UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

IMPLEMENTATION OF A STRUCTURED PL/I SUBSET COMPILER

by

COLIN BARRY GOLDBERG

A Thesis

Prepared under the Supervision of

Mr. K.J. McGregor

In Fulfilment of the Requirements for the

DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

Cape Town

September, 1974.

"IF FORTRAN HAS BEEN CALLED
AN INFANTILE DISORDER, PL/I
MUST BE CLASSIFIED AS A FATAL
DISEASE."

E. Dijkstra
in the
Introduction to the Art of
Computer Programming.

LIST OF CONTENTS

LIST OF CONTENTS (continued)

LIST OF ILLUSTRATIONS

ABSTRACT

The thesis describes the design and implementation of a PL/I subset compiler which produces a hypothetical stack code as output.

The compiler was based on a Pascal compiler developed by N. Wirth and U. Amman of Eidgenössische Technische Hochschule, Zurich, and was itself written in the Pascal language. PL/I programs using the compiler can now be compiled and executed (interpretively) on the UNIVAC 1106 computer at U.C.T.

The compiler was designed mainly as a teaching system. Its lesson is that structured programming is a powerful technique which facilitated its design and implementation.

CHAPTER 1

INTRODUCTION.

1. INTRODUCTION.

The implementation of a compiler would seem at first to be a gigantic task, more suited to production by a team of coders rather than a single thesis. Without debating this point, this paper is presented with the intention of showing how the very nature of the compiler, that is, as a structured program, facilitated its implementation.

Several compilers for PL/I or a subset of the language have appeared in recent years. IBM's range of PL/I compilers, originally defined by the Share group (Share, 1964), includes its PL/ONE(F), CALL/OS PL/ONE, ITF, optimising and Checkout compilers, but these were developed as full, commercial compilers rather than teaching-oriented ones.

PLAGO is a PL/I subset compiler developed at Brooklyn Polytechnic.

SPLINTER is a (scientific) PL/I subset interpretive system (Glass, 1968), developed by one person at the Boeing Company to be a modular highly flexible system which can be modified to process other source languages.

PLUTO, a teaching-oriented PL/I compiler system (Boulton & Jeanes, 1972), derives from the University of Toronto, and uses the approach of preserving much information relating to the structure of the source program, in its generated code, for use during execution.

Probably most well known is PL/C, developed at Cornell University (Morgan & Wagner, 1971), as a 'high performance' compiler whose main feature is the high degree of diagnostic assistance given to the user. Errors are not only detected, but an attempt is made to correct them and continue, without inhibiting execution.

A feature about all of the implementations mentioned above which were designed for a University environment, is that each has chosen a subset of the PL/I language particularly suitable to its needs and facilities, so that no two compilers are exactly the same.

Being a newcomer to the scene, PL/I has yet to be fully standardised, although efforts are underway through the ANSI committee X.3.J1 and the ECMA committee TC-10. (MacLaren, 1970).

The programming language at U.C.T., referred to hereafter as PL/UCT, represents a significant subset of PL/I, and in that subset, the compiler is completely compatible with all the compilers or compiler systems mentioned above, for a correct program.

The most distinctive feature about the PL/UCT compiler is that it is a structured compiler, and, unlike the other compilers which were mostly written in low level (assembler) languages, it was coded in the Pascal language. This had a tremendous impact on the compiler in that it hastened its development and facilitated its implementation, in ways that will be demonstrated in the following pages.

Apart from conforming to the rules of syntax for the subset of PL/I, the implementation was effected with the following major objectives in mind.

1) To produce a working compiler for a subset of PL/I using the structured method of programming, whose output was a 'hypothetical stack code' suitable for interpretation.

2) To orient the compiler towards teaching, rather than as a production compiler, by:

a) creating a low-overhead batch system, easy to use by novice students; and

b) including as far as possible diagnostic facilities and debugging aids during compilation and execution, in terms linked as closely as possible to the source program. The compiler should provide automatic diagnostic assistance without attempting to correct errors.

The aims of the compiler may be emphasized by a quote in SIGPLAN notices (Holt, 1973), under the heading 'Some vain suggestions':

> 'We must teach structured programming. The structured programming approach is just plain superior to encouraging students to spend their time inventing tricks to save micro-seconds. Students must be taught how to break large programming problems cleanly into smaller ones.
>
> We should teach precision programming. That is, a student must be taught that a good program is one which is correct before it is run.'

Thus the coding criteria were chosen, in order of importance:-

1. compatibility (within the subset)
2. diagnostic assistance
3. ease of use
4. compilation speed and space requirements
5. execution speed

In order that the compiler operate as a teaching system in the batch mode, it was designed as an in-core, one-pass processor. The compiler itself exists in the 'hypothetical stack code' which it produces, so that both compilation and execution are interpreted. The implementation was accomplished through the use of an interpreter (McGregor & McDermott, 1974), maintained and run on the UNIVAC 1106 at the U.C.T. Computer Centre. This results in an effective reduction in speed, so that the compiler cannot be compared with current 'production' models. Plans are underway, however, for the immediate generation of UNIVAC machine codes in the compiler which will increase its speed dramatically, and possibly allow for interfacing with other routines.

The Pascal compiler at U.C.T., written in Pascal, also exists in this stack code form, and is assembled and interpreted by the same interpreter to produce the PL/UCT compiler.

While this may be seen as a disadvantage, especially when it is compared with other high-level language compilers, it leads to the advantageous effects of

a Portability.

There are intentions for an interpreter suitable for the PDP 11/20 in the Department of Computer Science, U.C.T. (although this is attested by a quote in SIGPLAN NOTICES (Holt, 1972), : 'It is well nigh impossible to build a full PL/I compiler to run on a minicomputer, e.g. a PDP/11.') The conversion to the new machine should require few, if any changes to the actual compiler.

b Easier Debugging.

> 'The interpretive mode is a boon to the debugging aid designer, who now has tools available which he has never had before.' (Glass, 1968).

In addition, a major benefit of the structure of the compiler is the ease with which modules, e.g. the code generator, can be modified or replaced. Thus, provided that the same addressing philosophies are embedded in two machines, little effort need be expended in converting the compiler to generate code for the new machine.

The thesis itself comprises three main sections. Section 2 constitutes the main body of the thesis. In sub-section 2.1. the language subset of PL/I is defined and the implementation language, Pascal, chosen - mainly for its closeness to PL/I and its powerful features.

The structure of the compiler, which has a direct bearing on its behaviour, is detailed fully in sub-section 2.2. A description of each aspect of the compiler, including lexical analysis, the symbol table and data structures, and the statement processors, is given here.

The section on the implementation of the compiler describes how it was produced in practice on the U.C.T. Computer Centre's UNIVAC 1106.

Tests on the working compiler show it to be not only a commodity usable by the student community, but also a sound basis for future research.

CHAPTER 2.

THE COMPILER - DESIGN AND IMPLEMENTATION.

b) Pascal variables each have a single data type, although this may be a 'structured' type such as an array or record. PL/I allows multiple attributes for each variable to describe a number of different types of properties such as scale, base, mode. Thus the declarative structure in the two compilers is quite different, and major sections had to be rewritten in order to accommodate the idiosyncrasies of the PL/I language.

The design of the compiler is essentially top-down, being based on the structured programming method, and this concept was carried through the implementation. In fact this approach facilitated the implementation, because, without actually coding every detail from the beginning, the compiler could be tested from an early stage, then refined at each stage until its completion.

2.1. PLAN OF ACTION.

The first task was to define the subset of PL/I that would be implemented, and also the features of Pascal that would be retained.

2.1.1. LANGUAGE SUBSET.

The language of PL/UCT is a basic subset of PL/I containing most data structures and statements. Its grammar is described in a BNF-like notation in Appendix A, but its basic features are set out in the following resumé.

2.1.1.1. SYMBOL SET.

PL/UCT has a vocabulary consisting of basic 'symbols' classified into letters, digits, and special symbols made up from a 63 character set.

The compiler accepts numbers, character strings, and identifiers in free format from the source stream. The usual decimal notation

is used for numbers which are of type FIXED or FLOAT, while character strings are denoted by sequences of characters enclosed by single quotes. These strings are taken by the compiler to be equivalent to character arrays and may be up to 63 characters long. Identifiers may be up to 31 characters in length, are unique in 10, and delimited by blanks, or comments. The set of reserved words is made up from 71 identifiers (including abbreviations), each of which is translated into a special symbol on input to the compiler. Identifiers may denote variables, constants, types, labels, and procedure names.

2.1.1.2. VARIABLES AND ATTRIBUTES.

Variables may be declared FIXED or FLOAT, BINARY or DECIMAL, AUTOMATIC or STATIC, or EXTERNAL, although the attributes DECIMAL, STATIC, and EXTERNAL, are not yet supported. The precision attribute is not yet implemented and in fact values are represented as fullword (36 bits) integer or floating point.

Thus the statement

```
DECLARE A FIXED,
        (B,C) BINARY
        (D)   DECIMAL,
        (I)   FLOAT,
        E,F,M;
```

establishes A as FIXED BINARY, B and C as FLOAT BINARY, D as FLOAT DECIMAL, E and F as FLOAT BINARY, I as FLOAT BINARY, and M as FIXED BINARY.

Variables may also be declared as CHARACTER or character arrays, BOOLEAN, POINTER, LABEL, or SET. Full array facilities exist for all of these, including multiple dimensions, with no limit on the number of these dimensions, and user-defined subscript bounds. For example,

```
DCL  X,Y(100),T FIXED BIN,
        (P,LOC) POINTER,
        SWITCH LABEL;
DCL NAME(-100..100) CHARACTER(30);
```

declares two FIXED BINARY arrays X and Y, each of size 100, T as FIXED BINARY, pointers P and LOC, SWITCH as type LABEL, and array NAME containing 201 character strings of length 30.

Records are defined in the accepted manner. Fields in records may assume any of the attributes mentioned above, as in

```
DECLARE   1 STUDENT,
          2 NAME CHAR(20),
          2 YEAR FIXED BINARY;
```

Where there is no ambiguity a field may be referenced on its own, e.g. YEAR, however, if there is a possibility of confusion, fields must be referenced through the field hierarchy as in STUDENT.NAME.

The INITIAL attribute can be used to dynamically assign initial values to variables of type FIXED, FLOAT, or CHARACTER (array).

Thus

```
DECLARE   I,J,K INIT(0),
          MSG CHARACTER(5) INITIAL ('HELLO');
```

places zero into I, J, and K, and the string 'HELLO' into MSG on first entrance to the block.

A useful feature included in this implementation is the LIKE attribute, with which a variable may assume the attributes of another, as in

```
DCL       1 THAT,
          2 ATTRIBUTE CHAR,
          2 FINGER POINTER;
DECLARE THIS LIKE THAT;
```

Based variables, including records, are declared by the inclusion of the BASED attribute, in which case they may be generated at execution time. The BASED attribute performs an automatic declaration of the pointer, which may not be declared in a separate statement.

For example,

```
DECLARE   1 CELL BASED(CURSOR),
          2 ID CHAR(10),
          2 DESCRIP;
```

allocates space at compilation time only for the pointer CURSOR, the description of the record being stored for use by an ALLOCATE statement.

2.1.1.3. CONSTANTS AND TYPES.

Two extra features, which were retained from the original Pascal compiler but which are incompatible with ordinary PL/I, add to the power of the PL/UCT compiler.

The first is definition of constants, by
CONST <name> =<value> ;
Secondly, the user can define his own data types by writing
TYPE <name> =<type> ;
where <type> may be in the form

(i) [<list of identifiers>]
in which an ordered set of values is defined by enumeration of the identifiers which denote these values.
For example,

```
TYPE WKDAY = [ MON,TUE,WED,THUR,FRI,SAT,SUN ] ;
```

MON is defined as a constant with value 0, TUE with value 1, etc. WKDAY is a scalar type.

(ii) <constant> .. <constant> ;
i.e. subrange, in which the lower and upper bounds of a variable with this type are defined.
For example,

```
TYPE WORKWEEK = MON..FRI;
```

WORKWEEK is a subrange type.
The subrange type can be used to define array bounds, as in

```
DECLARE WAGES(WORKWEEK);
```

(iii) any standard data type.

e.g. TYPE REF = POINTER; INTEGER = FIXED;

enables convenient declaration of variables as

DECLARE P REF, X INTEGER;

The power of user-defined data types, especially in the form (i), is shown when used in conjunction with the data type SET, (also a non-PL/I feature) as in the following example which manipulates values corresponding to week-days in the year.

```
DECLARE  WEEK(1..52) WORKWEEK,
         TODAY,YESTERDAY WKDAY,
         WORKDAYS SET OF WKDAY,
         CURRENT 1..52;
```

Operators applicable to SET types are
OR (union) , AND (intersection) , - (set difference) , and IN (membership) , so that we can have

```
IF TODAY IN WORKDAYS THEN .....  ,
WEEK(CURRENT)   = TODAY ;
WAGES(YESTERDAY)= 0.0   ;
```

.... etc.

2.1.1.4. PROCEDURES.

Internal procedures and functions may be defined within the main program and procedures may be defined within procedures. Procedure definitions may be nested up to a maximum level of 10. A present PL/UCT restriction is that a definition must occur at the beginning of a block. All procedures must be declared in the outer block in which they occur and their definition also limits the scope of all identifiers declared within them. Parameters are called by reference.

In addition to user-defined procedures, a number of standard procedures and builtin functions make up the PL/UCT library. These are shown in Table 2.1., although a full description of each is given in Appendix B.

STANDARD PROCEDURES	BUILTIN FUNCTIONS
PACK	ABS
UNPACK	TRUNC
MARK	ORD
RELEASE	CHR
	PRED
	SUCC
	SIGN
	MIN
	MAX
	SUM
	TIME
	SIN
	COS
	EXP
	SQRT
	LOG

TABLE 2.1. STANDARD PROCEDURES AND BUILTIN FUNCTIONS.

Standard procedures and builtin functions need not be declared.

2.1.1.5. FILES

In addition to the standard input and output files SYSIN and SYSPRINT, PL/UCT allows declarations of files with fixed record lengths of 80 characters.

Thus

```
DECLARE WORK FILE ;
```

sets aside space on the stack for the buffer associated with the file.

At present, files may be used as input, or output, but not input-output. The maximum number of files that may be declared is 18.

2.1.1.6. STATEMENTS.

The following belong to the PL/UCT statement repertoire. A fuller description in BNF notation is tabled in Appendix A.

```
ALLOCATE <based variable> [SET(<pointer>)] ;
ASSIGNMENT (including multiple assignment)
COMPOUNDSTATEMENTS (with or without declarations)
[CALL] <procedure-name> (keyword CALL is optional)
CONST
DECLARE
DO <index> = <start> [TO<end>] [ BY <increment>][WHILE <expression>]
                                                <statements>END ;
DO <index> = <item> , <item> [WHILE <expression>] <statements> END ;
DO WHILE <expression> ; <statements> END
DO <statements> END ;
GET [ FILE( <filename> ) ] LIST( <list of variables> );
GO TO <label constant> / <label variable> (abbr. GOTO)
IF <expression> THEN <statement> ; [ ELSE <statement> ; ]
ON ENDFILE <statement> ;
<entry-name> : PROCEDURE [ ( <parameterlist> )] [RETURNS( <attribute> )] ;
PUT [SKIP] [ FILE( <filename> ) ] LIST( <list of expressions> ) [SKIP] ;
    [PAGE]                                                     [PAGE]
RETURN <expression>
STOP
TYPE
```

A label may be associated with any executable statement in the form < label > : .

All the above statements are implemented, and in addition, three Pascal statements are included.

```
CASE < case body > END ;
LOOP < statements > EXIT IF < expression > ; < statements > END ;
REPEAT < statements > UNTIL < expression > ;
```

Thus, although PL/UCT as a subset of PL/I can be said to be completely compatible in that subset, the ability to define constants and data types, together with the extra facilities of the CASE, LOOP, and REPEAT statements, combine to make PL/UCT a unique language and powerful force in teaching.

2.1.2. IMPLEMENTATION LANGUAGE.

The next step to consider in the design was the implementation language.

Although the Pascal compiler available at U.C.T. was itself interpreted as it existed only in a 'hypothetical stack code' form, the Pascal language was chosen for two main reasons:

Firstly, the 'skeleton', on which the PL/UCT compiler was based, was written in Pascal and, secondly, the Pascal language was most conducive to the method of structured programming used in the design. In retrospect, it was this method of design that contributed most to the quick implementation of the compiler; and, as it turned out, in spite of the obvious disadvantages of speed of interpretation, the interpreter was a boon in that it provided tools for debugging that would have been otherwise unavailable.

2.2. STRUCTURE OF THE COMPILER.

The simplicity of the structured programming approach can be seen in the direct association between the compiler and its definition in BNF-notation, in which every major nonterminal in the definition is associated with a procedure in the compiler. Thus, any reference to a major nonterminal in the definition results in a procedure call in the source text of the compiler. This greatly affects the compiler's behaviour in that the states of the compiler, and the links between them, are directly attributable to the syntactic constructs in the program being compiled. As a result of this, the compiler probably fulfils conditions of program-correctness, although this has yet to be proved.

The method of design of the compiler, using the top-down approach, allowed the development to proceed in stages, each stage corresponding to the completion of an 'outer' procedure which improved the capabilities of the compiler, enriched the language subset which was its object, and allowed a 'test-run' of the new procedure before its 'inner parts' had been completed. The important point is that the compiler was maintained as a 'whole' from start to finish, although, of course, it was a bigger 'whole' on its completion.

