ::= <IDENTIFIER> */
DO;
VAR$PTR = $O;
PARENT$TYPE, VAR$BASE(VAR$PTR) = SETIL;
CALL FNTR$VAPID($O, SP, VAR$ENTRY);
END;
/** 87 <IDENT VAR STRING> , */
/** 87 IF VAR$PTR <> 10 THEN */
DO;
VAR$PTR = VAR$PTR + 1;
VAR$BASE(VAR$PTR) = SETIL;
CALL ENTER$VAR$ID($O, SP, VAR$ENTRY);
END;
ELSE CALL FPR$F("$N");
/** 89 <P$FTP> ::= */
CALL STTSAV$BLCX;
/** 89 <POPF DECLAR> */
CALL SETSAVESLCK;
/** 90 <POPF DECLAR> ::= <PROC OP FUNCTION> ; */
/** 91 <POPF DECLAR> <PROC OP FUNCTION> ; */
CALL HEAD$SEL;

IF NOT LOOKUP$ONLY(SP) THEN CALL ERROR("IT");
ELSE DO:
    PASS = LOOKUP$ENTRY;
    CALL SETADLPTR(5);
    IF BYTEPTR = 21 THEN CALL FND$SUPRN;
END;

IF proc $ID THEN CALL F:S U?-T -1

CALL GOT$PARAMS;
CALL GOT$PARAMS;
CALL GOT$PARAMS;
CALL ENTER$ENTRY;

PAPAM$NUM = 2;
CALL ENTER$ENTRY(2,SP,PROC$ENTRY);

CALL ALLOC$VAR;
CALL ALLOC$VAR;
CALL ALLOC$VAR;

VAR <PARA GROUP>

DO:
    TEMP$YTE = VAR$PTR;
    DO VAR$PARAM$STR = 0 TO TEMP$YTE;
    BASE = VAR$BASE (VAR$PARAM$STR);
    CALL SETADLPTR(4);
    BYTEPTR = BYTEPTR OR 00H;
END;

FUNCTION <PARA GROUP>

DO:
    TEMP$YTE = VAR$PTR;
    DO VAR$PARAM$STR = 0 TO TEMP$YTE;
    BASE = VAR$BASE (VAR$PARAM$STR);
    CALL SETADLPTR(4);
    BYTEPTR = FUNC$ENTRY OR 00H;
END;

PROCEDURE <PROC IDENT LIST>

PROCEDURE <PROC IDENT LIST> ::= <IDENTIFIER>

164
DO;
VARS$TPR=0;
PARAMNUM = PARAMNUM + 1;
VARS$BASE(VARS$TPR)=SETRL;
CALL ENTER$SUERTN(0,SP,FUNC$ENTRY);
END;
/* 127 */  <PROC IDENT LIST> , <IDENTIFIER>/
IF VARS$TPR <> 10 THEN
DO;
VARS$TPR=VARS$TPR+1;
PARAMNUM = PARAMNUM + 1;
VARS$BASE(VARS$TPR)=SETRL;
CALL ENTER$SUERTN(0,SP,FUNC$ENTRY);
END;
FLSF CALL PRFOR('VN');
/* 128 */  <PAPA GROUP> ::= <PARA IDENT LIST> : <TYPE IDENT>/
/* 129 */  <PARA IDENT LIST> ::= <IDENTIFIER> /
DO;
VARS$TPR=0;
PARAMNUM = PARAMNUM + 1;
VARS$BASE(VARS$TPR)=SETRL;
CALL ENTER$VAR$ID(0,SP,FUNC$ENTRY);
END;
/* 113 */  <PARA IDENT LIST> , <IDENTIFIER>/
IF VARS$TPR <> 10 THEN
DO;
VARS$TPR=VARS$TPR+1;
PARAMNUM = PARAMNUM + 1;
VARS$BASE(VARS$TPR)=SETRL;
CALL ENTER$VAR$ID(0,SP,FUNC$ENTRY);
END;
ELSE CALL PRFOR('VN');
/* 111 */  <FUNCTION READING> ::= <FUNCT ID> : <RESULT TYPE> : */
DO;
CALL GOTS$PARAMS;
CALL GOTS$FUNC$TYPE;
END;
/* 112 */  <FUNCT II> (: * /
/* 112 */  <FORMAL PARA SFL CT LIST> ) : */
/* 112 */  <RESULT TYPE> : */
DO;
CALL GOTS$PARAMS;
CALL GOTS$FUNC$TYPE;
CALL ALTER$RETURN;
END;
/* 113 */  <FUNCT II> ::= FUNCTION <IDENTIFIER>
/*
DO;
PARAMNUM = 2;
CALL ENTER$SUERTN(0,SP,FUNC$ENTRY);
END;
/* 114 */  <RESULT TYPE> ::= <TYPE IDENT>/
CALL ALLOC$OFFSET(TYPELOCST);
/** 116 \t<STMT> ::= <PAL STMT>

   <UNBAL STMT>

   <LABEL DEF> <STMT>

   <PAL STMT> ::= <IF CLAUSE> <TRUE PART> ELSE <BAL STMT>

   CALL GENSALER(LBL, LABELSTACK(wp)-1);

   <SIMPLE STMT>

   <UNBAL STMT> ::= <IF CLAUSE> <TRUE PART> ELSE

   CALL GENSADDP(LPL, LABELSTACK(wp)-1);

   <IF CLAUSE> ::= IF <EXPRESSION> THEN

   IC;

   LABELSTACK(wp)=Llabcount;

   Llabcount=Llabcount+2;

   IF EXPRESSION(STK(wp+1))=BOCLANGTYPE THEN

   DO;

   CALL GENERATE(OUTX);

   CALL GENSALIP(ELC, LABELSTACK(wp));

   END;

   ELSE CALL ERROR("CE");

   END;

   <TRUE PART> ::= <BAL STMT>

   L0;

   CALL GENSALER(ERL, LABELSTACK(SP-1)+1);

   CALL GNSALDR(ERL, LABELSTACK(SP-1));

   END;

   <LABEL DEF> ::= <LABEL> :=

   IF LOCKUP$PN$ID(wp, LABEL ENTRY) THEN

   L0;

   CALL SETATIPTR(5);

   CALL SETADPTR(6+BTIPTR);  

   CALL GENSALDR(ERL, ALDPPTR);

   END;

   ELSE CALL ERROR("UL");

   <SIMPLE STMT> ::= <ASSIGNMENT STMT>

   <PROCEDURE STMT>

   <WHILE STMT>

   <REPEAT STMT>

   <FOR STMT>

   <CASE STMT>

   <WITH STMT>
END;

/* 142  \<TYPEDS LIST> ::= \<EXPRESSION>*/
EXPSCTR = 1;

/* 143  \<EXPRESSION> ::= \<EXPRESSION> , \<EXPRESSIOM>*/
EXPSCTR = EXPSCTR - 1;

/* 144  \<SIMPLE EXPRESSION> ::= \<EXPRESSION>*/

/* 145 */

/* 146  \<SIMPLE EXPRESSION> ::= */

/* 147 */
IF TRY$EXPR$TYPE THEN CALL PINT$SETFLCP;
ELSE CALL FPROR("GE");

/* 148 */
<RELATIONAL OPERATOR> ::= =
CALL SET$OP$TYPE(33);

/* 149 */
CALL SET$OP$TYPE(38);

/* 148 */
CALL SET$OP$TYPE(40);

/* 149 */
CALL SET$OP$TYPE(41);

/* 150 */
CALL SET$OP$TYPE(42);

/* 151 */
CALL SET$OP$TYPE(44);

/* 152 */
CALL SET$OP$TYPE(48);

/* 153 */
<TERM> ::= <FACTOR>/*

/* 154 */
<TERM> ::= <MULTIPLYING OPERATOR> <FACTOR>*/

DO;
IF READ$ARGS THEN
DO;
APPRADER = PARMNUMLCC(MP);
IF SHR(STR$PTCHR,2) THEN
CALL FPROR("NE");
END;
IF CHK$EXPR$TYPE THEN
DO;
DO CASE TYPE$STACK(MP)1;
/*0*/ IF EXPRESS$STK(SP) = 1H THEN CALL GENERATE(NUL);
ELSIF EXPRESS$STK(SP) = 2H THEN CALL GENERATE(MUL);
ELSIF EXPRESS$STK(SP) = 3H THEN CALL GENERATE(ISFC);
ELSE CALL ERROR("CP");
/*1*/ IF EXPRESS$STK(SP) = 1H THEN
DO;
CALL GENERATE(CNWI);/* CONVERT 1ST INTEGER */
CALL GENERATE(CN2I);/* CONVERT 2ND INTEGER */
CALL GENERATE(DIVP);
EXPRESS$STK(MP) = UNSIGN$EXPCH;
END;
ELSIF EXPRESS$STK(SP) = 3H THEN CALL GENERATE(DIV);
ELSE CALL FPROR("CE");
/*2*/ IF EXPRESS$STK(SP) = INTERPRET$TYPE THEN
CALL GENERATE(CIVI);
```
/* 155  <MULTIPLYING OPERATOR> ::= */
   CALL SET$OP$TYPE("\*");  /* */
/* 155  CALL SET$OP$TYPE("/");  */
/* 157  CALL SET$OP$TYPE("%");  */
/* 159  CALL SET$OP$TYPE("&")  */
/* 160  <SIMPLE EXPRESSION> ::= <TERM> */
/* 161  <SIGN> <TERM> */
DO:
   IF READPARMS THEN DO:
      APTRADR = PARMNUMLOC(SP);
      IF SHR(BYTEPTR,7) THEN
         CALL ERROR("NE");
   END;
   IF SGN$TYPE = NEG THEN DO:
      IF EXPRES$STK(SP) = ~SIGN$EXPCH THEN
         CALL GENERATE(NEG);  
      ELSE IF EXPRES$STK(SP) = INT7GR$TYPE THEN
         CALL GENERATE(NEG1);
      ELSE CALL ERROR("LO");
   END;
   SGN$FLAG = FALSE;
   CALL COPY$STACKS(MP,SP);
END;
/* 162  <SIMPLE EXPRESSION> */
/* 162  <ADDITION OPERATOR> <TERM> */
DO:
   IF READPARMS THEN DO:
      APTRADR = PARMNUMLOC(MP);
      IF SHR(BYTEPTR,7) THEN
         CALL ERROR("NE");
   END;
   IF CHK$EXPR$TYPE THEN DO:
      IF TYPE$STACK(MP1) = 57 THEN /* ARITH ADD */
         DO CASE EXPRES$STK(SP);
            CALL GENERATE(UNION);  /* CASE 0 = ORD TYPE */
```
CALL GENERATE(ATIII); /* CAST 1 - INTEGER */
CALL ERROR("CE"); /* CAST 2 - INTEGER */
CALL GENERATE(ALDPE); /* CAST 3 - REAL */
CALL ERROR("CE"); /* CAST 4 - STRING */
CALL ERROR("CE"); /* CAST 5 - BOOLEAN */

END/"" CAST */
FILE IF TYPE$STACK(MPP1) = SP THEN /* ARITH SUBPR */
DC CASE EXPRESSION(SP):
CALL GENERATE(STDF); /* CASE 0 - OPD TYPE */
CALL GENERATE(SUBI);
CALL ERROR("CE");
CALL GENERATE(SUBP);
CALL ERROR("CE");
CALL ERROR("CE");
END;
ELSE IF TYPE$STACK(MPP1) = SP THEN /* BOOLEAN OR */
DO;
IF EXPRESSION(STK(SP)) = BOOLEAN$TYPE THEN
CALL GENERATE(BOR);
ELSE CALL ERROR("CE");
END;
ELSE CALL ERROR("CE");
END;
/* 163  ASSIGN OPERATOR */
/* 164 */
CALL SET$OP$TYPE(26H);
/* 165 */
CALL SET$OP$TYPE(36H);
/* 166 */
CALL SET$OP$TYPE(37H);
/* 167 */
IF ($FORM$FIELD(SP) = 000) OR ($FORM$FIELD(SP) = 7FH) THEN
CALL CALL$AS$POPC(FALSE);
ELSE
CALL LOAD$VARI(SP);
/* 167 */
CALL CALL$AS$POPC(TRUE);
/* 168 */
CALL COPY$STACKS(FP, MPP1);
/* 169 */
/* NOT <FACTORM
DO;
IF EXPRESSION(STK(SP)) = BOOLEAN$TYPE THEN
CALL GENERATE(NOT);
ELSE CALL ERROR("CE");
CALL COPY$STACKS(MP, SP);
END;
/* 171 */
/* <NUMBER
IF TYPE$NUM = INTEGER$TYPE THEN
DO;
EXPRESSION(STK(SP)) = INTEGER$TYPE;
ALLOC$CTY = CONVE$PI(SP, POS);
CALL GEN$ADD(LDII, ALLOC$CTY);
100
FULL:
EXPRESS$STK(SP) $=SIGN$EXPON;
CALL CONVERTE(SP,PUSP);
CALL GENERATE(LDSI);
DO PTRPTR=0 TO PCTSIZE-1;
CALL GENERATE(BOINUM(PTRPTR));
END;

/* 172 */ ~NIL
;
/* 173 */ ~<STRING>
DO:
EXPRESS$STK(SP) $=STRING$TYPE;
CALL GENERATE(LDSI);
DO FOREVER;
DC PTRPTR = 1 TO ACCUM(2):
CALL GENERATE(ACCUM(PTRPTR));
END;
IF CONT THEN /* STRING > 32 CHARs */
CALL SCANSTR,
ELSE DO:
CALL GENERATE(NCP);
RETURN;
END;

END;