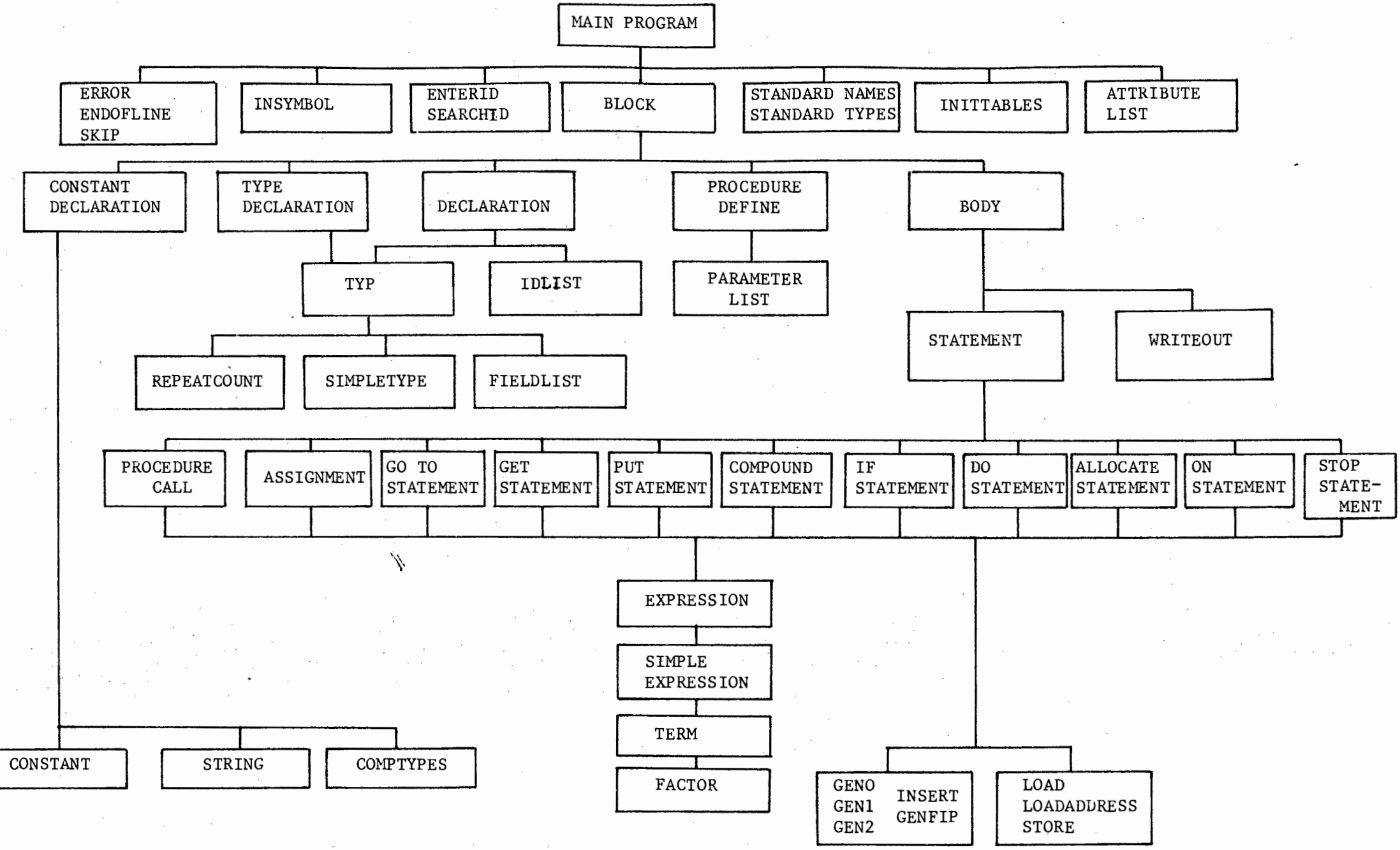


FIGURE 2.2. STRUCTURE OF THE COMPILER

Figure 2.2. shows the basic structure of the compiler. In the structure, each procedure is shown directly under the procedures which call it. The definition of every procedure is made as local as possible.

Each level in the hierarchical structure in Figure 2.2. corresponds roughly with a new stage of development, starting with the top block 'MAIN PROGRAM'.

Figure 2.3. shows how the procedures are joined to form the structured executable program. In the figure, each procedure is indented according to its (static) declaration level, thus defining the scope and flow of data from procedure to procedure, as well as the flow of control from higher to lower level.

```
MAIN PROGRAM
     ENDOFLINE
          DIAGNOSTIC
     ERROR
     INSYMBOL
          NEXTCH
          OPTIONS
     ENTERID
     SEARCHSECTION
     SEARCHID
     GETBOUNDS
     ATTRIBUTELIST
          MARK
               MARKSTP
               MARKIDP
          FOLLOWSTP
          FOLLOWIDP
     BLOCK
          BEGBLOCK
          ENDBLOCK
          SKIP
          ERRSKIP
          CONSTANT
          COMPTYPES
          STRING
          TYP
               REPEATCOUNT
               SIMPLETYPE
               FIELDLIST

0    1    2    3                    DECLARATION LEVEL
```

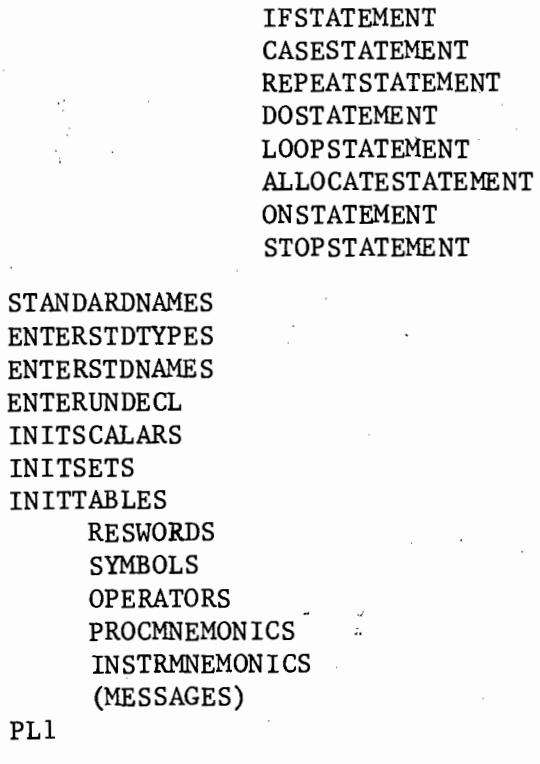


FIGURE 2.3. PROCEDURE NESTING

The figure corresponds exactly to the final program which is listed in Appendix C.

At the top of the hierarchy is the main program which contains the lexical analyser in procedure INSYMBOL, and the routine to process a PL/I block in procedure BLOCK.

2.2.1. LEXICAL ANALYSIS.

The function of INSYMBOL is to extract, at each call, the next symbol from the string of input characters forming the PL/I program, where a symbol might be a constant, an identifier, an operator, a reserved word, or another delimiter. This is defined at the beginning of the compiler in the user-defined type

```
SYMBOL = (IDENT, FIXCONST, FLTCONST, STRINGCONST, NOTSY,
          MULOP, ADDOP, RELOP, LPARENT, RPARENT, LBRACK,
          RBRACK, COMMA, SEMICOLON, PERIOD, ARROW, COLON,
          BECOMES, DECLARESY, CONSTSY, TYPESY, ALLOCATESY,
          BEGINSY, CALLSY, CASESY, CHECKSY, DOSY, ENTRYSY,
          PROCEDURESY, FORMATSY, FREESY, GETSY, GOTOSY, IFSY,
          LOOPSY, ONSY, PUTSY, READSY, REPEATSY, RETURNSY,
          STOPSY, WRITESY, ENDSY, POINTERSY, LABELSY, CHARSY,
          BITSY, FILESY, RECORDSY, SETSY, BINARYSY, DECIMALSY,
          FIXEDSY, FLOATSY, BASEDSY, LIKESY, INITIALSY, TOSY,
          BYSY, WHILESY, OFSY, UNTILSY, EXITSY, THENSY,
          ELSESY, PAGESY, LINESY, SKIPSY, LISTSY, EDITSY,
          RETURNSSY, OPTIONSY, EXTERNSY, FROMSY, INTOSY,
          AUTOSY, STATICSY, CATSY, NOSY, OTHERSY, ENDPROG) ;
```

INSYMBOL also prints a copy of the source program through prodecure ENDOFLINE.

INSYMBOL is in fact a finite-state machine which produces as its output the seven global variables

(1) the last symbol SY (of type SYMBOL)
(2) classification of the last symbol OP (operator, etc.)
(3) value of the last constant (fixed, float, or string)
(4) length of last string constant
(5) number of characters in the last identifier
(6) the last identifier
(7) the last character read.

INSYMBOL is used by most procedures involved in syntactic analysis to produce the next symbol before exiting, or before entering a lower-level procedure. That is, each procedure is 'synchronised' as an LR(1) processor.

2.2.2. THE SYMBOL TABLE.

Since a block may be defined within a block, procedure BLOCK is one of the few that is called recursively. Each call on BLOCK (performed when the compiler finds a new procedure-definition) results in the static declaration level being incremented by 1, and a fresh empty binary tree made available in the symbol table.

The symbol table is simply an array of identifier-entry nodes, which at each declaration level is organised into an unbalanced binary tree with an array element (node) as its root.

In this respect, procedures ENTERID and SEARCHID are of interest. Procedure ENTERID enters an IDENTIFIER cell (already generated in the heap) into the binary tree of the current declaration level. Its counterpart, SEARCHID, searches the array of trees from the current level down to find an entry with a given name. Consequently, for two identifiers declared at different levels, the latest, or most local, declaration would be found first. Thus is the scope of variables defined.

2.2.3. THE DEFINITION OF INTERNAL PROCEDURES.

The definition of a procedure is treated in PL/I as a statement within a block. However, in deciding on the status of procedure PROCEDUREDEFINE which handles this, a slight difficulty arises in that the main program - or outermost procedure - requires to be handled in the same way. This problem, which is emphasized by the fact that PROCEDUREDEFINE is only called once by procedure BLOCK - for the main program - and thereafter calls BLOCK for each definition, was solved by the implicit definition of a MASTER procedure which calls the main program. Thus the MASTER procedure, which is transparent to the user, is associated with the first call of procedure BLOCK, which then operates on the main procedure as an internal procedure.

The nesting of procedures in the source text does not mean that procedures in generated-code form are nested. An array, CODE, is declared local to each block definition and receives code generated for the block which is written to a file at the end of each block. Thus,

DECLARE < name > (< bounds >) < attributes > ;

Pascal, however, permits variable-length records to be generated at runtime, whose lengths depend on the values of so-called 'tagfields', while the PL/I record, like its fixed-length Cobol version, provides no such facility. Procedure FIELDLIST, which is fully recursive, reflects the simplicity with which this kind of (hierarchical) construct - that is, the PL/I record - can be parsed using the structured method.

In the same class is procedure IDLIST which handles lists of variables. Lists of variables may be symmetrically enclosed between any number of left and right parentheses, thus representing a context-free grammar, without this affecting the complexity, and resultant behaviour of the compiler to any great extent. This is not in itself an outstanding feature; however IDLIST, with some restructuring, has the potential to cater fully for factoring of attributes without adding much to the compiler's overheads in handling declarations.

The structures of procedures DECLARATION, IDLIST, TYP (which processes attributes), SIMPLETYPE, and FIELDLIST, are shown in the syntax diagrams in Figures 2.4. through 2.8. respectively.

So far, no mention has been made of how the compiler holds information about its data. This information is held in several different types of 'cells', implemented in Pascal records, the type depending on whether the cell contains the identity of a variable, field, or procedure, a description of a particular structure, or the address of a label, etc. The concept of a cell to hold information is not new nor peculiar to this compiler; however, the methods used by the compiler in building tables, and in particular the linking of cells, are of special interest.