/* 174 */ <ACTUAL PARA LIST> ::= <ACTUAL PARA>
PARAMUM(SP) $= 1;
/* 175 */ <ACTUAL PARA LIST> $= <ACTUAL PARA>
/* 176 */ <ACTUAL PARA LIST> $= [ <ELEMENT LIST> ]
CALL COPY$STACKS(SP, MPP1);
/* 177 */ <ELEMENT LIST> ::=<ELEMENT LIST>
CALL COPY$STACKS(SP, SP-3);
/* 178 */ <ELEMENT LIST> ::=<ELEMENT>
;
/* 179 */ <ELEMENT LIST> ::=<ELEMENT>
;
/* 180 */ <ELEMENT LIST> $= <ELEMENT>
IF EXPRESS$STK(MP) $= EXPRESS$STK(SP) THEN
CALL EPROP('IT');
/* 181 */ <ELEMENT> ::=<EXPRESSION>
;
/* 182 */ <EXPRESSION> $= <EXPRESSION>$,
IF EXPRESS$STK(MP) $= EXPRESS$STK(SP) THEN CALL EPROP('IT');
/* 183 */ <GOTO STMT> ::=GOTO <LABEL>
IF LOCKUPS$PL(SP, LAEL$ENTRY) THEN DO:
CALL SETADDRPTR(5);
CALL SETADDRSP(6+EXTPTR);
CALL GEN$ADDR(BRL,ADDPTR);
END;
ELSE IF:
    CALL ERROR( "UL" );
    CALL GENERATE( "YOP" );
    CALL GENERATE( "YOP" );
END;

/* 154  <COMPOUND_STMT> ::= BEGIN <STMT_LISTS> END */
/* 155  <STMT_LISTS> ::= <STMT> */
/* 156     <STMT_LISTS> ::= <STMT> */
/* 157  <PROCEDURE_STMT> ::= <PROCEDURE_IDEN> */
/* 158     <PROCEDURE_IDEN> ( */
/* 159     <actual_para_list> */
/* 160     IF FORMSFIELD( MP ) = BUILTINSPROC THEN */
/* 161     CALL CALL$A$PROC( FALSE ); */
/* 162     ELSE CALL CALL$A$PROC( TRUE ); */
/* 163  <PROCEDURE_IDEN> ::= <IDENTIFIER> */
DO;
    IF NOT LOOKUP$ONLY( SP ) THEN
        CALL ERROR( "UP" );
    ELSE DO;
        PASCLOC( SP ) = LOOKUP$ADDR;
        CALL SETADDRPTR( 4 );
        FORMSFIELD( SP ) = BYTEPTR;
        IF FORMSFIELD( SP ) = BUILTINSPROC THEN DO;
            CALL SET$PAST$PN( 7 );
            IF BYTEPTR = 26 THEN DO:
                PARMNUM( SP ) = 2;
                PARMNUMLOC( SP ) = APTRADDR + 1;
            END;
            ELSE IF BYTEPTR > 21 THEN DO CASE ( BYTEPTR - 22 ):
                NEW$STM = TRUE;
                DISPOS$STM = TRUE;
                READ$STM = TRUE;
                WRITE$STM = TRUE;
            END;
        END;
    END;
ELSE IF BYTEPTR = 22 THEN DO CASE ( BYTEPTR - 22 ):*/
END;
/* NOT BUILT IN */
ELSE IF:
    CALL SET$PAST$PN( 7 );
    PARMNUM( SP ) = BYTEPTR;
    CALL SET$PAST$PN( P );
    PARMNUMLOC( SP ) = ADDRPTR;
    APTRADDR = APTRADDR + 6;
    LABELSTACY( SP ) = ADDRPTR;
    PFADPARMS = TRUE;
    PARMNUMLOC( SP+2 ) = PARMNUMLOC( SP );
END;
END:
/* 190 */ <actual parm> ::= <expression>

if realstmt then call realstmt;
else if writestmt then call writevar;
else if not (headpars) then
do:
    headpars = true;
call generate(parm);
/* parameter is an expression value */
end;

/* 191 */ <expression> ::= <expression> ;<expression>

if not writestmt then call error("PE");
else do:
    if expexprstk(sf) <> integer$type then call error("PE");
call writevar(1);
end;

/* 192 */ <expression> ::=<expression> &<expression>

if not writestmt then call error("PE");
else do:
    if expexprstk(sp) <> unsigned$expn then call error("PE");
    if (expexprstk(sp) <> integer$type) and
       (expexprstk(sp-2) <> integer$type) then call error("AP");
call writevar(2);
end;

/* 193 */ <case stmt> ::= <case expression><case list> elem list

/* 194 */ end
do:
    case$stnm = true;
    if (expexprstk(mp1) = unsigned$expn) then
        call error("RT");
    labelstack(mp) = labelcount;
    labelcount = labelcount + 1;
    case$stk(case$count := case$count - 1) = 0;
end;

/* 195 */ <case list> elem list ::= <case list> elem list

if case$stnm then
do:
call gensall(epl,labelstack(mp-1));
call gens$addf(pl,(labelstack(mp)+1));
end;

/* 196 */ <case list> elem list

if case$stnm then
do:
call gens$addf(epl,labelstack(mp-1));
call gens$addf(epl,(labelstack(sp)+1));
end;
/* 167  <CASE LIST ELEMENT> ::= 
   CASE_STMT = FALSE; /*
/* 196  ^ <CASE PREFIX> <STM> */
/* 199  <CASE PREFIX> ::= <CASE LABEL LIST> : */
/ * 218  DO;
   CALL GENSAKER(EPL, LABELSTACK(MP));
   CALL GENSAKR(LBL, LABELSTACK(MP));
END;
/* 284  <WITH_STMT> ::= <WITH> <REC_VARIABLE_LIST> <DO> */
/* 288  <DO_STMT> */
/* 281  <WITH> ::= WITH */
/* 292  <REC_VARIABLE_LIST> ::= <VARIABLE> */
/* 303  <VARIABLE> */
/* 293  <DO> ::= DO */
/* 304  LABELSTACK(SP) = LABELCOUNT;
   CALL GENSAKR(EPL, LABELSTACK(SP));
   LABELCOUNT = LABELCOUNT + 1;
END;
/* 305  <WHILE_STMT> ::= <WHILE><EXPRESSION> <TO> <DO_STMT> */
/* 306  DO;
   CALL GENSAKR(EPL, LABELSTACK(MP));
   CALL GENSAKR(LBL, LABELSTACK(MP));
END;
/* 307  <EXPR> ::= SIMPLE */
/* 308  <SIMPLE> */
/* 309  CALL GENFASTE(LABELSTACK(SP));
END;*/
/* 290  194 */
IF NOT LOOKUPSONLY(SP) THEN
  CALL ERROR("CV");
ELSE DO;
  APTRADD = LOOKUPSALEP + 4;
  IF $PTPOCR = 1 BF THEN CALL ERROR("CV");
ELSE CALL SET$VAR$TYPE;
END;
END;
/
* 211  <INITIAL VALUE> ::= <EXPRESSION>
*
;
/
* 212  <FINAL VALUE> ::= <EXPRESSION>
*
;
/
* 213  <REPEAT SMT> ::= <REPEAT> <STAT LISTS> UNTIL <EXPRESSION>
*
;
* 214  <REPEAT> ::= REPEAT
*
;
* 215  <TO> ::= TO
*
;
* 216  <DOWN TO> ::= DOWN TO
*
END; /* OF CASE STATEMENT */
END SYNTHESIZE;

END SYNTEZ;
DEC0DEF.SFC

DECLARE:
LIT LITERALLY 'LITERALLY',
PCB ADDRESS INITIAL('SEC'),
PCB BYTE INITIAL(1),
I BYTE,
FALSE BYTE INITIAL(0),
TRUE BYTE INITIAL(1),
R8PESS ADDP INITIAL(100H),
BYTE, ADDP INITIAL(100H),
BYTE, ADDRESS, 'DECLAR',
FUNCTION', 'EXTERNAL',
BYTE, 'PROCEDURE',
BYTE, '0FFH',
BYTE.