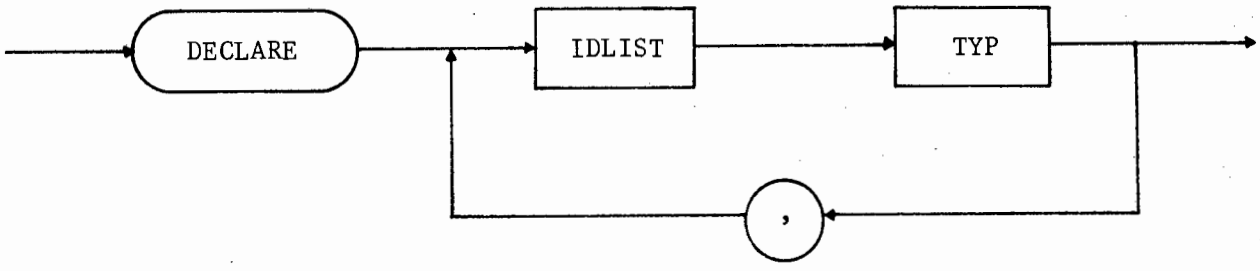


FIGURE 2.4. DECLARATION

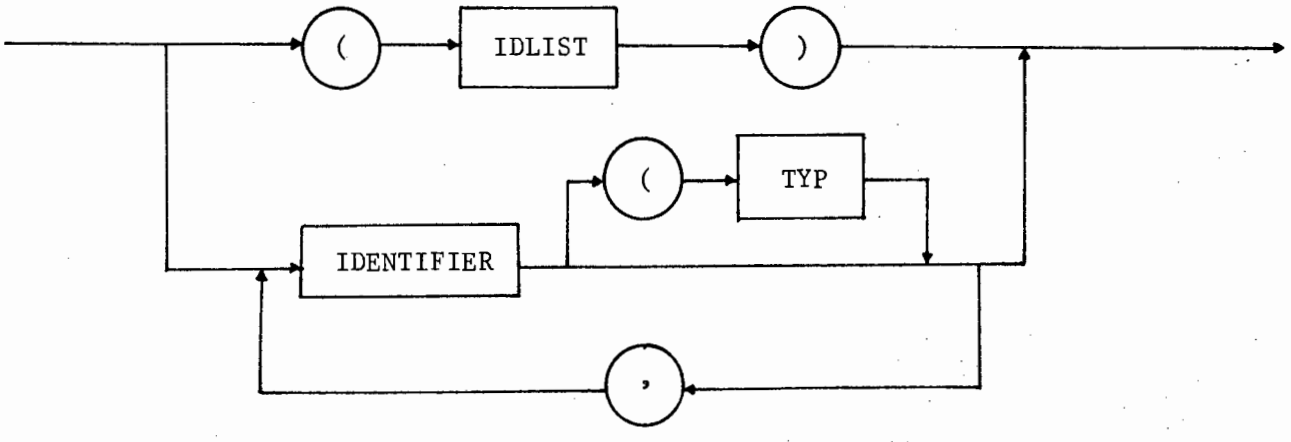


FIGURE 2.5. IDLIST

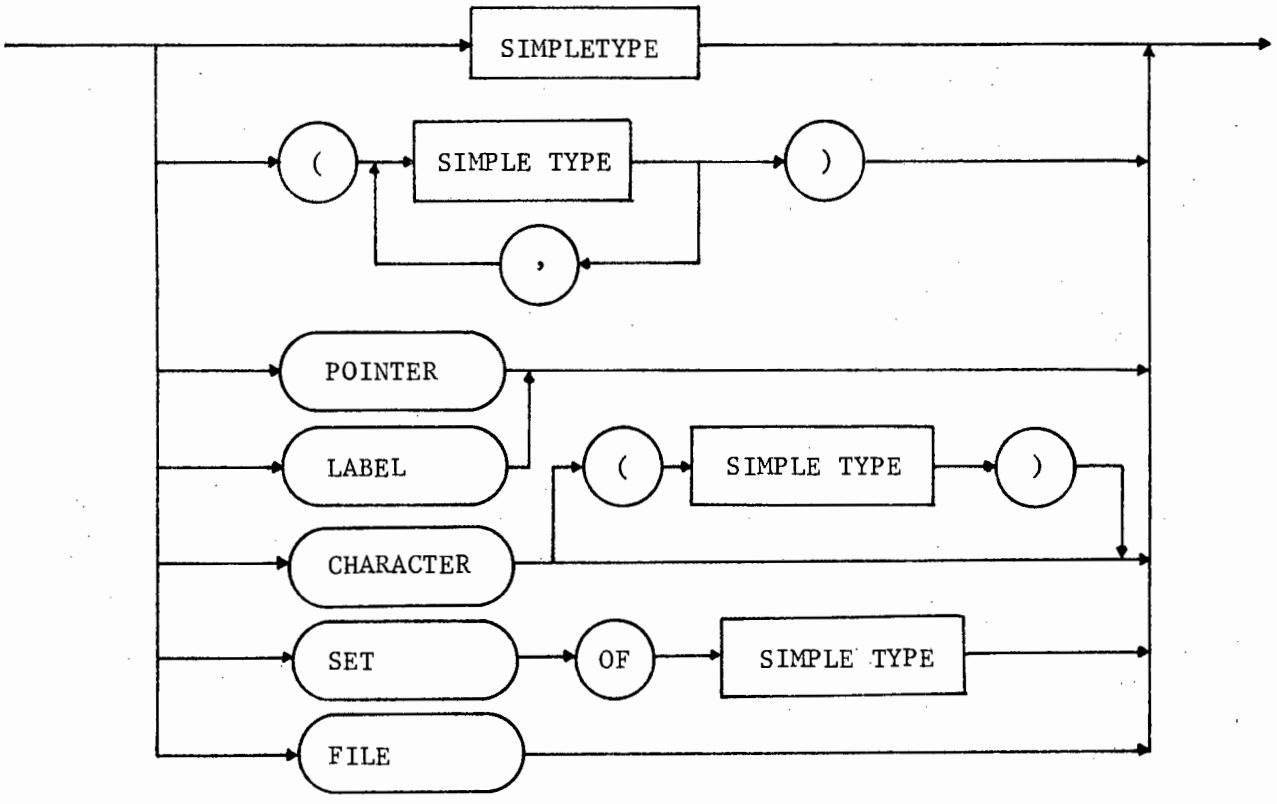


FIGURE 2.6. TYP

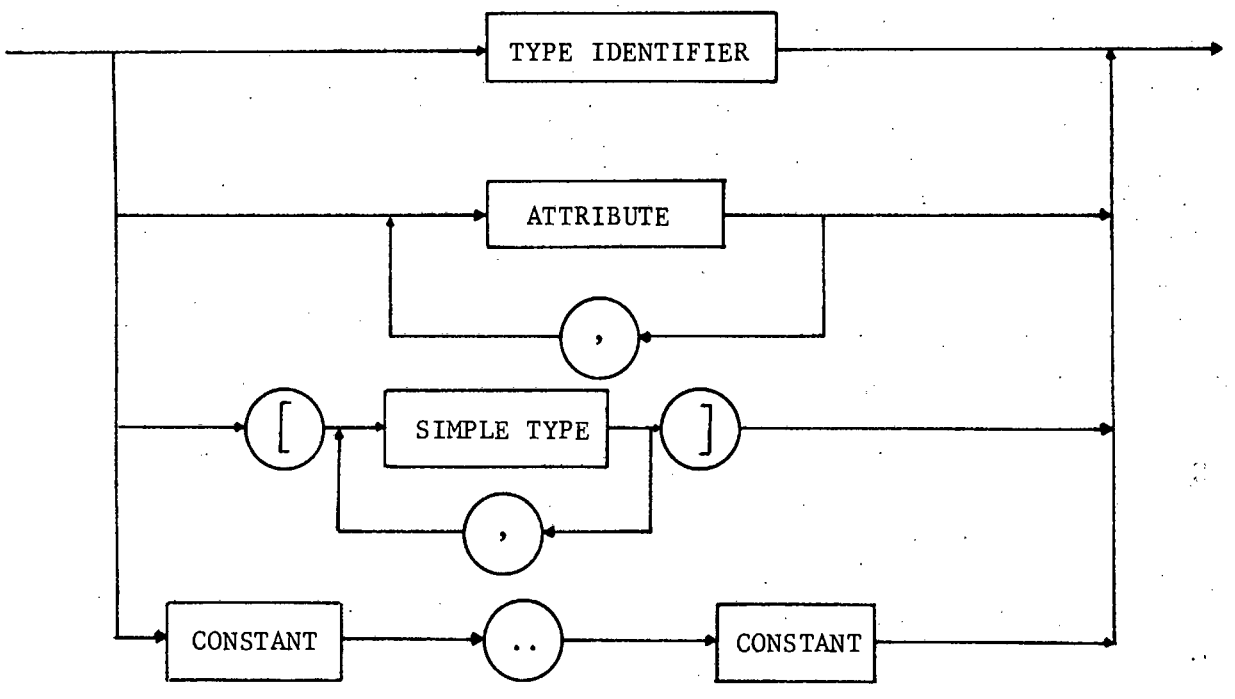


FIGURE 2.7. SIMPLETYPE

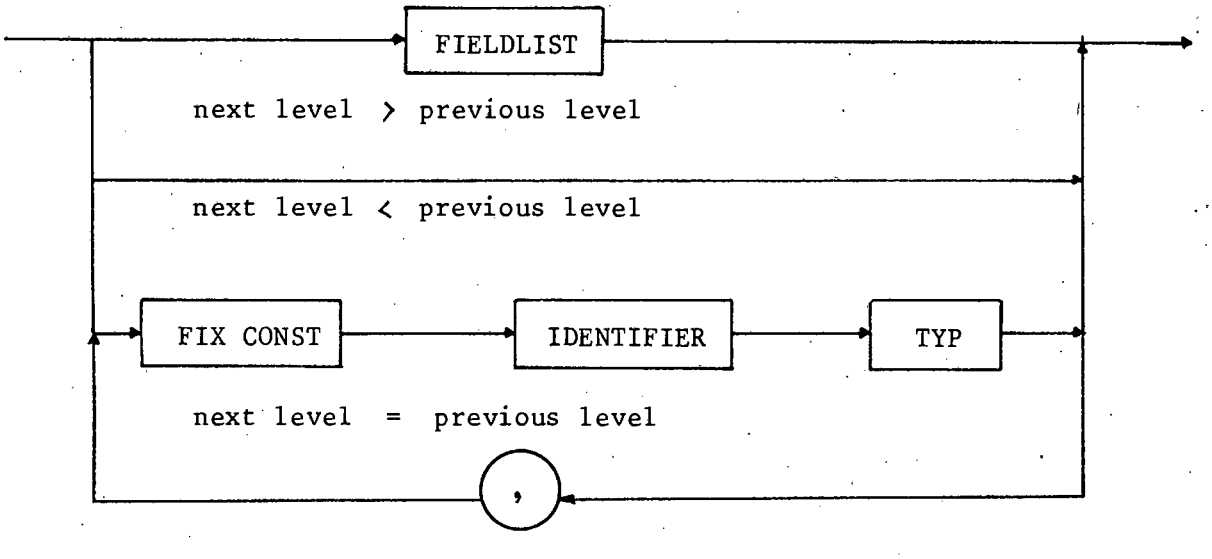


FIGURE 2.8. FIELDLIST

These cells, being generated, naturally, at execution time, are pooled in an area called the 'heap'. Different types of cells are generated depending on whether the cell is an IDENTIFIER entry record linked into the symbol table, or a STRUCTURE record which reflects one of the basic data structuring facilities mentioned earlier.

Identifier information is held in a cell of user-defined type IDENTIFIER in the following Pascal record structure:

```
| NAME   | NEXT   |
| LLINK  | RLINK  |
| IDTYPE | BASE   |
case CLASS of
    TYPES    case BASED of TRUE : POINTERID
    KONST    VALUE
    VARS     KIND , LEVEL , ADDRESS
    FIELD    LEVEL , RELATIVE ADDRESS
    PROC     case KIND of
    FUNC        STANDARD : KEY
                DECLARED : LEVEL , ADDRESS ,
                   case  IDKIND of
                   ACTUAL : FORWARDDECLARED, EXTERNAL.
```

where

NEXT	may contain a pointer to another IDENTIFIER cell if a linking other than that used in the symbol table is needed, such as in a list of parameters.
LLINK,RLINK	point to IDENTIFIER cells to the left and right, respectively, in the symbol table.
BASE	indicates BINARY or DECIMAL.
IDTYPE	is a pointer which indicates the particular data structure.
CLASS	indicates whether the identifier is a user-defined type, a based variable, a constant, variable, field, or procedure-name. CLASS is a so-called tagfield, which, by its value, determines the length of the IDENTIFIER record.

The field IDTYPE is a pointer to a cell of type STRUCTURE, which is made up of the following fields.

```
SIZE
case FORM of
     LABEL
     SCALAR    case KIND of DECLARED : FIRST CONSTANT
     SUBRANGE  RANGETYPE, MIN, MAX
     POINTER   ELEMENT TYPE
     SET       ELEMENT SET
     ARRAY     ELEMENT TYPE, INDEX TYPE
     RECORD
     FILE      FILE NUMBER, END-OF-FILE CONTINGENCY ADDRESS
```

where

FORM	describes the form that the variable may take, and in cases such as POINTER, ARRAY, the cell may contain pointers to other cells so that a structure may be defined in toto by a series of linked cells, each describing an aspect of the whole structure. This method is very flexible in that it can allow, for example, an array of records or a record of arrays, depending on how the cells are linked.
SIZE	gives the total size of the structure.

Note that while there is one IDENTIFIER cell for each identifier, identifiers may share STRUCTURE cells, and this is especially so in the case of simple types like FIXED and FLOAT.

Figure 2.9. illustrates a typical set of structures that would be created and linked together for the following compound declaration:-

```
DECLARE  ONE,TWO BINARY FIXED,
         THREE,FOUR FLOAT,
         FIVE(-5..5) CHARACTER(10);
```

The links through pointer NEXT, between TWO and ONE, and between FOUR and THREE, are similar. FOUR's NEXT points to THREE, while THREE's NEXT contains the NULL pointer. This linkage is due to the action of procedure IDLIST

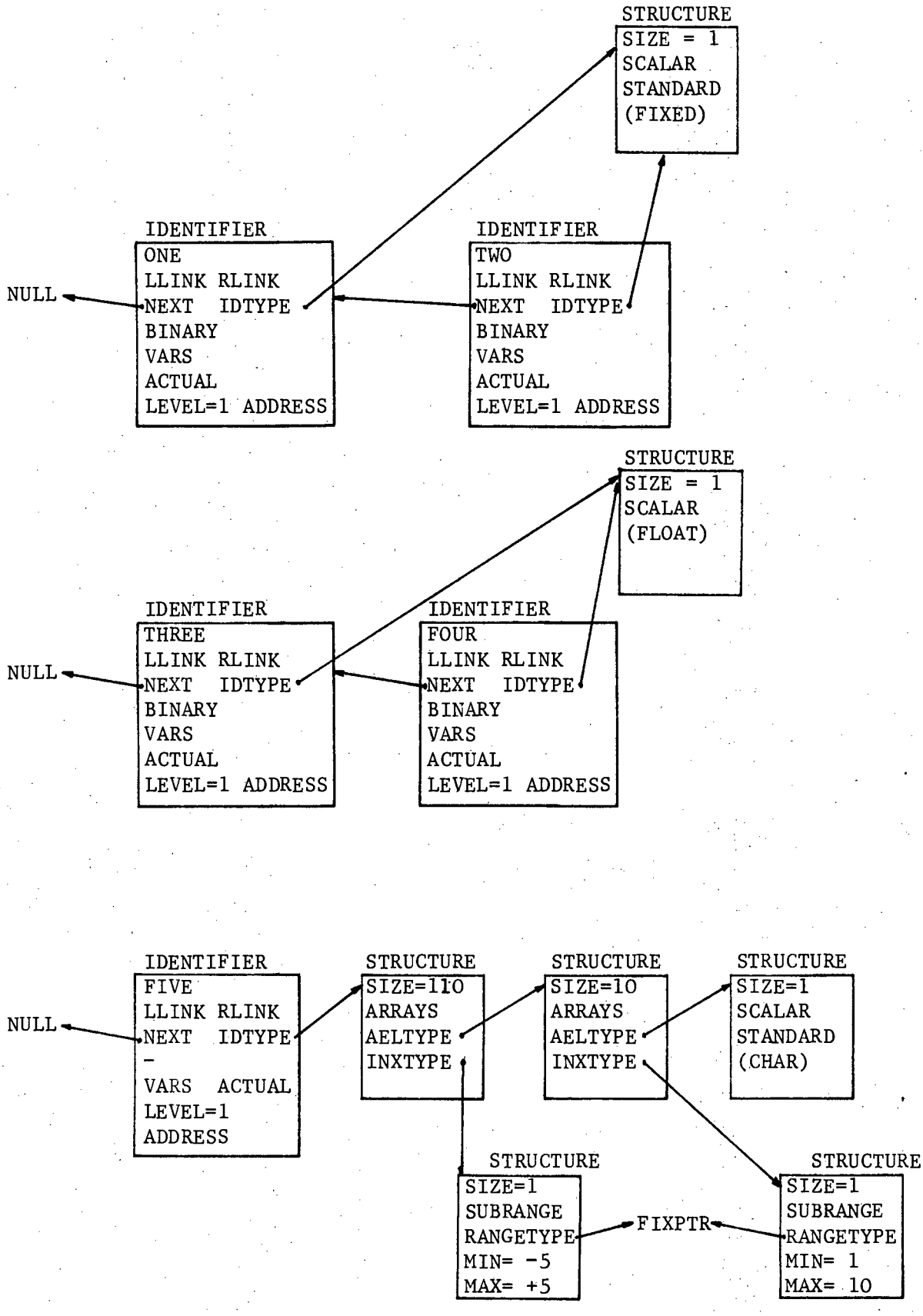


FIGURE 2.9. IDENTIFIER AND STRUCTURE CELLS

[CREATED FOR THE STATEMENT:

```
DECLARE  ONE,TWO  BINARY FIXED,
         THREE,FOUR  FLOAT,
         FIVE(-5..5)  CHARACTER(10) ; ]
```

which operates on lists of variables, and as shown in the figure, all the variables linked in a list share a common STRUCTURE record - the STRUCTURE cell for FIXED in the first case, FLOAT in the second. In the array cell-structure, FIVE's IDTYPE points to a list of STRUCTURE cells of form ARRAY, one for each dimension, plus one for the array type. Each ARRAY STRUCTURE-cell points to the index-type, e.g. integer or subrange, in this case a subrange of -5..+5, and the element type. Thus, in the example, the element type for the first STRUCTURE-cell is itself an array of one dimension, whose element type is CHARACTER.

The IDENTIFIER cell for each of the variables above contains the declaration level, the base address of the variable, and a FORMAL/ACTUAL indicator, in addition to the left and right links for the symbol table.

The record structure example shown in Figure 2.10 for the typical record declaration

```
DECLARE  1 WORKER,
           3 REFERENCE CHAR(6),
           3 NAME,
            5 FIRST CHAR(10),
            5 LAST  CHAR(15),
           3 PAYRATE FLOAT ;
```

links IDENTIFIER cells for each field, which are entered into the symbol table in the normal way, only by the pointer NEXT, which points to the field one up in the hierarchy. Fields highest up in the hierarchy point to the IDENTIFIER cell for the record, in this case WORKER. IDTYPE for each field points to a cell-structure in the normal way, while WORKER's IDTYPE points to a STRUCTURE cell of form RECORD. Thus field-names are only duplicated if their respective NEXT's point to the same location, allowing fields with the same name to be declared in different records. Through these NEXT pointers, the compiler can trace any reference

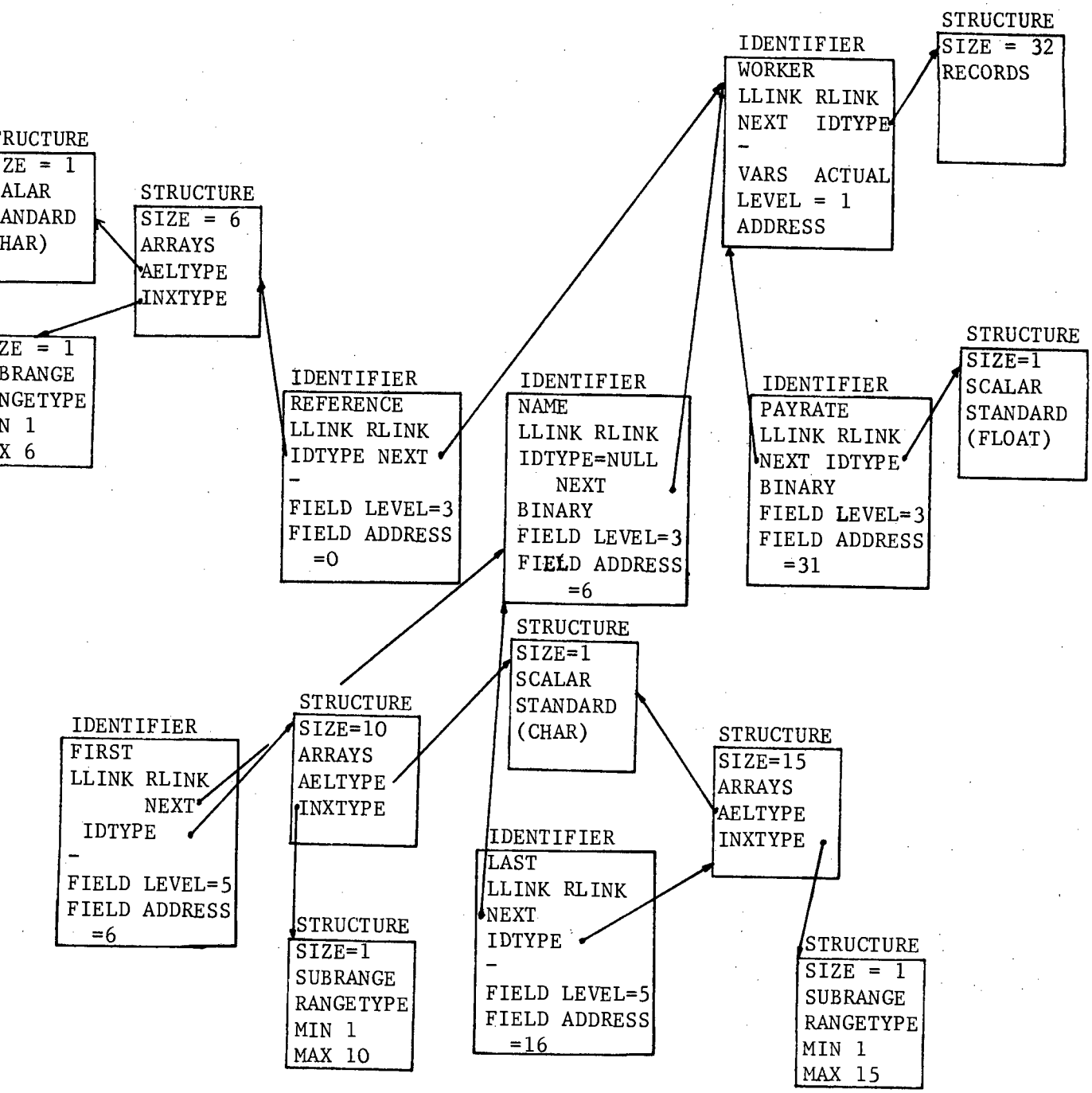


FIGURE 2.10 CELL STRUCTURE FOR A RECORD

[FROM THE DECLARATION:

```
DECLARE   1   WORKER,
              3   REFERENCE CHAR(6),
              3   NAME,
                5  FIRST CHAR(10),
                5  LAST  CHAR(15),
              3   PAYRATE FLOAT;  ]
```

to a fieldname to its record cell, so that qualification of fieldnames is not always necessary.