MON1: PROC(FUNC, INFO) EXT;
DCL FUNC BYTE, INFO ADDRESS;
END MON1;

MON2: PROC(FUNC, INFO) BYTE EXT;
DCL FUNC BYTE, INFO ADDRESS;
END MON2;

ROOT: PPCS EXT;
END ROOT;

PRINT$CHAR: PROCEDURE (CHR) :
DECLARE CHAR BYTE;
CALL MON1(2, CHAR);
END PRINT$CHAR;

CPLP: PROC;
CALL PRINT$CHAR(13);
CALL PRINT$CHAR(10);
END CPLP;

196
P: PROCEDURE(ADD1);
   DECLARE ADDR ADDRESS, C FAST ALL '1' BYTE;
   CALL CPLP;
   DO I=0 TO 4;
      CALL PRINTSCHAR(C(I));
   END;
   CALL PRINTSCHAR( '
');
   END P;

GETSCHAR: PROCEDURE BYTE;
   IF (ADDR:=ADDR+1) > BUFSIZE THEN DO:
      IF MONP(ZP,PC3) <> 0 THEN TO:
         CALL P'(,'ENI ')');
      END;
      ADDR:=80;
      RETURN CHAR;
   END GETSCHAR;

WRITESSTRING: PROCEDURE;
   DECLARE J BYTE;
   DO WHILE 1;
      J = GETSCHAR;
      IF J <> 0 THEN CALL PRINTSCHAR(J);
      ELSE RETURN;
   END:
   END WRITESSTRING;

D$CHAR: PROCEDURE(OUTPUT$BYTE);
   DECLARE OUTPUT$BYTE BYTE;
   IF OUTPUT$BYTE < 10 THEN CALL PRINTSCHAR(OUTPUT$BYTE + 
      36); FALSE CALL PRINTSCHAR(OUTPUT$BYTE + 37H);
   END D$CHAR;

D: PROCEDURE (COUNT);
   DECLARE (COUNT, J) ADDRESS;
   DO J=1 TO COUNT;
      CALL D$CHAR(SHR(GETSCHAR,4)):
      CALL D$CHAR(CHAR AND 0TH);
      CALL PRINTSCHAR( '
');
   END;
   END D;
PRINT$BECT: Procedure (COUNT);
DCI (COUNT, J, L, K) BYTE;

P*EXPON: PROCEDURE (VALUE);
DCI (VALUE, X, COUNT) BYTE;
DCI FLAG BYTE;
DI X = 0 TO 2;
FLAG = FALSE;
COUNT1 = 2C;
DO "WHILE VALUE >= DECIAK):
VALUE = VALUE - DECIAK;
FLAG = TRUE:
COUNT1 = COUNT1 + 1;
END:
IF FLAG OR (X = 2) THEN
CALL PRINT$CHAR(COUNT1);
ELSE CALL PRINT$CHAR(" ");
FND;
RETURN;
END P*EXPON:

DO L = 0 TO (COUNT-1);
DCNUM(L) = GET$CHAR;
ENT:
CALL PRINT$CHAR(" ");
IF DCNUM(COUNT-1) >= 80H THEN CALL PRINT$CHAR("-");
ELSE CALL PRINT$CHAR("+");
CALL PRINT$CHAR("0");
CALL PRINT$CHAR("-");
DO L = 0 TO COUNT-2;
J, K = DCNUM(L);
K = SFR((X AND £2H) 4); /* EXTRACT THE MSP FM THE BYTE */
******************************************************************************
CALL D$CHAR(K);
J = (J AND 8FH); /* EXTRACT THE LSP FM THE BYTE */
CALL D$CHAR(J);
END:
J, K = (DCNUM(COUNT-1) AND 7FH); /* GET RID OF SIGN */
IF K = 40R THEN CALL PRINT$CHAR("+");
ELSE CALL PRINT$CHAR("-"); /* SIGN OF EXPONENT */
CALL PRINT$CHAR("E");
CALL P*EXPON(K AND 2FH);
END PRINT$BECT;

PRINT$REST: Procedure;
DC$CHAR
IF CHAR = FNTP THEN
  DO:
  CALL P(.("END "));
  CALL BOOT;
END;

IF (CHAR=WPST) OR (CHAR=WRTI) OR
(CHAR=WRTS) THEN LO; CALL D(1); RETURN; ENI;
IF (CHAR=LRP) OR (CHAR=LRL) OR (CHAR=LIT) OR (CHAR=LITA) OR
(CHAR=BPL) OR (CHAR=BC) OR (CHAR=PLC) OR (CHAR=PPG) OR
(CHAR=PAR) OR (CHAR=PARY) THEN TO:
  CALL D(2); RETURN; ENI:
IF CHAR = KASE THEN DO; CALL P(4); RETURN; ENI;
IF CHAR = LIT THEN DO; CALL PRINTSLOT(5); RETURN; ENI;
IF CHAR = LDSI THEN DO; CALL WRITESSTPINC; RETURN; ENI;
END;
ENT PRINTSSTPINC;

/***** PROGRAM EXECUTION STARTS HERE *****/
MAINLICE: LO;
IF MCN2(15, PFB) = 255 THEN
  DO:
  CALL P(.("NO FITF FOUND");
  CALL BOOT;
END;
DO WHILE 1;
IF GETCHAR <= 72 THEN
  DO CAST CHAR;
  CALL P(.("NO")");
  199
CALL P('.('STEP '));
CALL P('.('STI '));
CALL P('.('STD '));
CALL P('.('UNION '));
CALL P('.('STEIT '));
CALL P('.('ISFC '));
CALL P('.('CNAI '));
CALL P('.('ERL '));
CALL P('.('PIC '));
CALL P('.('CNAI '));
CALL P('.('NXSET '));
CALL P('.('YCS '));
CALL P('.('FAR '));
CALL P('.('FARM '));
CALL P('.('DAP '));
CALL P('.('INC '));
CALL P('.('TEL '));
CALL P('.('DLE '));
CALL P('.('SFT '));
CALL P('.('SUB '));
CALL P('.('LDSI '));
CALL P('.('CASE '));
CALL P('.('LOE '));
CALL P('.('LCDR '));
CALL P('.('LODI '));
CALL P('.('BLUE '));
CALL P('.('PLVI '));
CALL P('.('PIVS '));
CALL P('.('OTE '));
CALL P('.('AFT '));
CALL P('.('PTS '));
CALL P('.('TUMP '));
CALL P('.('ABS '));
CALL P('.('SCR '));
CALL P('.('STN '));
CALL P('.('CCS '));
CALL P('.('APCTN '));
CALL P('.('APCTN '));
CALL P('.('LM '));
CALL P('.('SCPT '));
CALL P('.('CT '));
CALL P('.('ECLN '));
CALL P('.('EXF '));
CALL P('.('TPUNC '));
CALL P('.('BOUND '));
CALL P('.('OPE '));
CALL P('.('CHR '));
CALL P('.('SUCC '));
CALL P('.('PPE '));
CALL P('.('SEEK '));
CALL P('.('PUT '));
CALL P('.('CIT '));
CALL P('.('RESET '));
CALL P('.('PWRT '));
CALL P ( "\".)" );
CALL P ( "\"..\" " );
CALL P ( "\"...\" " );
CALL P ( "\"......\" " );
CALL P ( "\"FWD\" " );
CALL P ( "\"YTRNL\" " );
CALL P ( "\"ADV\" " );
END: /* OF CASE STATEMENT */
ELSE CALL P ( "\"ZZZ\" " );
CALL PRINTRES;
END: /* OF TO WHILE */
END PRINTLINE;
END IFLOGIC:
SYMBOLS

DECLARE

LIT LITERALLY 'LITERALLY',
EXT LIT 'EXTERNAL',
FCF ADDRESS INITIAL ('FCF'),
ADD LIT 'ADDRESS',
FCFBYTE BASED FCF (1) BYTE,
DPCC(5) ADDR INITIAL(122H,122H,122H,122H,122H),
DPCC(3) BYTE INITIAL(12H,12H,12H),
I BYTE '1',
TRUE LIT '0',
FALSE LIT '0',
COPYING BYTE INITIAL(TRUE),
ADDR ADDRESS INITIAL (120H),
CBAP BASED ADDPL BYTE,
BUFFSEND LIT '07H',
FORMMK ADDRESS 'DECLARE';
DCL LIT 'DECLARE',
PROCEDURE '1AH',
LIT 'PROCEDURE',
DATA ('S', 'Y', 'M'),
BUFFER BYTE,'TABLE',*/ STARTING LOCATION AT Compilation */
OFFSET ADR, */ NEW VALUE OF TABLE ENTRY */
PARMST LISTING(17) ADR, //LOCATION OF SUPREM FORMAL PARAMETERS

SUPERN. BYTT INITIAL(?),
PARMSNUM(13) BYTT, //KEEPS COUNT OF NUMBER OF PARAMETERS
SAVEPBASE ADR, //Saves base location
LPW BYTE, //LENGTH OF PRINNAME

BASE ADR, //BASE OF CURRENT ENTRY
SETBLOC ADR, //CURRENT TOP OF SYMBOL TABLE
SBT ADR
L ADDP, //LENGTH OF SYMPOL TABLE
PTR BASEL BYTP, //FIRST BYTE OF ENTRY
APTRADP ADDP, //UTILITY VAR FOR TABLE
ADDPTRP BASEL APTRADP ADDP
BYTEPTR BASEL APTRADP BYTE
PRINTNAMY ADDP, //SET PRIOR TO LOOKUP CP FSTEP
SYMASY BYTE.

MAC1 PROCEDURE (F,A) EXT,
DECLARE F BYTE, A ADDRESS;
END MAC1;

MAC2 PROCEDURE (F,A) BYTE BYT,
DECLARE F BYTE, A ADDRESS;
END MAC2;

ROOT PROC EXT;
END ROOT;

PRINTCHR PROCEDURE (CCHAR);
DECLARE CHAR BYTE;
CALL MAC1(2,CHAR);
END PRINTCHR;

CPLF PROCEDURE;
CALL PRINTCHR(17);
CALL PRINTCHR(10);
END CPLF;

PRINT PROC(A);
DCL A ADDRESS;
CALL MAC1(9,A);
END PRINT:
GETSCHAR: PROCEDURE BYTE;
  IF (A:ERl:=A:ERl-1) > BUFFER THEN
  TO:
  IF MOMP(22,703) <> 0 THEN
  DO;
   CALL PRINT(.('THE END $'));
  END;
  A:ERl:=B:SH;
  END GETSCHAR;

DISCHAR: PROCEDURE (OUTPUT$BYTE);
  DECLARE OUTPUT$BYTE BYTE;
  IF OUTPUT$BYTE < 10 THEN CALL PRINT$CHAR (OUTPUT$BYTE + 37H);
  ELSE CALL PRINT$CHAR (OUTPUT$BYTE + 38H);
  END DISCHAR;

C: PROCEDURE (COUNT);
  DECLARE (COUNT, J) AT FRSS;
  DO J=1 TO COUNT;
   CALL DISCHAR (SFR (BYTEPTR + 1));
   CALL DISCHAR (BYTEPTR AND 255);
   APTRADD = APTRADD + 1;
  END;
  END C:

/*******************************************************************************/
PRINT$BOT: PROCEDURE (COUNT);
  DECLARE (COUNT, J, X, L) BYTE;

P$EXPON: PROCEDURE (VALU);
  DECLARE (VALU, X, COUNT1) BYTE;
  DECLARE FLAG BYTE;
  DO X = 0 TO 2;
   FLAG = FALSE;
   COUNT1 = 30H;
   DO WHILE VALU >= DEC1(X);
    VALU = VALU - DEC1(X);
    FLAG = TRUE;
    COUNT1 = COUNT1 + 1;
   ENI;
  IF FLAG OR (X >= 2) THEN
   CALL PRINT$CHAR (COUNT1);
  ELSE CALL PRINT$CHAR (" ");
  END;
  RETURN;

205
END PSXYON;

EO L = 9 TO ('COUNT-1');
   RCDNUM(L) = BYTEPTP;
   APTRADP = APTRADDR-1;
END;
CALL PRINTSCFAR(0);
IF RCDNUM('COUNT-1') >= E2H THEN CALL PRINTSCFAR(' -');
   ELSE CALL PRINTSCFAR('+');
CALL PRINTSCFAR(0);
CALL PRINTSCFAR(1);
DO L = 2 TO COUNT-2;
   J.K = RCDNUM(L);
   K = SHR((J AND 2FFH),4); /* EXTRACT THE MSB FROM THE BYTE */
   CALL D$CFAR(K);
   J = (J AND 0FFH); /* EXTRACT THE LSB FROM THE BYTE */
   CALL D$CHAR(J);
END;
J,K = (RCDNUM('COUNT-1') AND 7FH); /* GET BIT OF SIGN */
IF K >= 4FH THEN CALL PRINTSCFAR(1); /* SIGN OF EXPONENT */
   ELSE CALL PRINTSCFAR(0);
   CALL PSXYON(K AND 3FH);
END PRINTSCF;

DOSYM: PROCEDURE;
PCPSBYTE(32), PCPSBYTE(0) = 2;
DO I = 6 TO 2;
   PCPSBYTE(I+9) = FILETYPE(I);
END;
IF MCVZ(16,FCB) = 255 THEN
   DO;
      CALL PRINT('. (EFFP--GONE TO BOOT !)');
      CALL BCTT;
   END;
END DOTSYM;

DISKERR: PROC;
DO;
   CALL PRINT('. (DF !) ');
   CALL BCTT;
END;
END DISKERR;