IDENTIFIER and STRUCTURE cells are not the only ones used by the compiler. The INIT cell holds initial values for variables declared with the INITIAL attribute. As PL/UCT deals only with AUTOMATIC variables at present, initial values have to be generated dynamically on each entry to a block, and this is done by procedure BODY. This generation of initial values is performed before the statements of the block are executed. The method used by the compiler is to build a chain of these cells at each declaration level, and each contains enough information for the values to be generated, in the form:

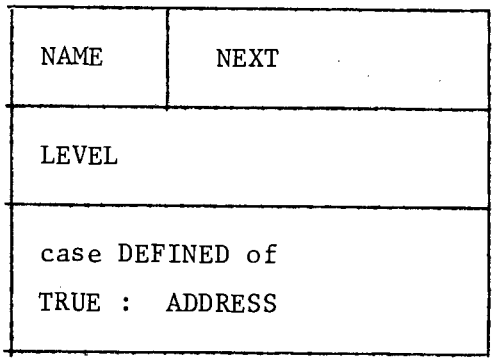
NEXT	TYPE
ADDRESS	REPEATFACTOR
VALUE	unused

where

NEXT	points to the next cell
TYPE	indicates FIXED, FLOAT, or CHARACTER(array)
REPEATFACTOR	instructs that a number of addresses are to contain the initial value
VALUE	is FIXED, FLOAT, CHARACTER(array) corresponding to TYPE.

Cells are not only used for variables and their attributes. For each label that the compiler encounters, and also at

each GOTO statement which references a yet-undefined label, a new LABEL cell is generated and appended to the label chain. Each LABEL cell contains

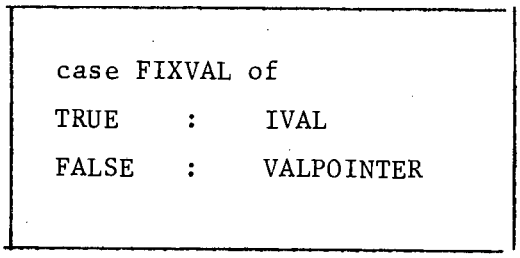


where

NEXT	points to the previous LABEL cell
LEVEL	contains the declaration level
ADDRESS	holds the statement address if DEFINED = TRUE.

Two further cells are used to contain any constants that may appear in a program. Cell VALU may contain either an actual value, if the constant is FIXED, or, if the value is a FLOAT constant, set, or character string, a pointer to another cell CONSTANT.

Thus VALU looks like



VALPOINTER points to a CONSTANT cell of the form

```
case CCLASS of
 FLOAT   :   RVAL
 SET     :   SETVAL
 STRING  :   LENGTH , SVAL : ARRAY OF CHAR
```

The last type of cell is one which is defined in the main program and whose function is to hold global information about the current expression being compiled. One or two other 'copies' of this cell DESC have application in procedures which temporarily store local 'expression' information for their own use, but basically, the one-off cell DESC contains

```
TYPTR
case KIND of
  CST    :      VALUE
 VARBL   :   case ACCESS of
                DIRECT    :   LEVEL , ADDRESS
               INDIRECT   :   DISPLACEMENT
                INDEXED
EXPRESSION
```

where

TYPTR	holds the IDTYPE of the current IDENTIFIER cell,
KIND	indicates whether the expression is a constant, variable, or other value.
ACCESS	indicates the access method in the case of a variable.

A short digression may be appropriate here. The use of cells is one way of building up compiler tables dynamically, and each cell is merely an entry in a specific table, although space for each table is not allocated exclusively beforehand. The concept of cells is heavily relied upon in the DITRAN compiler, (Moulton & Muller, 1967, Doig, 1970), where their use is extended to that of STATEMENT cells. The STATEMENT cell, which may hold information such as the statement type, number, declaration level, pointer to the previous cell, pointer to a list of identifiers referenced in the statement, referenced-flag, usage frequency (for execution time), could have two functions. On the one hand it would be most useful at execution time, where, used in conjunction with the error routines, it could assist in generating messages in source-linked terms. Also, statistical profiles could be compiled, by a run-time support system, on statement usage and most frequently occurring errors.

Secondly, the STATEMENT cell, by linking a series of 'expression' or other cells, could represent the syntax structure of a statement, and moreover, with the addition of a forward pointer to link STATEMENT cells, a program could be represented as a directed graph, which would be of tremendous assistance in the semantic and flow analyses of the program, and therefore in the optimisation of object code.

In the same way as the compiler holds information about the current expression being compiled, the execution-time support system could use a cell, perhaps implicitly, in the stack, for the statement currently being executed.

Of course the overheads in maintaining such a system would have to be balanced against the benefits that it would reap, but this is not a subject for discussion here. Suffice it to say that the implementation of these ideas is possible, and may be very-much needed in a teaching-oriented compiler of this sort.

Although no STATEMENT cells are used, the structure of the statement processors would allow easy addition of this feature to the compiler. In fact, a series of linked cells representing the syntactic construct of each statement would correspond to the flow of the compiler in syntactic and semantic analyses, and a dump of these cells after compilation of a program would present us with a complete history of the process.

2.2.5. THE BODY PART.

Procedure BODY follows procedure DECLARATION in the definition of each block, and is responsible for management of the statement drivers and the writing out of code. Procedure STATEMENT, a finite-state machine which drives the statement processors and caters for the declaration of labels, is called by procedure BODY.

As each statement in PL/I denotes an action and statements are mutually exclusive (although statements may contain statements), they are handled by self-contained modular drivers. PL/UCT could therefore be enhanced by the mere addition of statement drivers without altering the basic structure of the compiler.

Although each statement processor is a separate, self-contained procedure, the compiler, being a one-pass processor, does not follow the procedure of using separate routines to perform syntactic and semantic analyses and code generation. Instead, these are performed contextually within each statement processor, either indirectly via the expression analyzer, or directly through the primitive procedures LOAD, LOADADDRESS, STORE, and GENFJP. Figure 2.11 shows the parallel nature of syntactic semantic analysis and code generation 'phases', which at all times are accompanied by the symbol table and related cells.

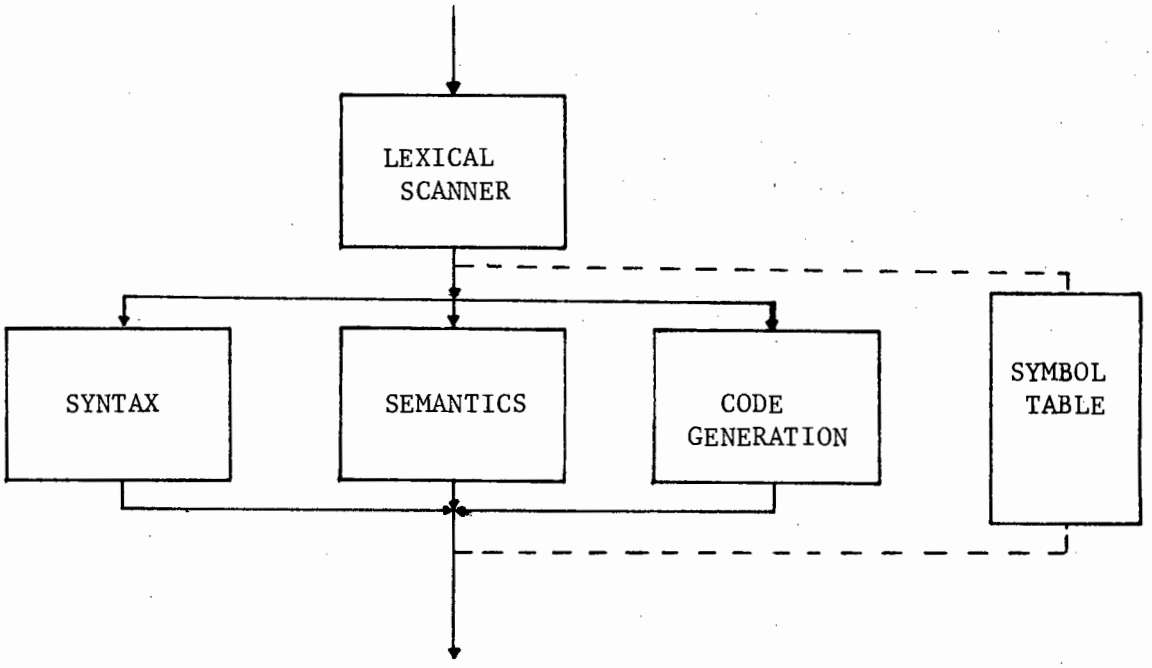


FIGURE 2.11. FLOW OF COMPILER PROCESSES

One consequence of this structure is that the source text of a user-program governs, completely, the order in which code is generated. This greatly affects the compilation process in that local optimisation, both peephole and sub-expression, is made difficult, if not impossible to accomplish. However, code optimisation was not included in the current implementation of the compiler, and so will not be discussed further.

Since syntactic and semantic analyses and code generation are common to all statement processors, a discussion of the principles underlying their use, and the philosophy of the compiler with regard to treatment of errors is necessary before we continue with a description of each procedure.

2.2.5.1. ANALYSIS OF ERRORS.

Error detection and recovery is accomplished in two distinct spheres - by recognition of context-free or context-sensitive constructs. The compiler performs the former by partitioning possible symbols into two disjoint subsets containing symbols

relevant or irrelevant to the syntactic construct being analyzed. This partitioning is not fixed, however, but is a function of the construct. This is achieved in practice by allowing the set of relevant symbols, FSYS, to be an argument in each procedure call. Each symbol fetched from the input stream is tested for membership of this set, the test usually being made an entry to, and before exit from each procedure, and errors are processed by the ERROR procedure if the test for relevance fails.

The ERROR procedure is also used to record context-sensitive errors, PL/I being a highly context-sensitive language, but in this case their detection relies on testing the compatibility of two STRUCTURE cells.

For example, in procedure ASSIGNMENT, where the construct

⟨target⟩=⟨expression⟩

is analysed, the IDTYPE pointer in the target's IDENTIFIER cell, is compared for compatibility with the pointer TYPTR in the global cell GDESC which describes the expression, and this is done through Boolean function COMPTYPES.

These two mechanisms are shown, for example, in procedure DOSTATEMENT in analyzing the statement

DO WHILE ⟨expression⟩ ; ⟨statements⟩ END ;

```
IF SY = WHILESY THEN
  BEGIN INSYMBOL ; EXPRESSION(FSYS) ;
  IF  ¬COMPTYPES (TYPTR,BOOLPTR) THEN
      ERROR ('TYPE IS NOT BOOLEAN') ;
  IF SY = SEMICOLON THEN INSYMBOL
      ELSE ERROR ('MISSING SEMICOLON') ;
  STATEMENT(FSYS)
  END ;
```

The behaviour of the compiler is different for the two types of errors. Each call of procedure EXPRESSION constructs a part of the parse tree. However, due to the one-pass nature of the compilation, the parse tree is

implicit in the generation of code and there is no intermediate stack. If a semantic error occurs, the compiler generally omits generation of code if the type-incompatibility of two operands is serious enough to warrant such an action. Recovery is difficult, as these context-sensitive errors cannot be corrected by backtracking up the parse tree.

The usual consequence of the compiler detecting a context-free error, however, is that the source text is skipped over until a relevant symbol is found, and progress then continues. This means that great care has to be exercised in choosing a set of relevant symbols, so that

a) the compiler does not lose itself jumping over large sections of program, and that

b) errors are not only treated as punching mistakes overlooked by the overanxious student of PL/I.

Thus the Declaration Part might choose, as its set of relevant symbols, an identifier or expected data attribute, while BODY should be limited to skipping to the next statement, and jumping to the respective statement processor. Future versions could improve on the present method by generating code to branch to a standard execution-time error routine. This would enable at least partial execution of a user's program, although the author is not in complete agreement with the PL/C policy of correcting a user's mistakes, and then continuing as far as possible with execution (Morgan & Wagner, 1971, Conway & Wilcox, 1973). The feeling is that a teaching compiler, rather than encourage laxity in a programmer, should impose a constructive discipline by not only detecting errors, but pointing out a 'suggested correction', in a suitable message, without actually performing the correction.

In the compiler, error messages are handled by procedure ENDOFLINE, which, for each source line, prints a list of error numbers placed

in array ERRLIST by procedure ERROR, up to a maximum of 9 errors per line. Due to present space restrictions, each error message contains only the error number which needs to be located in a directory. Future versions, however, should output a message as well as the number. When this is implemented, error messages, which are grouped accordind to error type and location in the compiler (the two roughly correspond), could include parameters to specify invalid symbols, characters, and names. When statement cells are implemented, they could be used in conjunction with run-time message parameters to make execution diagnostics more meaningful.

The improvement could be made very flexible with a suitable modular design of the error-message handler, which could be altered or replaced when the need for a different diagnostic power arises. This is actually foreseen in the compiler in the existence of procedure DIAGNOSTIC, which at present merely prints out the error number, but whose complexity could be increased to that of a powerful yet sympathetic 'teacher' of PL/I.

2.2.5.2. CODE GENERATION.

The 'hypothetical stack code' generated by the compiler is tabled in Appendix E. Instructions are generated in each statement processor indirectly through procedure EXPRESSION, or directly through procedures LOAD, LOADADDRESS, STORE, and GENFJP, which call GEN0, GEN1, and GEN2 to place instructions in the array CODE. Again there is no segregation of the code-generating process, as shown in the following extension of the DOSTATEMENT segment, where LADDR is a local store of the instruction counter IC, to be used as a jump address, and procedure INSERT inserts an address in a given instruction in array CODE. CIX is the index of array CODE.

```
IF SY = WHILESY THEN
  BEGIN INSYMBOL ; LADDR := IC ;
    EXPRESSION(FSYS) ;
    IF COMPTYPES (TYPTR,BOOLPTR) THEN
        GEN1('FALSE JUMP',0)
    ELSE ERROR('TYPE IS NOT BOOLEAN') ;
    LCIX := CIX ;
    IF SY = SEMICOLON THEN INSYMBOL
    ELSE ERRSKIP('MISSING SEMICOLON') ;
    STATEMENT(FSYS) ;
    GEN1('UNCONDITIONAL JUMP',LADDR) ;
    INSERT(LCIX,IC)      /* INSERT THE ADDRESS OF THE */
  END ;                  /* NEXT INSTRUCTION IN THE FALSE */
                         /* JUMP INSTRUCTION */
```

Note that if TYPTR is not Boolean (BOOLPTR), then the 'FALSE JUMP' instruction is not generated. On the other hand, if a semicolon is missing, the compiler searches for one and then tries to process the statement in the normal way, although it still produces an error message.

In order to load or store the proper value from or into its location, procedures LOAD, LOADADDRESS, and STORE have to discover whether the value is a constant, in which case they must be able to locate its value or a pointer to the value, or a variable, in which case the method of access can be

(i) direct - the variable can be located from
< current minus declaration level, relative address
from base of procedure >

(ii) indirect - the address is calculated from
< address on top of the stack + displacement >

(iii) indexed - the address is the value on the stack.

The compiler uses the global cell GDESC (global descriptor) to hold information about the currently-compiled expression as a utility for the semantic and code-generation processes.

This information is placed in GDESC by procedure EXPRESSION, which analyzes expressions, builds the parse tree (implicitly, in the code), and in doing so performs its own semantic and syntactic analyses. Procedure EXPRESSION is prepared to handle the most complex PL/I expression in any context, even though the form may be restricted by the calling procedure (through FSYS, for example). Consequently, in many cases an expression may be evaluated by the compiler although the offending construct may be flagged in error.

Procedure EXPRESSION is fully recursive and the structure, which contains nested procedures SIMPLEEXPRESSION, TERM, and FACTOR, correspond to the BNF notation for < expression > shown in Appendix A 4. Figures 2.12. through 2.16. show this diagrammatically for procedures EXPRESSION, SIMPLEEXPRESSION, TERM, FACTOR, and SELECTOR respectively.