PRINTDFC: PPROC(VALUF);
   DCL VALUE ADDR, I BYTE, COUNT BYTE;
TCL = FLAG BYT$;
FLAG = FALSE;
DO I = 3 TO 4;
COUNT = 36F;
DO WHILE VALUE >= DECIS(I);
VALUE = VALUE - DECIS(I);
FLAG = TRUE;
COUNT = COUNT + 1;
END;
IF FLAG OR (I >= 4) THEN
CALL PRINTCHAR(COUNT);
FLAG = FALSE;
CALL PRINTCHAR(
');
END;
RETURN;
END PRINTDEC;

SETADDPTR: PROC(OFFSET);
DCL OFFSET ADDR;
ADDRTRAILR = BASE + OFFSET;
END SETADDPTR;

SET$PASTSPN: PROC(OFFSET);
DCL OFFSET BYTE;
CALL SYTADDRSPN(6);
CALL SETADDPTR(BYTEPTR - OFFSET);
END SET$PASTSPN;

COPY$$STRL: PROC ADDR;
/* COPIES FILE.SYM TO MEMORY. LOOKING FOR TWO FILLERS
(1AH) IN A ROW */
DCL X ADDR;
X = 2;
DO WHILE COPYING;
CALL SETADDPTR(X);
BYTEPTR = GETCHR;
X = X + 1;
IF BYTEPTR = FILLER THEN
DO;
X = X + 1;
CALL SETADDPTR(X);
BYTEPTR = GETCHR;
IF BYTEPTR = FILLER THEN
DO;
COPYING = FALSE;
BYTEPTR = CHR;
END;
END;
RETURN X;

237
END COPY$PTR:

RESET$LOCATION: PROC(A) ALL:
DCL A ADDR:
OFFSET = A - TABLE$START;
RETURN OFFSET;
END RESET$LOCATION:

TAB1: PROC;
CALL PRINT'(" $ ");
END TAB1:

TAB2: PROC;
CALL TAB1;
CALL TAB1;
END TAB2;

WRITE$ENTRY: PROC;
DO CASE (FORM AND $73);  
CALL PRINT'("LABEL ENTRY $ ");
CALL PRINT'("CONSTANT ENTRY $ ");
CALL PRINT'("TYPE ENTRY $ ");
CALL PRINT'("VARIABLE ENTRY $ ");
CALL PRINT'("PROCEDURE ENTRY $ ");
CALL PRINT'("FUNCTION ENTRY $ ");
CALL PRINT'("FILE ENTRY $ ");
CALL PRINT'("USER DECLARED ENTRY $ ");
FND: /* CASE */
END WRITE$ENTRY;

PRINT$SIZE: PROC;
DCL SIZE BYTE;
CALL SETADDRPTR(6);
SIZE = EYEPTR;
DO I = 1 TO SIZE;
   CALL SETADDRPTR(6+I);
   CALL PRINT$CHAR(EYEPTR);
FND;
CALL CRLF;
END PRINT$SIZE:

PAN$PTR: PROC(A);
DCL (A, BAS$1) ADDR;
BAS$1 = FASE;
BAS$ = A;
CALL SETADDR$PTR(7);
CALL CRLF;

208
CALL 'TAPI;
CALL 'PRINT(.'WITH LOW VALUE $');
IF (SHP(FORM,7) AND FORMMASK) THEN
  CALL 'PRINTSCAP(BYTEPTR);
ELSE CALL 'PRINT$FC(ADDPTR);
CALL 'PRINT(.'AND HIGH VALUE $');
CALL 'SETSAID$PTP(2);
IF (SHP(FORM,7) AND FORMMASK) THEN
  CALL 'PRINT$CAP(BYTEPTR);
ELSE CALL 'PRINT$FC(ALLPTR);
END RANGER;

USER$DEFINE: PROC;
DO CASE (SHP(BYTEPTR,E) AND FORMMASK);
  DO;
    CALL 'PRINT(.'ENUMERATED TYPE - $');
    CALL 'PRINT$IL;
    CALL 'PRINT(.'THE VALUE IS $');
    CALL 'SETPAST$PW(7);
    CALL 'PRINT$FC(BYTEPTR);
  END;
  END;
  DO CASE (SHP(BYTEPTR,E) AND FORMMASK);
    CALL 'PRINT(.'AN ENUMERATED SUBRANGE $');
    DO;
      CALL 'PRINT(.'AN INTEGER SUBRANGE $');
      LPN = LPN + 13; /* LENGTH OF 4TH ENTRY */
    END;
    CALL 'PRINT(.'A CHARACTER SUBRANGE $');
    END; /* OF CASE */
  CALL RANGER(EASE);
END;
DO;
CALL 'PRINT(.'AN ARRAY $');
CALL 'SETADDPPTR(5);
I = BYTEPTR;
LPN = LPN + 13 + (4*I); /* LENGTH OF 1ST ENTRY */
CALL 'CRLF;
CALL TAB2;
CALL 'PRINT(.'THE COMPONENT TYPE IS $');
CALL 'SETADDPPTR(10);
TO CASE PITYPTR;
    CALL 'PRINT(.'SCALAR $');
    CALL 'PRINT(.'INTEGER $');
    CALL 'PRINT(.'CHAR $');
    CALL 'PRINT(.'REAL $');
    CALL 'PRINT(.'STRING $');
    CALL 'PRINT(.'POINTER $');
END; /* OF CASE */
CALL 'CRLF;
CALL TAB2;
CALL 'PRINT(.'IT REQUIRES $');
CALL SETADP$TR(6);  
CALL PRINT$PC(ADDRPTR);  
CALL PRINT$EC("BYTES OF STORAGE");  
CALL CPE$;  
CALL TARG$;  
CALL PRINT$E("THERE IS AN $’");  
CALL PRINT$E("DIMENSIONS IN THIS ARRAY $’");  
CALL SETADP$TR(9);  
DO WHILE I <> 0:  
  APTRADDR = APTRADDR + 4;  
  CALL RANGE$(ADRPTR);  
  LPN = LPN + 13; /* LENGTH OF APE ENTRY */  
  I = I - 1;  
END;  
END;  
DO;  
END;  
TC$:  
CALL SETADP$TR(3);  
SAVE$BASE = BASE;  
BASE = ADEPTR;  
CALL PRINT$EC;  
BASE = SAVE$BASE;  
END;  
DO;  
CALL PRINT$E("A SET OF $’");  
CALL SETADP$TR(5);  
SAVE$BASE = BASE;  
BASE = ADEPTR;  
CALL PRINT$EC;  
BASE = SAVE$BASE;  
END;  
DO;  
CALL PRINT$E("A FILE OF $’");  
CALL SETADP$TR(6);  
SAVE$BASE = BASE;  
BASE = ADEPTR;  
CALL PRINT$EC;  
BASE = SAVE$BASE;  
END;  
DO;  
CALL PRINT$E("A POINTER OF TYPE $’");  
CALL SETADP$TR(5);  
SAVE$BASE = BASE;  
BASE = ADEPTR;  
CALL PRINT$EC;  
BASE = SAVE$BASE;  
END;  
END; /* OF CASE */  
END USER$DEFINED;  

CHECK$COLLISION: PROC;  
/* LOOKS FOR ADDRESS IN COLLISION FIELD, THEN REPLACES  
COLLISION CHAIN BACKWARD, PRINTING PRINT$AMTS. STOPS  
WHEN NO FURTHER COLLISIONS OR TABLE RUNS OUT. */  
CALL SETADP$TR(6);  
LPN = BYTEPTR;  
CALL TAP1$;  
CALL PRINT$E("EASY VALUE = $’");  

210
CALL $TATTPTR(E);  
CALL PRINT$(F, $TATTPTR);  
CALL SET$ADDRPTR(2);  
IF ALLPTR = 2 THEN  
   CALL PRINT$(" AND TYPE ARE NO COLLISIONS ");  
ELSIF  
   SAV$BASE = BASE;  
   TO WHILE ADDR$PTR >= TART$START;  
   BASE, ADDR$PTR = ADDR$PTR;  
   CALL PRINT$(" WHICH COLLIDES WITH ");  
   CALL PRINT$IE;  
   CALL SET$ADDRPTR(2);  
   CALL TAB2;  
END;  
IF ADDR$PTR = 2 THEN  
   CALL PRINT$(" AND THERE ARE NO FURTHER COLLISIONS ");  
ELSIF DO;  
   CALL PRINT$(" ANY OTHER COLLISIONS OCCUR IN THE ");  
   CALL CLR$;  
   CALL PRINT$(" BUILT-IN SYMBOL TABLE ");  
END;  
BASE = SAV$BASE;  
END;  
CALL CLR$;  
END CHCK$COLLISION;  
ENTRY$TED: PROC;  
   CALL WRITEENTRY;  
   CALL PRINT$II;  
   CALL CHCK$COLLISION;  
   CALL TAB1;  
END ENTRY$TED;  
CHCK$TYPE: PROC(A);  
   DCL A BYTE;  
   DCL TYPE BYTE;  
   TYPE = (SHR(A, 3) AND FORM$MASK);  
   TO CASE TYPE;  
   /* SCALAR-ORDINATE */  
   CALL PRINT$(" SCALAR ORDINATE ");  
   /* INTEGER */  
   CALL PRINT$(" INTEGER ");  
   /* CHARACTER */  
   CALL PRINT$(" CHARACTER ");  
   /* REAL */  
   CALL PRINT$(" REAL ");  
   /* COMPLEX */  
   LO;  
   SAV$BASE = BASE;  
   CALL SET$PAST$PN(9);  
   PAST$ = ADDR$PTR;  
   CALL SET$ADE$PTR(4);  
211
IF (BYTE$ AND FORM$) = 16 THEN
CALL USP$DEFINED;
ELSE CALL PRINT$;
BYTE$ = SAV$BYTE$;
END;
/* PROCLEAN */
CALL PRINT$ .("$OOL$AN $'");
END; /* CASE TYPE */
CALL CPLF;
CALL TAFI;
END CHECK$TYPE;

BEGIN $TYPE$CONST: PROC(A);
/* CHECK FOR TYPE OF CONSTANT AND PRINT IT */
ICL A BYTE;
DC CASE 4:
/* 0+ signer IDENTIFIER */
CALL PRINT$ .("+ UN-IDENTIFIER $'");
/* 1+ IN-TEGER */
CALL PRINT$ .("+ IN-TEGER $'");
/* 2+ REAL */
CALL PRINT$ .("+ REAL $'");
/* 3+ STPRINT */
CALL PRINT$ .("+ STRING $'");
/* 4,+6,+7 NOT DEFINED */
;
/* 2+ S+IGNER IDENTIFIER */
CALL PRINT$ .("+ S+IGNER IDENTIFIER $'");
END; /* CASE */
END CHECK$TYPE$CONST:

BEGIN PRINT$PRT: PROC(A);
ICL A BYTE;
IF A = 18 THEN
CALL PRINT$ .("THE ASSIGNED PRT LOCATION FOR THE SP IS $'");
ELSE CALL PRINT$ .("THE ASSIGNED PRT LOCATION IS $'");
CALL SFR$PAST$SP$IN/A/;
CALL PRINT$SP$DEF(AT$DRP$TP);
CALL CRLF;
END PRINT$PRT;

BEGIN PRINT$LAPELF: PROC;
CALL PNT$P$HEAD;
CALL PRINT$ .("THE ASSIGNED LAPELF VALUE IS $'");
CALL SFR$PAST$SP$N(7);
CALL PRINT$SP$EC(AT$DRPTR);
CALL CRLF;
END PRINT$LAPELF;
PRINTSCONST: PROC;
TC TYPE, SIZE, I) TYPE:
    CALL WRITESTATE;
    CALL PRINTID;
    CALL CHECKSCOLLISION;
    CALL PRINT(.' THE CONSTANT TYPE IS $');
    TYPE = (SHP FORM, 3) AND INDEX;
    CALL CHECKTYPESCONST(TYPE);
    CALL CRM;
    CALL PRINT(.' THE CONSTANT VALUE = $');
    IF TYPE = 1 THEN
        DO:
            CALL SFTSPAN(7);
            CALL PRINTDEC(ALDRPT);
            LPN = LPN + 8;
    END;
    IF TYPE = 2 THEN
        DO:
            CALL SFTSPAN(7);
            CALL PRINT(8);
            LPN = LPN + 16;
    END;
    IF (TYPE = 0) OR (TYPE = 2) OR (TYPE = 6) THEN:
        DO:
            CALL SETSPAN(7);
            SIZE = BYTEPTR;
            DO I = 1 TO SIZE;
                CALL SFTADDRPTR(7 - LPN + I);
                CALL PRINTCFAR(BYTEPTR);
    END;
    LPN = LPN + SIZE + 6;
    END:
END PRINTSCONST;

PRINTTYPE: PROC;
    CALL ENTRY$HEAL;
    CALL PRINT(.' THE PARENT TYPE IS $');
    DO CASP (SHP FORM, 3) AND FORM$ASK);
        CALL PRINT(.' INTEGER $'); /* 0 */
        CALL PRINT(.' FEAL $'); /* 1 */
        CALL PRINT(.' CHAR $'); /* 2 */
        CALL PRINT(.' BOOLEAN $'); /* 3 */
        /* 4 */
        /* 5 */
        /* 6 */
    DO:
        CALL SETSPAN(7);
        SAVE$BASE = EASE;
        BAST = ALDRPT;
        CALL SFTADDRPTR(4);
        IF (BYTEPTR AND FORM$ASK) = 0? THEN
            CALL USER$DEFINED;
        ELSE CALL PRINT$ID;

213
EAST = SAVESPAS;
END:
END /* OF CASE */
END PRINTTYPE;

PRINT$VARIABLE: PROC:
CALL ENTRY$HEADER;
CALL PRINT$("THE VARIABLE TYPE IS "$);
CALL CHECK$TYPE($FORM$);
CALL PRINT$SPRT$(?);
END PRINT$VARIABLE;