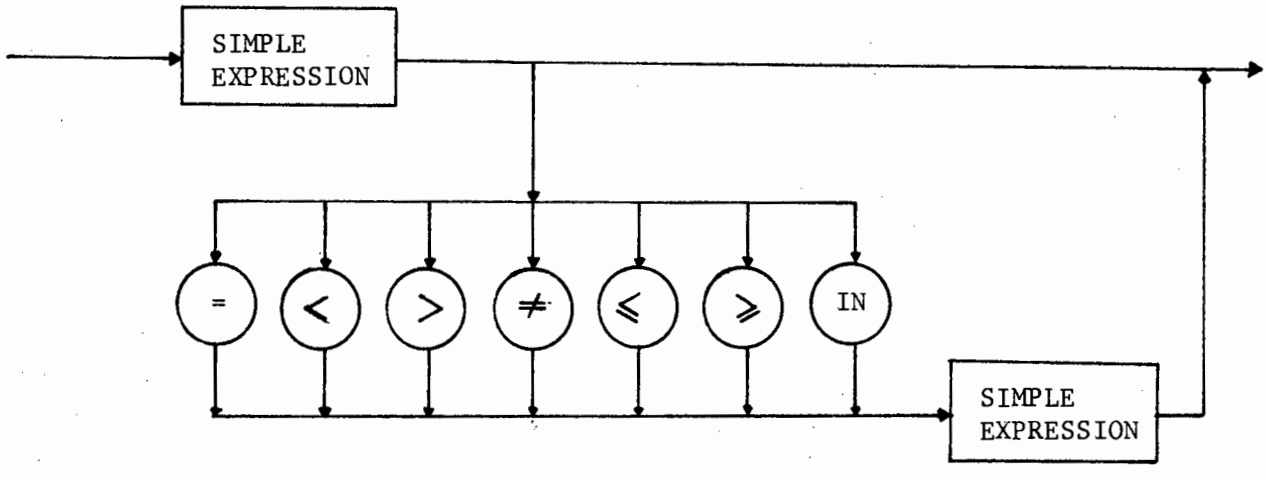


FIGURE 2.12. EXPRESSION

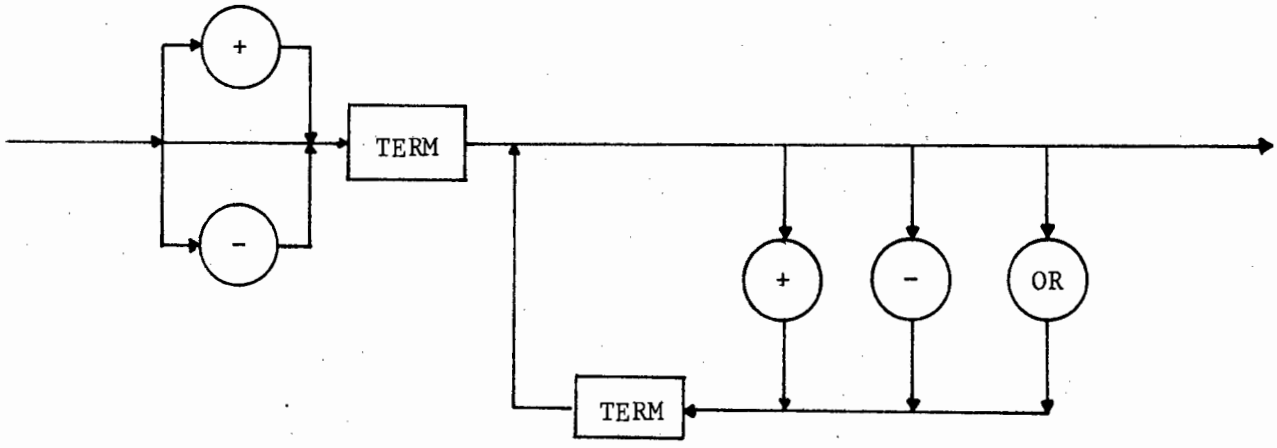


FIGURE 2.13. SIMPLEEXPRESSION

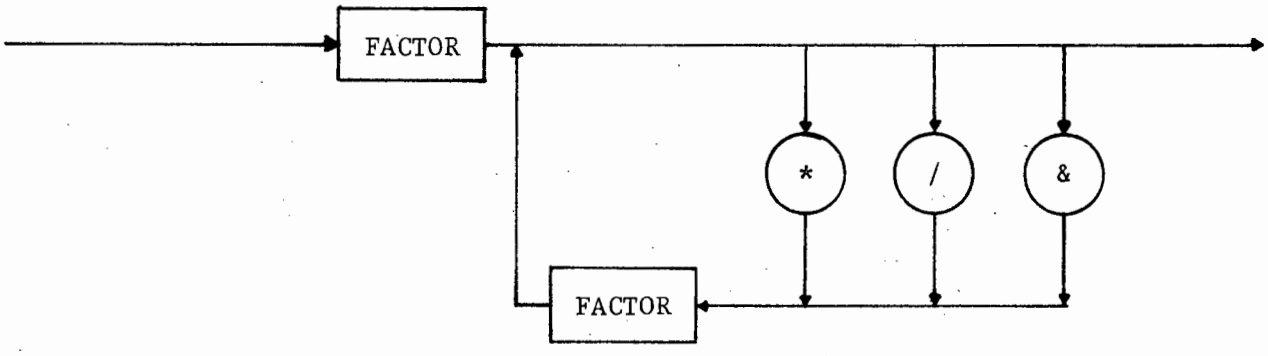


FIGURE 2.14. TERM

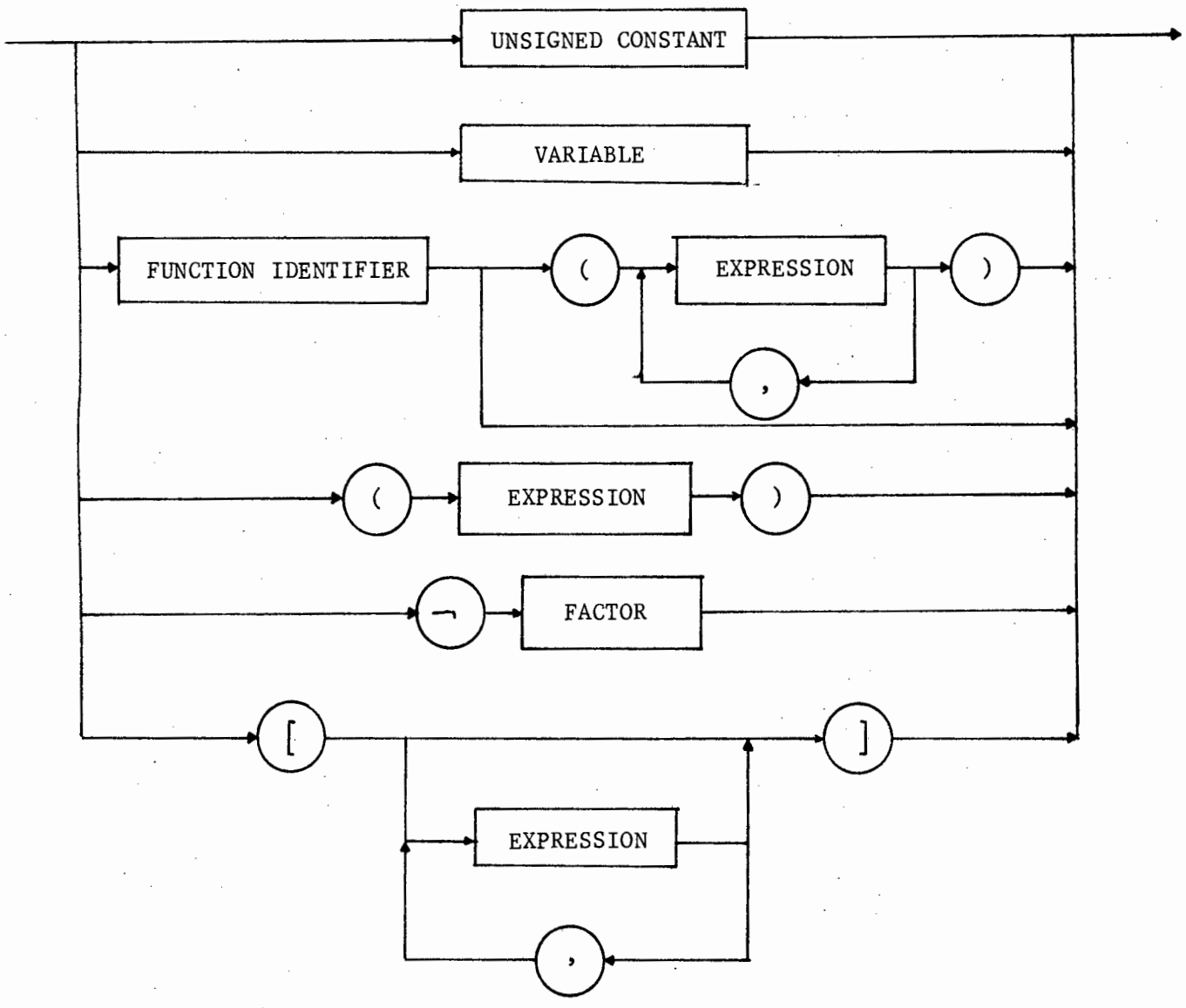


FIGURE 2.15. FACTOR

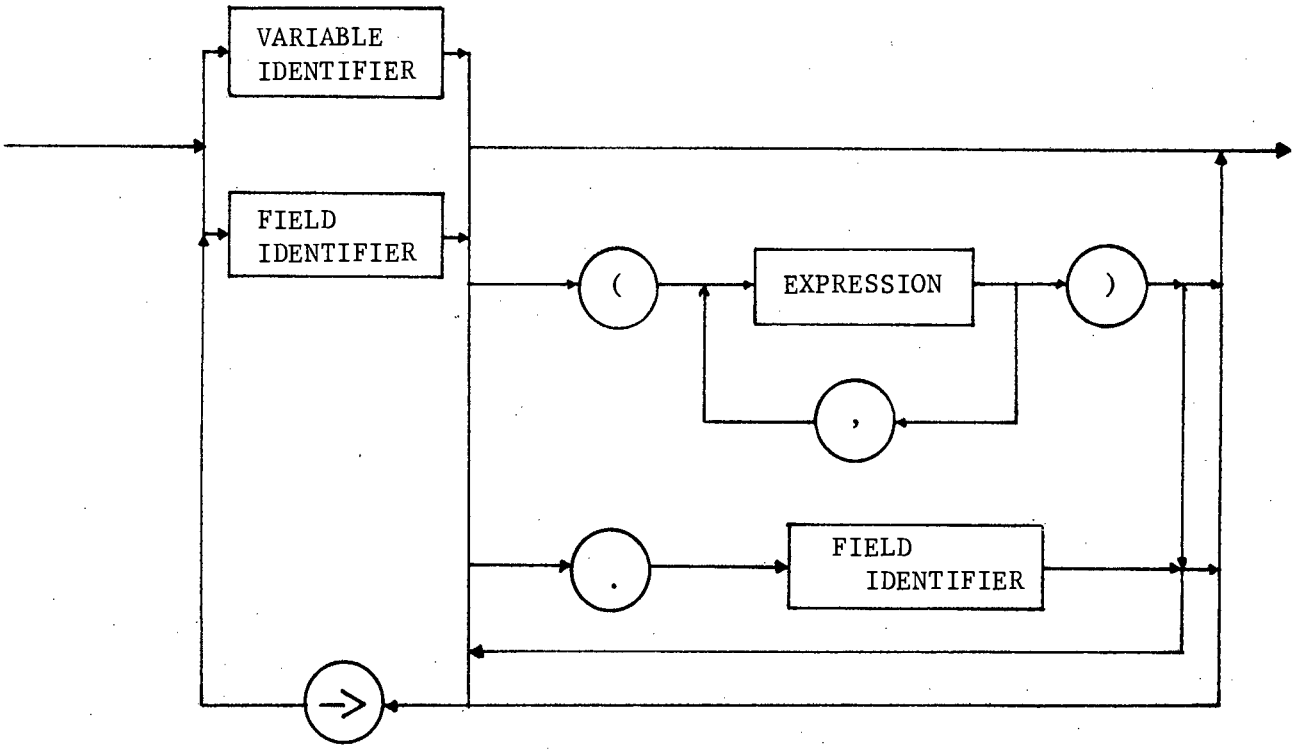


FIGURE 2.16. SELECTOR (VARIABLE)

2.2.6. STATEMENT PROCESSORS.

Procedure EXPRESSION is declared local to procedure STATEMENT. The syntax diagram of procedure STATEMENT, given in Figure 2.17, shows a finite-state machine which uses a transition table (in the form of a Pascal CASE statement) to drive the statement processors. Following is a brief resumé of the processors as they exist at present.

2.2.6.1. ASSIGNMENT.

The structure of procedure ASSIGNMENT matches the diagram in Figure 2.18. which is derived from

$$\langle \text{assignment statement} \rangle ::= \langle \text{target variable} \rangle \left\{ , \langle \text{target variable} \rangle \right\}^{\circ} = \langle \text{expression} \rangle$$

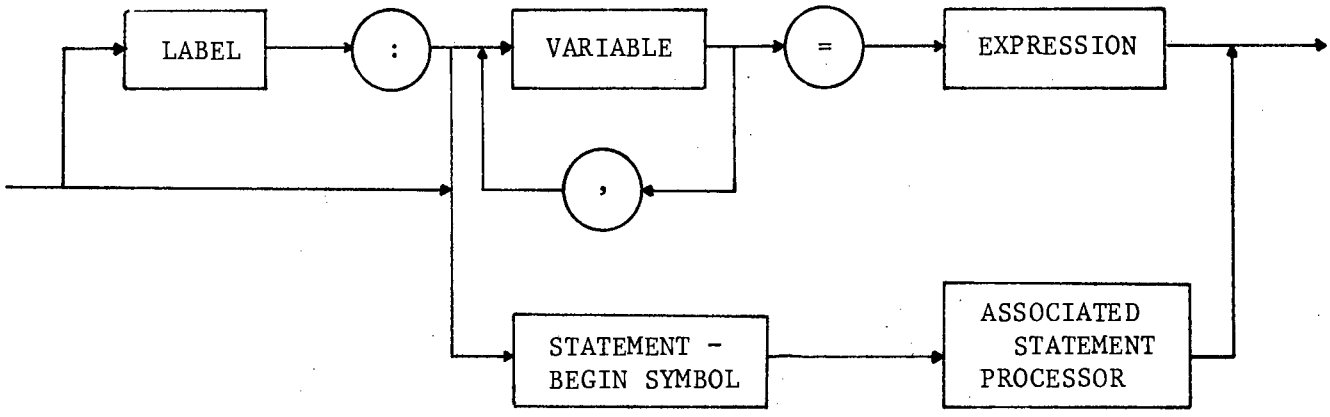


FIGURE 2.17. STATEMENT

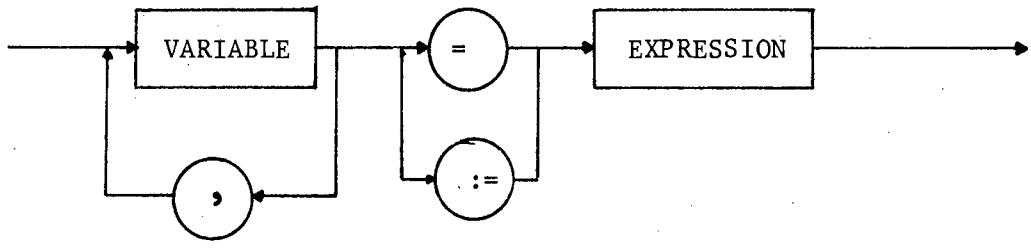


FIGURE 2.18. ASSIGNMENT

The procedure uses a local array of cells of type DESC to store information for each target variable, in case of multiple assignment, the number of targets being restricted to 10, and compares structures for each target with that of the expression before generating the necessary code. Conversions between FIXED, or subrange thereof, and FLOAT variables only are permitted.

2.2.6.2. ALLOCATESTATEMENT.

This procedure processes a statement of the form

ALLOCATE < based variable > SET (< pointer >) ;

where implicit pointer qualification takes place if the SET option is not exercised. An IDENTIFIER cell in the symbol table is expected with its CLASS = TYPES and BASED = TRUE , in which case code is generated to allocate space on the heap (the size is given by IDTYPE ⟶ SIZE) and to load the pointer with the address. Notice here that the cell structures for based variables are the same as those for user-defined types, the only difference being the TRUE value of field BASED.

2.2.6.3. COMPOUNDSTATEMENT.

There are two types of compound statement according to the compiler - those with and those without declarations. Each is treated differently as illustrated in Figure 2.19. in that the presence of the keywords DECLARE , CONST , or TYPE cases the invocation of procedure BLOCK , otherwise it is processed as a sequence of statements with only one entry point.

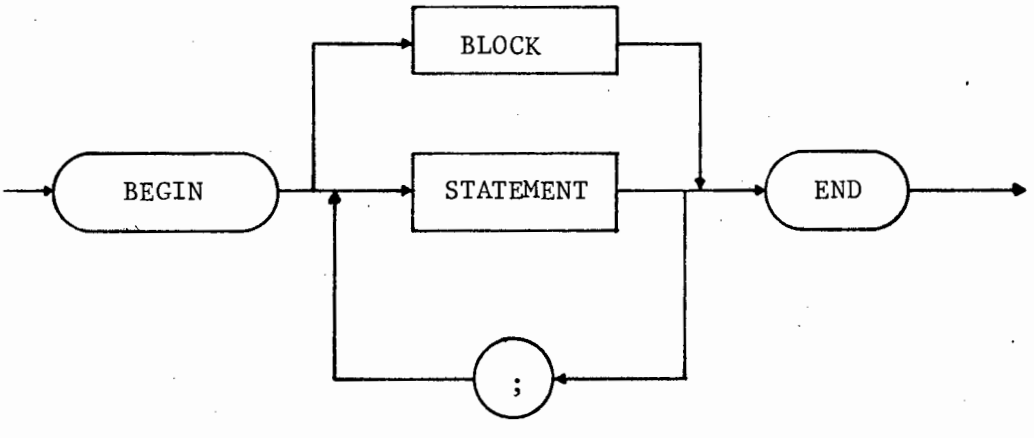


FIGURE 2.19. COMPOUNDSTATEMENT

2.2.6.4. PROCEDURE CALL.

The presence of a previously-declared procedure identifier, with keyword CALL optional, serves to execute the procedure defined under it. The feature of the optional 'CALL' , compatible with both Pascal and PL/I, was allowed by the fact that procedure STATEMENT can test identifiers found at the beginning of a statement for CLASS = PROC and transfer control accordingly.

The actual parameters which may follow a procedure call include procedure identifiers, fixed, float, character, label, set, or pointer variables and arrays. Parameters are all called by reference, and value parameters are not used, therefore values can be passed both ways.

The syntax of the procedure call is illustrated in Figure 2.20. In this implementation there is no special processing for procedures that are used recursively as all procedures are treated alike; therefore there is no need to define a procedure as RECURSIVE.

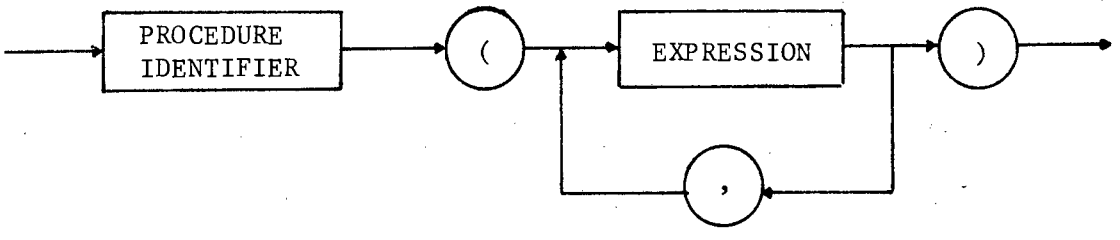


FIGURE 2.20. PROCEDURE CALL

2.2.6.5. DOSTATEMENT.

This procedure caters for four types of DO statement, according to the following:

```
< do statement > ::= DO < value-progression specification > ;
                                    < statements > END ;
< value-progression specification > ::= < do index > = < do list > |
                                        WHILE < expression > |
                                        < empty specification >

< empty specification > ::=
< do index >            ::= < identifier >
< do list >             ::= < do item > { , < do item > }^0
< do item >             ::= < item specification > | < item specification >
                                                     WHILE < expression >
< item specification >  ::= < value > TO < final value >
                                                     BY < increment >
    < value >           ::= < expression >
< final value >         ::= < expression >
< increment >           ::= < constant >
```

For a non-empty < value-progression specification > , the do-index , value , and final value must be of type FIXED and may not be altered in the range of the DO statement. A DO statement with an empty < value-progression specification > is the same as a compount statement without declarations, i.e. a sequence of statements with one entry point. The syntax diagram for the procedure is given in Figure 2.21. A few examples here, however, might be appropriate in order to demonstrate the power of the statement.

Stated formally,

```
DO i  = b BY s WHILE < expression > ; < statements > END ;
DO i  = p,q,b TO e,r,s ; < statements > END ;
DO WHILE < expression > ; < statements > END ;
DO i  = b, TO e, WHILE < expression >, b2 TO e2 WHILE < expression > ;
                                    < statements > END ;
DO; < statements > END ;
```

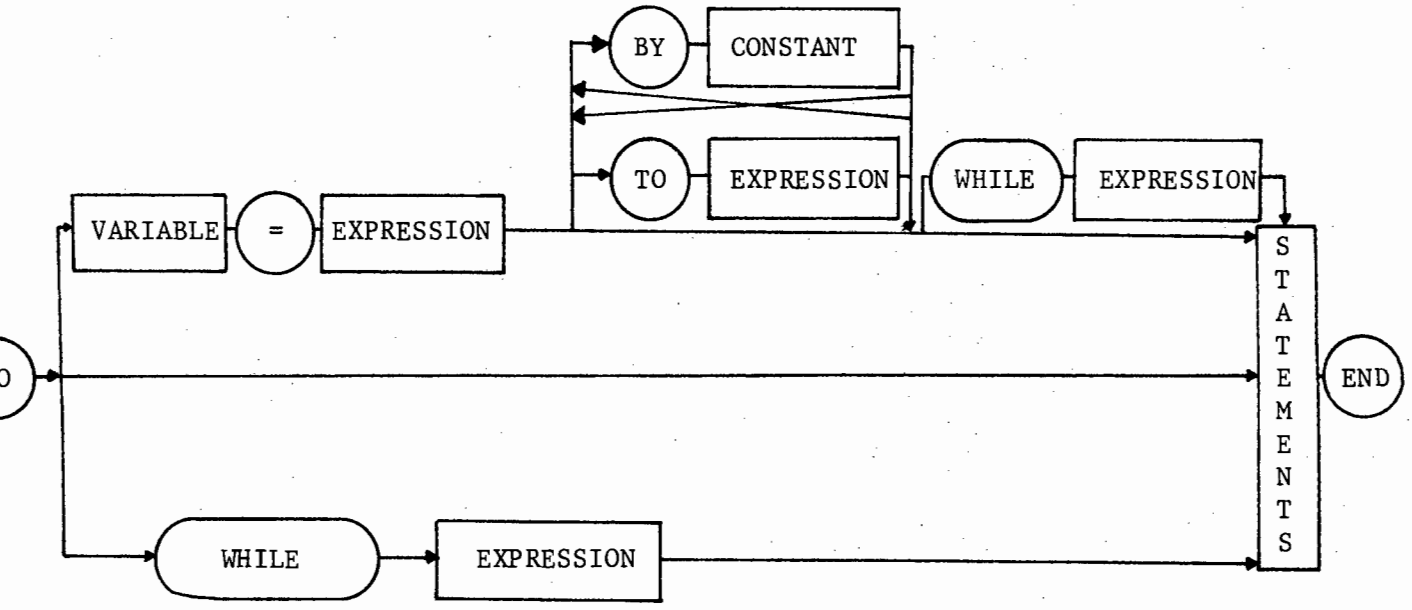


FIGURE 2.21. DOSTATEMENT

2.2.6.6. GETSTATEMENT.

⟨get statement⟩ ::= GET LIST (⟨list of variables⟩) |
GET FILE (⟨filename⟩) LIST (⟨list of variables⟩)

The default card-file SYSIN is assumed by the compiler if the FILE option is left unstated. Although GET LIST specifies stream input, the compiler actually allocates space for each file buffer, including the standard input and output buffers, in the user program, and these buffers, which are transparent to the user program, are used by the I/O run-time routines invoked by the generated code for GET.

The present implementation does not allow record or array identifiers in the list of variables, and only single variable-names are permitted.

2.2.6.7. PUTSTATEMENT.

The syntax of a PUT statement is very similar to that of a GET.

```
<put statement>  ::= PUT  < put list > |
                    PUT  FILE (< filename >) < put list >
<put list >      ::={< put options > }°   LIST ( < list of expressions > )
                                          { < put options > }°
<put options>    ::= SKIP | PAGE
```

Buffers are handled in the same way as for GET.

2.2.6.8. GOTOSTATEMENT.

```
<goto statement >   ::=  GO TO < label identifier > |
                         GOTO  < label identifier >
<label identifier>  ::=  < label constant >  |  < label variable>
<label constant >   ::=  < identifier >
<label variable>    ::=  < identifier>
```

Along with its syntax, this procedure is a simple one which generates an unconditional jump to the statement associated with the label specified. The scope of the label, tested at the end of the procedure containing the GO TO, is restricted to that procedure. It is therefore not possible to jump into or out of a procedure.

Procedure GOTOSTATEMENT sets a flag NXTSTLAB to ensure that every independent GO TO statement is followed by a labelled statement.

2.2.6.9. IFSTATEMENT.

```
<if statement>   ::= IF <expression> THEN  <statement> |
                     IF <expression> THEN  < statement > ;
                          ELSE  < statement >
```

< expression > must be of type Boolean, and < statement > may be any PL/I statement. If it is also an IF statement, the compiler interprets the statement as being equivalent to

```
IF < expression > THEN
    BEGIN if < expression > THEN < statement > ELSE < statement >
    END ;
```

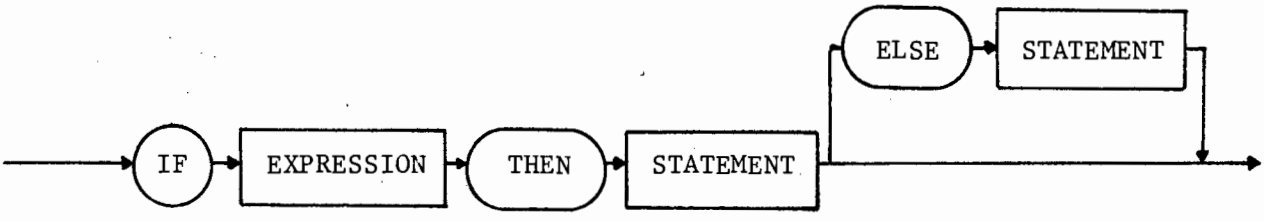


FIGURE 2.22. IFSTATEMENT

2.2.6.10. ONSTATEMENT.

The only ON condition implemented is ENDFILE, for which the construct is

< on statement > ::= ON ENDFILE (< filename >) < statement >

The processing performed by procedure ONSTATEMENT is to place the address of the contingency routine generated from < statement > into the field EOFADDR in the STRUCTURE cell of the file specified. A jump is also generated round the segment so that it does not form a part of the procedure in which it is defined. If an EOFADDR exists in a file's STRUCTURE cell, i.e. if a value has been inserted by procedure ONSTATEMENT, procedure GETSTATEMENT generates an END-OF-FILE test at each read, and a jump (if true) to the address contained in EOFADDR.

2.2.6.11. RETURNSTATEMENT.

< return statement > ::= RETURN < expression >

This statement is valid only in a function procedure. RETURN STATEMENT, after calling procedure EXPRESSION, stores the Function Result in the stack address at the base of the procedure, where it is interpreted as the current value on the stack on exit from the function procedure.

2.2.6.12. STOPSTATEMENT.

< stop statement > ::= STOP

As in procedure GOTOSTATEMENT, procedure STOPSTATEMENT sets a compiler flag NXTSTLAB to ensure that the next statement has an associated label.

2.2.6.13. THE EMPTY STATEMENT.

The empty statement consists of no symbols and causes control to exit from procedure STATEMENT.

< empty statement > ::=

It is especially useful in statements where a construct must be defined, but no action performed.
For example,

```
IF < expression > THEN
   IF < expression > THEN < statement > ;
     ELSE ;
     ELSE < statement > ;
```

Each of the non-empty statements defined above falls into one of four categories ; simple, compound, conditional, or repetitive.

Although a statement may contain other statements,

necessitating a recursive call of procedure STATEMENT, statements are analyzed and code generated for each one independently, and in sequence, until the end of a procedure is reached. At this point, control is transferred to procedure WRITEOUT, described below.

2.2.7. CODE OUTPUT.

Procedure WRITEOUT is a stand-alone procedure used to write out, to a disc file, in numeric and/or symbolic form, the code for each procedure. Thus the array CODE stores code for one procedure at a time and the compiler never holds the code for the entire program at any one time. (The one-pass nature of the processor avoids the necessity.) This fact, coupled with the compiler's present action of assigning jump addresses relative to the start address of the main procedure, makes it difficult to effectively implement external procedures in this system, although the present addressing mechanism is most conducive to efficient operation by the interpreter.

However, this could be improved by calculating instruction addresses relative to the base of each procedure, and changing the procedure call in the stack code to reference the name of the procedure instead of its address. Conditional, unconditional, and indexed jumps would then all contain addresses relative to the base of the procedure in which they were defined. This would confine all jumps and GOTO's to transfers within the procedure, which is in keeping with the concepts of structured programming.

This would permit external procedures to be compiled separately, a practice very much favoured by those advocates of efficient compilation, and a common library of procedures could be developed for the general benefit of the student user community.

2.3. IMPLEMENTATION.

Organisation of the implementation was dictated by the one-pass nature of the compiler. The compiler generates code into an array CODE, 'local' to each procedure in the user's program, before translating it into its symbolic form and writing it to disc. Although code is written out separately for each procedure, starting with the innermost procedure of each block, the structure of the language requires that the symbol table be present during the whole compilation, as shown earlier in Figure 2.11.

Figure 2.23. shows the complete organisation of the one-pass compiler system as a modular structure. The compiler itself, with its internal hierarchy of procedure, is embedded within the interpreter which simulates the machine/operating system for the self-contained processor.

Three major sections constitute the compiler system which is incorporated in the interpreter.

(1) The binary form of the 'hypothetical stack code' of the compiler.
(2) The stack.
(3) The heap.

The stack and the heap, which contains run-time generated elements, form the 'data bank' of the compiler, and together take up 15 000 words of main storage. Since the stack and heap move in opposite directions - the stack in the normal 'upward' direction and the heap starting at the top end of the bank - a check has to be performed at each movement to ensure that they do not overlap.

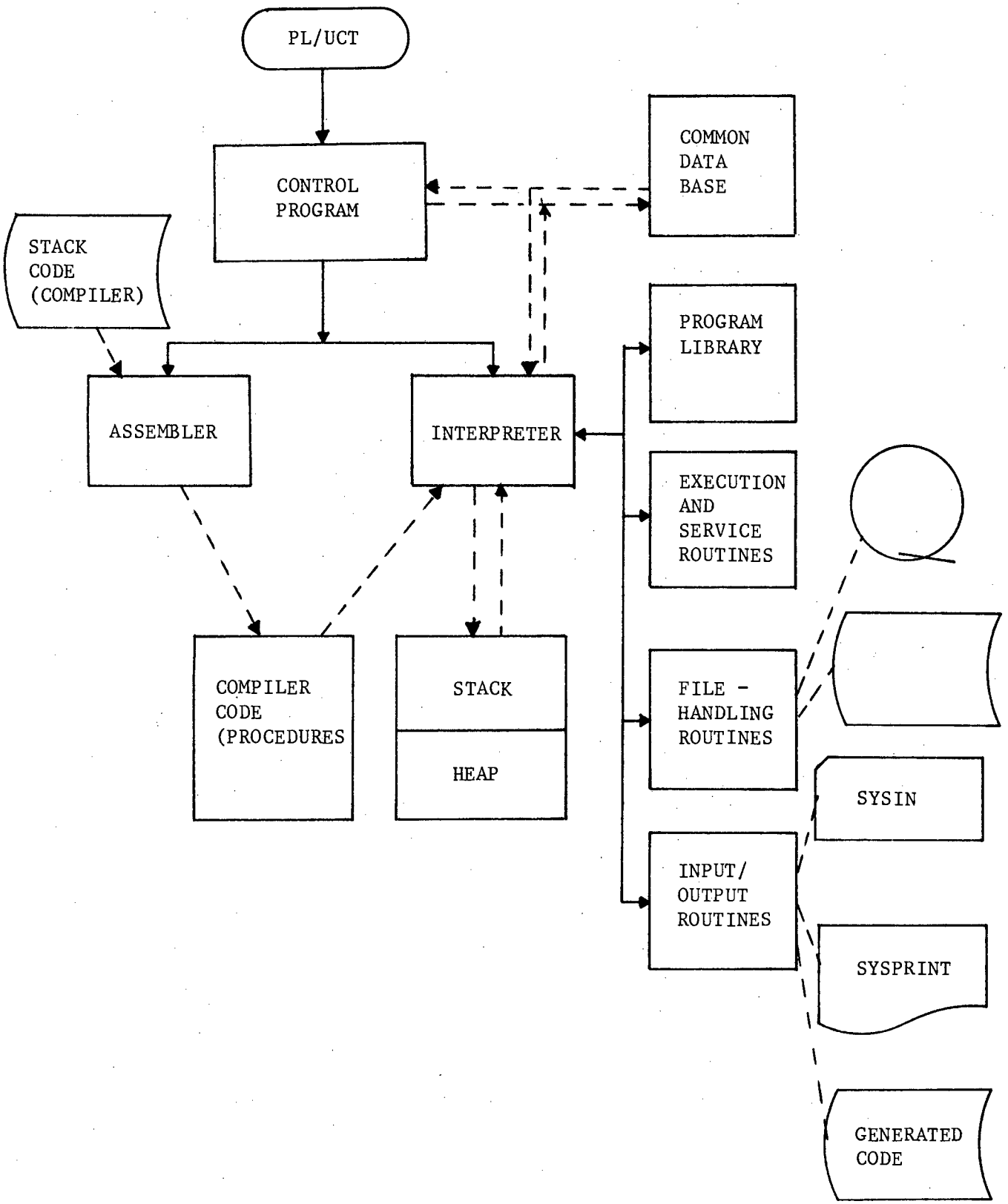


FIGURE 2.23. STRUCTURE OF THE COMPILER SYSTEM

2.3.1. THE STACK CODE.

The stack code consists of a set of mutually-exclusive procedures which form an instruction-bank of reentrant modules. The calling procedure for each of these 'modules', and therefore the overheads of each call, is the same although the compiler, for example, contains only five recursive procedures out of a total of 97. Appendix E gives a complete list, in sequence, of the procedures of the compiler as they appear in the instruction-bank of the interpreter.

In the implementation, each procedure defined contains an implicit declaration of the Function-result, the Static and Dynamic links, and the Return Address as 'local' parameters to the procedure. These cells are used by the run-time interpreter in the management of blocks and stack, and are completely transparent to the user although, of course, they take up space, called the 'static area', in the local area of each procedure.

The contents of these cells during the execution of a procedure are:-

(1) Function result: This is the value that will be left on top of the stack on exit from a function procedure.

(11) Static link: contains the base address of the procedure defined outer to the current procedure.

(111) Dynamic link: contains the base address of the calling procedure, which may or may not be the same as the static link.

(1V) Return Address: holds the address in the calling procedure to which control will be transferred on exit from the procedure.

These four constitute the static area for the procedure. Three other areas are defined as separate entities at the base of each procedure.

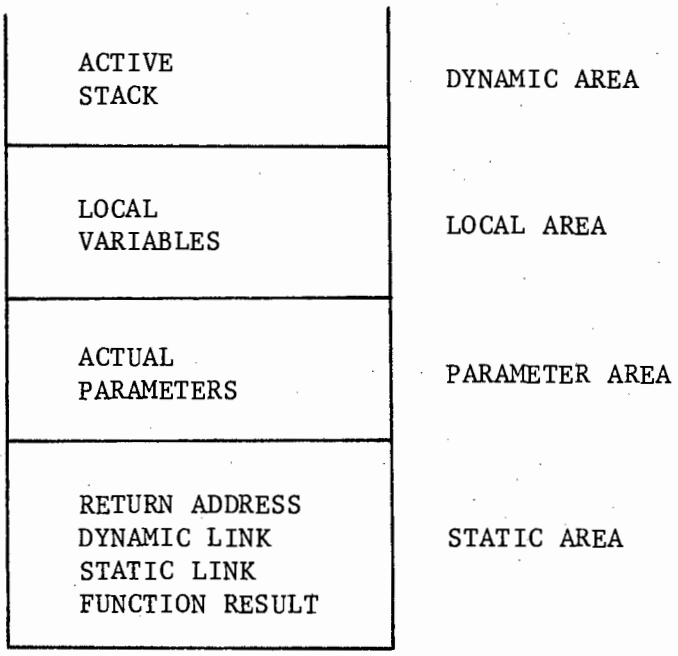


FIGURE 2.24. AREAS ON THE STACK

The parameter area stores any arguments associated with a procedure call. These (or rather their addresses) are loaded before entry to the procedure is performed, as shown in Figure 2.25. The local area contains all variables declared local to the procedure. The dynamic area, also called the active stack, is available for manipulation of data and computations.