SUBROUTINE: PROC:
CALL J BYTE$;
CALL PRINT$("THERE ARE $"$);
CALL SET$PAST$SPN$(?);
J = BYTE$PTR;
CALL PRINT$DEC($BYTE$PTR$);
CALL PRINT$("PARAMETERS $"$);
CALL CPL$;
CALL SET$PAS$N$(?);
PARAM$LISTING($SUBRN$=SUBRN$+1), APTRADDR = ALTRPTR$;
PARAM$NUM$(SUBRN$) = J;
DO I = 1 TO J:
   CALL TAE$;
   CALL PRINT$("NC. $"$);
   CALL PRINT$DEC$(I);
   CALL TAE$;
   IF SUB$(BYTE$PTR,7) THEN:
      IF SUB$(BYTE$PTR,6) THEN CALL PRINT$("FUNCTION $"$);
         ELSE CALL PRINT$("VALUE $"$);
         END;
      ELSE IF BYTE$PTR = 4 THEN CALL PRINT$("PROCEDURE $"$);
         ELSE CALL PRINT$("VALUE $"$);
         CALL CHECK$TYPE($FORM$);
         APTRADDR = APTRADDR + ?;
      END; /* DO I */
   CALL PRINT$SPRT$(10);
   CALL PRINT$SPRT$(12);
   CALL TAE$;
   CALL PRINT$("THE LABEL VALUE PRECEDING THE CODE IS $"$);
   CALL SET$PAST$SPN$(14);
   CALL PRINT$DEC$(ALTRPTR$);
   CALL CR$LF$;
END SUBROUTINE;

BRANCH: PROC:
SUBRN$ = SUBRN$ + (3 * PARAM$NUM$(SUBRN$));
SUBRN$ = SUBRN$ - 1:
END PROC:

PRINT$PROC: PROC:
  CALL ENTRY$HEADER;
  CALL SUBROUTINE;
END PRINT$PROC:

PRINT$FUNC: PROC:
  CALL ENTRY$HEADER:
  CALL PRINT(’("THE FUNCTION "TYPE IS $")
  CALL SETTYPE(16);
  FORM$ = BYTEPTR;
  CALL CHECK$TYPE(FORM$);
  CALL SUPPORT;
END PRINT$FUNC:

PRINT$FILE: PROC:
  CALL ENTRY$HEADER;
END PRINT$FILE:

SKIP$PROC: PROC;
  TO CASE(SHR(FORM,3) AND FORM$ASK);
    DC:
      CALL SETADLPTR(6);
      SETL$ = SBTBL + 12 + BYTEPTR;
    FND;
    SBTBL$ = SBTBL + 16;
    DC:
      CALL SFTADLPTR(5);
      SBTBL$ = SBTBL + 12 + (2 * BYTEPTR);
    END;
    DC:
      IF FORM$ = "IP" THEN SBTBL$ = SBTBL + 9;
      ELSE DO:
        CALL SETADLPTR(6);
        SBTBL$ = SBTBL + 14 + BYTEPTR;
      END;
      END;
      SBTBL$ = SBTBL + 7;
      SETL$ = SBTBL + 7;
      SBTBL = SBTBL + 7;
    END: /* OF CASE */
END SKIP$PROC;

STAPS: PROC;
  CALL CRLF;
  CALL PRINT(’("---------------------------------------$")
  CALL CRLF;

215
MOVE: PROC(SOURCE,DESTIN,L):
DCL (SOURCE,DESTIN,L) ADDR,
(SCHAP BASED SOURCE, DCHEAP BASED DESTIN) BYTF;
DO WHILE (L=L - 1) <> FFFF:
DCHEAP=SCHAP;
DESTIN=DESTIN-1;
SOURCE=SOURCE+1;
END:
END MOVE:

MAINLINE:TC;
CALL DOTSYN;

BASE, SETBL = MEMORY;
L = CCPTSETBL;
CALL S$ADDREPTR(4);
FORM = BYTEPTR;
DO CASE (FORM AND FORM$MASK):
  CALL SET$PAST$SN(11);
   DO;
     CALL SET$ADRPTR(4);
     IF SETP(BYTEPTR,4) THEN CALL SET$PAST$SN(17);
     ELSE CALL SET$PAST$SN(11);
   END;
  CALL SET$PAST$SN(11);
  CALL SET$PAST$SN(13);
  CALL SET$PAST$SN(12);
  CALL SET$PAST$SN(19);
  CALL SET$PAST$SN(9);
/* THIS ENTRY IS IMPOSSIBLE FOR THE FIRST ENTRY */
END; /* CASE FORM */
/* STARTING LOCATION OF THE SYMBOL TABLE */
TABLE$START = ADDRPTR;
CALL MOV$(SETBL,TABLE$START,L);
BASE, SETBL = TABLE$START;
/* START */
CALL S$ADDREPTR(2);
DO WHILE ADDRPTR <> JØH;
   CALL SET$ADRPTR(4);
   FORM = BYTEPTR;
   CALL STARS;
   DO CASE (BYTEPTR AND FORM$MASK);
/* LABEL */
   DO;
     CALL PRINT$LABEL;
     SETBL = SETBL + 9 + LPN;
   END;
/* CONSTANT */
DC;
CALL PRINT$CONST;
CALL PRINT$TYPE;
SETBL = SETBL - LPN;
/* TYPE */
DO;
   CALL PRINT$TYPE;
   SETBL = SETBL + 9 + LPN;
END;
/* VARIABLE */
DO;
   CALL PRINT$VARIABLE;
   SETBL = SETBL + 11 + LPN;
END;
/* PROCEDURE */
DO;
   CALL PRINT$PROC;
   SETBL = SETBL + 16 + LPN;
END;
/* FUNCTION */
DO;
   CALL PRINT$FUNC;
   SETBL = SETBL + 17 + LPN;
END;
/* FILE */
DO;
   CALL PRINT$FILE;
   SETBL = SETBL + 7 + LPN;
END;
/* USER DEFINED ENTRY */
DO;
   CALL SKIPPER;
END:
/* OF CASE */
IF SETBL = PARM$LISTING(SUBRTW) THEN CALL TRANCH;
ELSE = SETPL;
CALL SETADDRPTR(2);
END:
CALL CRLF;
CALL PRINT$("THE SYMBOL TABLE HAS BEEN PRINTED.");
CALL ROOT;
END MAINLINE;
END SYM;
LIST OF REFERENCES


### INITIAL DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Address</th>
<th>Code</th>
<th>City, State, Zip Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>2</td>
<td>Defense Documentation Center</td>
<td></td>
<td>Cameron Station, Alexandria, Virginia 22314</td>
</tr>
<tr>
<td>2.</td>
<td>2</td>
<td>Naval Postgraduate School</td>
<td>0142</td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>3.</td>
<td>1</td>
<td>Department Chairman</td>
<td>52</td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>4.</td>
<td>1</td>
<td>Asst Professor E. J. MacLennan</td>
<td>52M1</td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>5.</td>
<td>1</td>
<td>IT Mark S. Moranville, USN</td>
<td>52M1</td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>6.</td>
<td>1</td>
<td>LLDP Robert F. Stilwell, USN</td>
<td>52Sb</td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>7.</td>
<td>1</td>
<td>LDR Frank Burkhead, USN</td>
<td>52Fg</td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>8.</td>
<td>1</td>
<td>Microcomputer Laboratory</td>
<td>0131</td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>9.</td>
<td>1</td>
<td>Capt Konrad S. Tinius, USAF</td>
<td></td>
<td>HQ USEUCOM (ECAFP) APO NY 09128</td>
</tr>
</tbody>
</table>