The diagrams in Figure 2.25. give some explanation of the processes of block entry and block exit handled by the run-time interpreter.

Four pseudo-instructions are associated with block invocation and management.

a) MST - marks the stack by preserving the current position of the top of the stack, plus any other values needed on return from the procedure.

b) CUP - call user procedure - transfers control to the address given in its parameter Q after storing the Return Address.

c) ENT - announces entry into a procedure and is the first instruction encountered in each procedure.

d) RET - restores the stack to its original condition before transferring control back to the calling procedure.

The general sequence of actions for a procedure call is

(1) mark the stack
(2) load parameter addresses
(3) invoke the procedure.

The machine code itself is a triple of the form

OP , PARAMETER , ADDRESS ,

generated for a 'hypothetical' stack machine. A list of these pseudo-instructions, with their functions, is tabled in Appendix D, where they are grouped into classes dealing with

(1) data manipulation and computation
- set, logical, and arithmetic operations, loads, stores, and type conversions.

(2) flow of control, block invocation and management.

(3) builtin procedures and functions, execution-time and service routines, including input/output.

The design of these pseudo-instructions is such that, provided the execution-time and service routines exist, and the program can be assembled into a proper machine code, the compiler (and others) could be executed without the

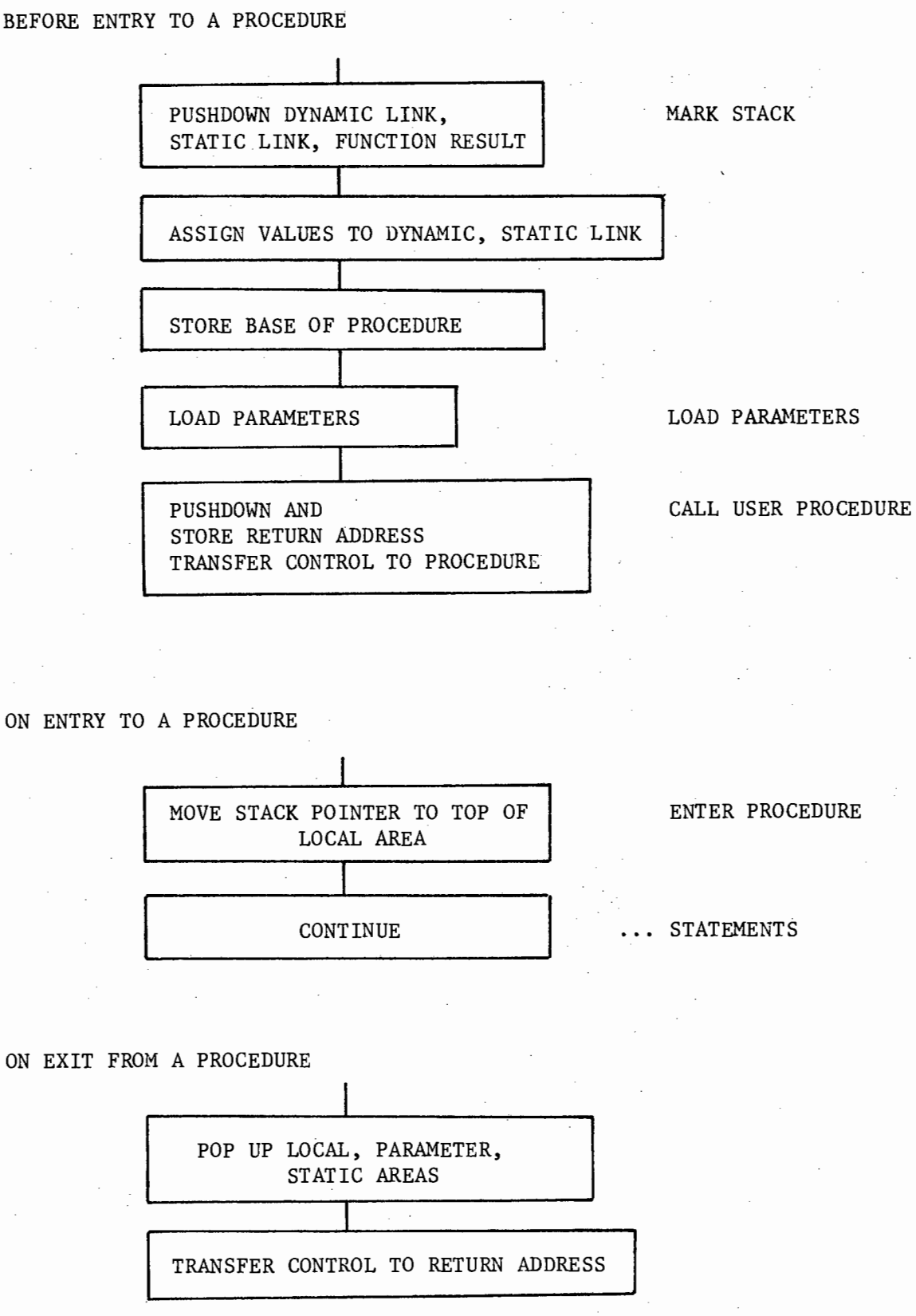


FIGURE 2.25. BLOCK ENTRY AND EXIT

use of the interpreter. Thus, besides the duties of stack management, code interpretation, and the provision of service routines, the interpreter performs no data checking or conversions without direction from the stack code, all checking being left in the hands of the compiler, and we might consider the interpreter as a simulator for a pseudo-machine.

With an operator range of 1 - 63, a maximum parameter value (corresponding to the maximum string length) of 63, and an address limit of 16 000, the interpreter was able to contain each instruction in one UNIVAC 1106 word of 36 bits.

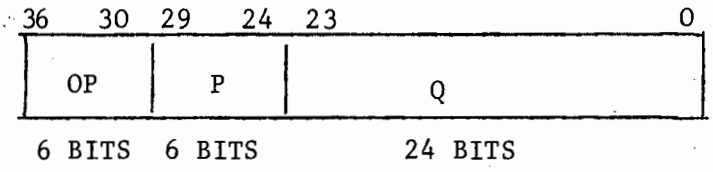


FIGURE 2.26. PSEUDO-CODE STRUCTURE

Each instruction having only one parameter and one address, the compaction of each code-line into one word allowed the compiler to be stored neatly in a 16 000 word array. With problems of housekeeping and link-editing at a minimum, and with all executable code located in one place, debugging of the compiler was greatly facilitated and testing made all the more reliable.

The notion of using one parameter and one address is a convenient one for a stack machine. At execution time the interpreter, when loading a simple variable, for instance, can locate its level and relative address in parameter P and address Q respectively.

For example, if I is a simple variable in the statement

```
I = I + 1 ;
```

then the corresponding code, shown in symbolic form, would be

```
LOD    2    8      load I from relative location 8
                   two outer blocks 'away'.
LDCI        1      load constant immediate
ADI                add integer
STR    2    8      store at address < 2,8 > .
```

If I is a field in a record, however, the code would look like

```
LDA    1    5      load the record address,
INC    2           incremented by the relative
                   address of the (target) field.
LDA    1    5      load the (same) record address,
IND    2           and index- fetch the field value.
LDCI        1
ADI
STO                store at address on top of the
                   stack.
```

There is an even further possibility, that of a field in a record that has been dynamically allocated.

In the statement

```
I = I + 1 ;
```

I is implicitly qualified by both record name and pointer, and the code expands to

```
LOD    1    6      load the address of field I
INC    2           through the address contained
                   in the pointer at < 1,6 >
LOD    1    6      load the pointer again
IND    2           and fetch the contents of
                   the address (+2) given
                   by the pointer.
LDCI        1
ADI
STO
```

Thus the number of instructions generated for the simple assignment statement varies from 4 to 7, depending on the complexity of the expressions.

2.3.2. THE EXECUTION SUPPORT ROUTINES.

The execution-time support system comprises the heap and stack management routines, input/output modules, the library of builtin functions and procedures incorporated into the compiler, together with any others necessary for the smooth running of the system.

Although they are all part of the interpreter, each routine is separate in location and function, and all are initiated by PL/UCT stack code.

Table 2.27. lists the execution-time routines embedded in the interpreter. At present, PL/UCT caters only for stream I/O, but record-oriented I/O should not be difficult to implement as the interpreter itself is record-oriented.

Mnemonic	accessible through	
MAX	implicit function	MAX
MIN	implicit function	MIN
NEW	ALLOCATE statement	
PAK	standard procedure	PACK
RST	standard procedure	RELEASE
SAV	standard procedure	MARK
SGN	implicit function	SIGN
SUM	implicit function	SUM
TIM	implicit function	TIME
EXP	mathematical function	EXP
COS	mathematical function	COS
LOG	mathematical function	LOG
SIN	mathematical function	SIN
SQT	mathematical function	SQRT
RDC, RDI, RDR	GET statement	
WRC, WRI, WRO, WRR, WRS	PUT statement	

TABLE 2.27. EXECUTION-TIME ROUTINES

The present number of 22 routines may be expanded to 63 without a major redesign of the pseudo-instruction.

2.3.2.1. EXECUTION ERRORS.

The interpreter incorporates a run-time error-message handler which, unlike the routines listed above, is initiated within the interpreter. On occurrence of an execution error, the type of error is established, and the stack and heap are then dumped in sections relating to the calling sequence of procedures. This provides a walkback to the main program, thus giving the exact path of the execution process. Addresses are shown in the dump which can then be traced back to the source listing.

The errors at present catered for by the interpreter include

(1) stack overflow
(2) active stack underflow
(3) value undefined
(4) heap overflow

and in each case execution is terminated.

Stack and heap overflow should only occur in programs with numerous (or looping) procedure calls or allocation of run-time variables, while active stack underflow, which relates to the 'dynamic area' above the local area for each procedure, is really a compiler fault, and may result from a compilation error where omitting the generation of a load instruction may create one too many stores.

The question of maintaining the 'value undefined' error is a tricky one, and while one of the main objectives of the compiler is to encourage a self-discipline amongst its users, this could be abolished in favour of a more lenient policy of allowing a certain number (say two) of undefined values to be replaced by zero, with a message advising the user to this effect.

This development could be part of a new execution-error handling strategy which would

a) allow a user-defined limit of execution errors before termination;

b) inform the user of execution errors in source-language-linked terms, giving such information as the statement number, object-code address, error type, the source symbol concerned, together with its name and address if a variable, plus a walkback of statement numbers and calling procedures, to the main program;

c) collate statistics on such data as

(i) usage frequency of each type of statement
(ii) frequency of occurrence of each type of execution error.

Here again a 'structured' approach could be used in which facilities and checking routines could be mobilized through options on the processor-call statement, thus giving a flexibility of execution-diagnostic power.

2.3.3. THE INTERPRETER SYSTEM.

It can be seen that the interpretive mode of execution provides tools hitherto unavailable in conventional absolute-code execution. For instance, an execution time trace may be instituted, without recompilation, by inserting a trap into the interpreter's instruction-fetch routine. This debugging aid was an absolute necessity during the testing of the compiler.

Another useful aid would be the dumping of the stack under certain, user-specified, conditions. Both features would be easy to implement as part of a general aid to users.

At the time of implementing the compiler system, one of the main goals was its establishment as a batch compiler oriented towards large-scale use by students. In pursuance of this aim, and in order for the compiler to be used efficiently and be made easily available to users under normal operating procedures, PL/UCT was built to operate under the standard (1100 series) operating system on the UNIVAC 1106. As a result, the compiler system duplicated many of the functions of the operating system in order to provide the user with the required error-checking and support routines.

The goal of the interpreter-compiler system is reflected in its structure, but its modular design also achieves a certain flexibility in that its compilation-execution flow can be changed to meet the needs of a conversational compiler system.

At present, a call of the PL/UCT processor initiates the flow, illustrated in Figure 2.28. of compilation followed by immediate execution if compilation is successful.

In addition to the diagnostic facilities, a number of options are available to the user. These belong to two sets - compiler options, and processor-call options. The compiler options form part of the source program, and may direct the compiler to

(i) produce an attribute listing
(ii) inhibit the source listing of the program
(iii) list the symbolic form of the generated code.

Future options, not yet implemented, could provide for

(i) local optimisation
(ii) printing a timing analysis
(iii) allow run-time checking of subscripts (or its inhibition)

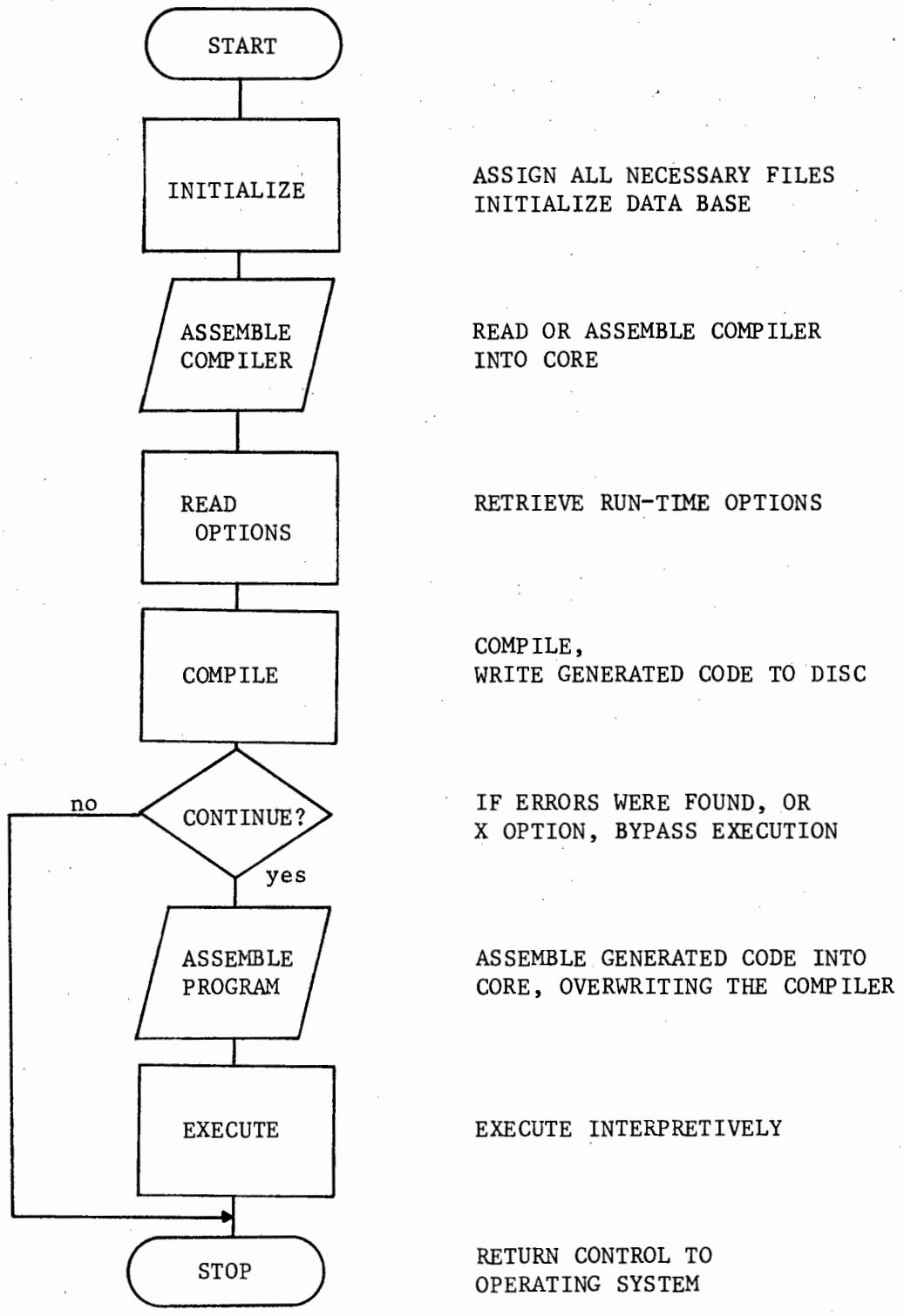


FIGURE 2.28. INTERPRETER FLOW

(iv) cross-reference listings

(v) printing selected portions of intermediate code

(vi) a user-defined 'fatality level' of compilation-errors.

The processor-call options serve to direct the control section of the interpreter, and their appearance on the control statement indicate one or more of the following:

I insert the following program deck into a file (element) named on the control statement.

T provide a trace of pseudo-code addresses.

U update the named file (element) with the following cards.

X bypass execution.

In addition, three other options were used in debugging and producing the compiler which are not available for general use

A assemble the symbolic stack code of the compiler into the binary array KODE, and write the array to disc.

B trace selected sections of the compiler.

E dump the contents of stack and heap on occurrence of a compilation error.

This last option, E, was especially useful in detecting 'illegal' errors produced by the compiler.

The interpreter uses a temporary drum file to store the code generated for the user program, and then 'assembles' it into the interpreter-array KODE, overwriting the compiler, before initiating execution. The card deck consists of the processor-call card followed by the program, a separator, and the data, as in the following example.

```
@ PL/UCT,  options          processor call
  [Source Program
  [
@ EOF                       separator
  [Data Cards
  [
```

FIGURE 2.29. EXAMPLE OF A CARD DECK

The only output from the PL/UCT system is the printer listing ; all programs are compiled, and executed, if successful, and generated code is never saved.

The present core requirements of the system are 50 000 words of main storage, which includes space for a maximum-sized PL/I program of 16 000 code instructions.

The upper-limit of 18 external files is a restriction imposed by the UNIVAC operating system and not the compiler. For each file declaration, space for the buffer is allocated within the compiler, and is a part of the stack, although it is manipulated by the file-handling routines within the interpreter. Because the interpreter is Fortran-written, the compiler associates a logical unit number with each file, starting with 11. Internal files are then linked to 'external' files on disc, drum or tape, through the interpreter and operating system, via the @USE executive control statement.
Thus

```
DECLARE SORT FILE ; /* PL/I declaration */
```

and

```
@ASG,T TEST              assign a temporary file
@USE   11,TEST           link it to unit no 11
```

will link the external file TEST with the PL/I internal file SORT.

These features provide a high degree of sophistication not very far from some of the already-established PL/I compilers mentioned. With only few restrictions on program size, numbers of constants, statements, and instructions per procedure, programs reaching the proportions of the compiler may be written.

2.4. OBSERVATIONS AND RESULTS

The actual performance of PL/UCT, demonstrated partly in the test programs in Appendix I, is as good as, if not better than anticipated. Compilation speeds, shown with each program, seem to be an order of magnitude slower than, for instance, the Algol compiler. This reduction in speed is to be expected, since the interpretive mode entails the expansion of each pseudo-instruction to between 20 and 80 machine instructions.

On average, simple PL/I statements involve the generation of 3 to 8 instructions, but some others, for example DO statements, contain many more. Although it cannot be gauged exactly, this gearing of source statements to machine instructions gives rise to heavy usage of the central processor, resulting in a CPU-bound compiler.

Examination of the test programs listed in Appendix I will show the power and breadth of the PL/UCT subset. These test programs were designed to test several main aspects of the compiler:-

- simple declarations and expressions
- data structures
  variables, arrays, records
- statement structures
  simple, compound, conditional, repetition
- block structures, recursion
- character strings
- builtin functions
- stream input/output

2.4.1. Test Program CALC

Program CALC, which has simple block and statement structures, has no intrinsic purpose but provides a comprehensive check of expressions containing references to simple variations, arrays, records and fields.

A number of features have been successfully compiled:

- Note the composit nature of the DECLARE statements and the default assignment of attributes, as in the case of J,K,L,M,N INITIAL(0),. The declaration S1,S2(10,20) declares both S1 and S2 as arrays. ITEM is an example of a record declaration, while the LIKE attribute is used to declare SAME as a record, and TARGET as FIXED BINARY.

- The (arbitrary) appearance of LAB1 and LAB2 demonstrate the use of statement labels.
- Various subscript expressions for arrays A1, CH, and S1 show the flexibility of the expression analyzer for subscripts.
- Character string assignment is shown in statement 17.
- Multiple assignment in statement 18 assigns the value '-A' to L,M, and N.
- Automatic conversion of FIXED and FLOAT variables is shown in statements 21,22, and 26.
- One- and two-dimensioned arrays are demonstrated in A1 and S1,S2.
- Implied and explicit qualification of record fields is used in statements 24 and 25.

The program performs a simple calculation which involves arithmetic operations, array subscripts, and type conversion, and output is performed with the PUT statement.

2.4.2. Test Program POLY

The purpose of program POLY is to calculate the value of a polynomial of which the degree and coefficients are read in from cards.

Once again, POLY has a simple block and statement structure, but in addition to the features such as assignment which have already been demonstrated in program CALC, the following are included:

- Stream input and output of simple variables, array elements, and character strings. A mixture of character strings and variables is shown in statement 13.
- The SKIP and PAGE options in PUT statements.
- DO statements of type 1 with (assumed) positive and negative increments of the do-index.
- A compound statement which contains all but the first two executable statements.

2.4.3. Test Program MAXIMUM

This program was inserted to illustrate the use of built-in functions in PL/UCT.

In addition to the use of function MAX, the program proves by its output the successful compilation and execution of input/output of array elements, and DO statements.

2.4.4. Test Program SORT

Using the standard Bubble Sort mechanism, program SORT demonstrates the use of nested DO statements in sorting a list of numbers which have been read from cards. Note the use of labels for each DO statement, and the compound statement embedded in the IF statement. Note also the declaration of X as an array, but the placing of T after array X causes it to be declared as a single variable.

2.4.5. Test Program FACTORIAL

Internal procedure FACT declared within main program FACTORIAL is called recursively to produce, as a result, the factorial of N which is read from a card.

The printed output attests to the success of this feature.

Note the abbreviated form of the DECLARE keyword, DCL, and the appearance of parentheses around identifiers N,F , an optional feature in PL/UCT.

2.4.6. Test Program PROCS

This program was designed to illustrate the definition of internal procedures, within the main program, and within internal procedures themselves.

Three internal procedures are defined, and one function. Procedure P1, which uses parameters of type FIXED and CHAR and which declares a local variable M , is called in the main program. P1 actually calls itself recursively to solve the Towers of Hanoi problem, and the solution is shown on the printed output.

Procedure P2 declares local variables D and E (which now override global variables D,E) and contains local procedure P3. The three assignment statements in P3 , plus the one in P2 are intended to illustrate the scope of variables A,D,F, FLOCAL, and G. Examination of the code generated for these statements will prove the correctness of their definition.

The definition of an internal function is given in F1, where A is declared as a formal parameter distinct from the global variable at level 1.

At its present stage of development the compiler successfully compiles a correct program in the PL/I subset. Where the compiler may fall down is in the compilation of an incorrect program, and this is most probably due to a wrong choice of relevant symbols used in checking context-sensitive errors. Comprehensive analysis of the behaviour of the compiler is obviously desirable, and further study can only lead to improvements to make PL/UCT a sophisticated compiler.

CHAPTER 3

CONCLUSION.

3. CONCLUSION

In assessing a compiler system such as PL/UCT, it is important to view all aspects in the light of the original aims of the project.

Flaws do still exist, and this is to be expected in a system of such complexity implemented in such a short period of time. Basically, however, the main objectives have been achieved.

· An effective PL/I subset has been designed and successfully implemented using the structured method of programming in a high-level language. The language subset is quite comprehensive and contains facilities which could usefully serve an intensive programming course. Its compatibility with well-known and widely-used PL/I means that students would learn language techniques that they may very well use after the course.

The implementation language described in section 2.1.2. had a marked effect on the design. Features such as dynamic allocation of user-defined records, which, in the case of identifiers, substituted for hash-coded tables, were most conducive to quick and sophisticated programming techniques. And the exact and precise structure of the compiler allow easy reference to the different sections.

· Output code is generated and interpreted in a batch environment. The compiler is easily accessible, and student jobs easy to run. The interpreter and service routines, described in section 2.3.3., co-operate to process user programs in the same way as other language processors. The clean, understandable program listings in Appendix I demonstrate the normal performance of the compiler.

· Comprehensive error checking is performed. The technique, using Pascal's SET feature, of dividing the input text into two disjoint parts - the set of relevant symbols and the set of irrelevant symbols - provided a powerful method of checking context-free errors while the use of descriptor cells, detailed in section 2.2.4. contributed much to semantic error analysis.

3.1. FUTURE IMPROVEMENTS

The PL/UCT compiler system has the potential for improvement in a number of areas.

Firstly, the language subset could be extended by the addition of new features and statements not yet implemented.

READ and WRITE would provide much-needed facilities in many applications.

FORMAT statements would improve the power of GET and PUT with the EDIT feature.

Input/output of arrays and repetitive specifications in I/O statements are urgently required.

Before these are inserted into the compiler, however, the task which should receive highest priority is the immediate generation of machine code in the compiler. Procedure WRITEOUT , which outputs the code to a file, is best equipped to accomplish this by easy alteration of code mnemonics to the UNIVAC (or PDP) assembly language.

The benefits of this would be three-sided:
· Compilation speeds would be greatly enhanced, satisfying the requirements of a fast-throughput batch system.

BIBLIOGRAPHY.

GLASS, R.L. — An Elementary Discussion of Compiler/Interpreter Writing. Document BCS- G0200. The Boeing Co. Oct 1968.

HASSITT, A., LAGESCHULTE, J.W., LYON, L.E. — Implementation of a High Level Language Machine. CACM 16,4 April 1973 p 199.

HIGMAN, B. — A Comparative Study of Programming Languages. Macdonald London 1967.

HOARE, C.A.R., WIRTH, N. — An Axiomatic Definition of the Programming Language Pascal. Eidgenössische Technische Hochschule, Zurich, Nov 1972.

HOLT, R.C. — Introducing Computer Programming using PL/I. Sigplan Notices, Dec 1970.

KENNEDY, M., SOLOMON, M.B. — Eight Statement PL/C(PL/ZERO) plus PL/ONE. Prentice Hall Inc. 1972.

LUCAS, P., WALK, K. — On the Formal Description of PL/I. Annual Review in Automatic Programming 6,3 1970 p 105.

MACLAREN, D. — PL/I Languages Development in ANSI/x. 3J1 and ECMA/TC10. ACM SIGPLAN NOTICES Dec 1970.

McCLURE, R.M. — An Appraisal of Compiler Technology. PROC AFIPS SJCC 1972 p 40.

McGREGOR, J.J., McDERMOTT, T.S. — An Interpreter System for Pascal and PL/I. Internal Report, UCT Computer Science Dept., July, 1974.

MORGAN, H.L., WAGNER, R.A. — PL/C : - The Design of a High-Performance Compiler for PL/I. AFIPS SJCC 1971 p 503.

WEGNER, P. — Programming Languages, Information Structures and Machine Organization. McGraw-Hill 1968.

WIRTH, N. — Program Development by Stepwise Refinement. CACM 14,4 1971 p 221.

WIRTH, N. — Systematic Programming : An Introduction. Prentice Hall Inc. 1973.

WIRTH, N. — The Programming Language Pascal (Revised Report). Eidgenössische Technische Hochschule, Zurich, Nov 1972.

WOLMAN, B.L. — Debugging PL/I Programs in the Multics Environment. PROC AFIPS FJCC 1972 p 507.

ZELKOWITZ, M.V. — Reversible Execution. CACM 16,9 Sept 1973 p 566.

APPENDIX A

THE GRAMMAR OF PL/UCT

APPENDIX A. THE GRAMMAR OF PL/UCT.

In the following notation, possible repetition of a construct is indicated by $\{ \text{----} \}^{0}$ (o or more repetitions) or $\{ \text{----} \}^{'}$ (1 or more repetitions).

BASIC VOCABULARY

```
<letter>     ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|
                 P|Q|R|S|T|U|V|W|X|Y|Z
<digit>      ::= 0|1|2|3|4|5|6|7|8|9
<character>  ::= <letter>|<digit>|+|-|*|/|<|>|¬|=|#|
                 [|]|(|)|&|$|'|:|;|.|,|!

<multiplying operator>  ::= *|/|&(AND)
<adding operator>       ::= +|-|OR
<relational operator>   ::= =(EQ)|#(¬=,<>,NE)|¬>(LE)|>(GT)|
                            <(LT)|¬<(GE)|IN
<operator>   ::= ¬(NOT)|<sign>|<multiplying operator>|
                 <adding operator>|<relational operator>

<block-begin symbol>    ::= <declaration-begin symbol>|<statement-begin
                            symbol>
<declaration-begin symbol> ::= DECLARE(DCL)|CONST|TYPE
<statement-begin symbol>   ::= <identifier>|ALLOCATE|BEGIN|
                               CALL|CASE|DO|GO|GOTO|GET|IF|
                               LOOP|ON|PUT|REPEAT|RETURN|STOP|
                               PROCEDURE(PROC)
<special symbol>        ::= [|]|(|)|'|:|;|.|,|!|->|:=|$|
                            <operator>|<block-begin symbol>|END|POINTER|
                            LABEL|CHARACTER(CHAR)|BIT|FILE|SET|
                            BINARY(BIN)|DECIMAL|FIXED|FLOAT|BASED|
                            LIKE|INITIAL(INIT)|TO|BY|WHILE|OF|UNTIL|
                            EXIT|THEN|ELSE|PAGE|LINE|SKIP|
                            LIST|EDIT|RETURNS|OPTIONS|EXTERNAL|FROM|
                            INTO|AUTOMATIC|STATIC
<identifier> ::= <letter>|{<letter or digit>}^0
<letter or digit> ::= <letter>|<digit>
```

```
<unsigned integer>   ::= {<digit>}'
<unsigned floating-point> ::= <unsigned integer> .{<digit>}' |
          <unsigned integer> .{<digit>}' E <scale factor>
<unsigned number>  ::= <unsigned integer> | <unsigned floating-point>
<scale factor>     ::= {<digit>}' | <sign> {<digit>}'
<sign>             ::=  + | -

<string>           ::=  ' {<character>}' '

<constant>         ::= <unsigned number> | <sign> <unsigned number> |
        <constant identifier> | <sign> <constant identifier> | <string>
<constant identifier> ::= <identifier>
```

DECLARATIONS

```
<constant declaration> ::= CONST <constant definition>
                              {; <constant definition>}°
<constant definition>  ::= <identifier> = <constant>

<simple-variable attribute> ::= <scale> | <base> | <scope> | <storage class> |
                                INITIAL(<constant> {,<constant>}°) |
                                LIKE <identifier>
<variable attribute> ::= <simple-variable attribute> | POINTER | LABEL |
                         CHARACTER {(<index type>)}° | BIT(BOOLEAN) |
                         SET OF <simple type>
<attribute>          ::= <variable attribute> | FILE
<scale>    ::= FIXED | FLOAT
<base>     ::= BINARY | DECIMAL
<scope>    ::= EXTERNAL | (INTERNAL)
<storage class>  := AUTOMATIC | STATIC | BASED(<identifier>)

<type>             ::= <simple type> | <attribute>
<simple type>      ::= <scalar type> | <subrangetype> |o <type identifier>
<scalar type>      ::= (<identifier> {,<identifier>}o)
<subrange type>    ::= <constant> .. <constant>
<type identifier>  ::= <identifier>

<type declaration> ::= TYPE <type definition> {; <type identifier>}°;
<type definition>  ::= <identifier> = <type>
```

⟨item declaration⟩ ::= ⟨procedure declaration⟩ | ⟨variable declaration⟩ |
⟨record declaration⟩

⟨procedure declaration⟩ ::= ⟨identifier⟩ ENTRY | ⟨identifier⟩ ENTRY ⟨parameter type list⟩ | ⟨identifier⟩ ENTRY RETURNS(⟨type⟩) | ⟨identifier⟩ ENTRY ⟨parameter type list⟩ RETURNS(⟨type⟩)

⟨parameter type list⟩ ::= (⟨type⟩ {,⟨type⟩}^{0})

⟨variable declaration⟩ ::= ⟨item list⟩ ⟨attribute list⟩

⟨item list⟩ ::= ⟨item⟩ {,⟨item⟩}^{0} | (⟨item⟩ {,⟨item⟩}^{0})

⟨item⟩ ::= ⟨identifier⟩ | ⟨identifier⟩ (⟨index type⟩ {,⟨index type⟩}^{0})

⟨index type⟩ ::= ⟨simple type⟩

⟨record declaration⟩ ::= 1 ⟨record identifier⟩ , ⟨field list⟩

⟨record identifier⟩ ::= ⟨identifier⟩

⟨field list⟩ ::= ⟨field item⟩ {,⟨field item⟩}^{0}

⟨field item⟩ ::= ⟨field level⟩ ⟨field identifier⟩ ⟨attribute list⟩

⟨field level⟩ ::= ⟨unsigned integer⟩

⟨field identifier⟩ ::= ⟨identifier⟩

⟨attribute list⟩ ::= {⟨attribute⟩}^{0}

⟨declaration⟩ ::= DECLARE(DCL) ⟨item declaration⟩ {,⟨item declaration⟩}^{0};

⟨declaration part⟩ ::= {⟨constant declaration⟩}^{0} {⟨type declaration⟩}^{0} {⟨declaration⟩}^{0}

⟨variable⟩ ::= ⟨variable identifier⟩ | ⟨indexed variable⟩ | ⟨field designator⟩ | ⟨based variable⟩

⟨variable identifier⟩ ::= ⟨identifier⟩

⟨indexed variable⟩ ::= ⟨variable identifier⟩ (⟨expression⟩ {,⟨expression⟩}^{0})

⟨based variable⟩ ::= ⟨pointer variable⟩→ ⟨variable⟩ | ⟨variable⟩

⟨pointer variable⟩ ::= ⟨variable⟩

⟨field designator⟩ ::= ⟨record identifier⟩ . {⟨field identifier⟩ .}^{0} ⟨field identifier⟩ | {⟨field identifier⟩ .}^{0} ⟨field identifier⟩

⟨field identifier⟩ ::= ⟨identifier⟩

⟨factor⟩ ::= ⟨variable⟩ | ⟨unsigned constant⟩ |
⟨function designator⟩ | ⟨set⟩ | (⟨expression⟩) |
¬ (NOT) ⟨factor⟩

⟨function designator⟩ ::= ⟨function identifier⟩ |
⟨function identifier⟩ (⟨actual parameter⟩ { , ⟨actual parameter⟩ }^{o})

⟨function identifier⟩ ::= ⟨identifier⟩

⟨actual parameter⟩ ::= ⟨variable⟩ | ⟨expression⟩

⟨set⟩ ::= [⟨expression⟩ { , ⟨expression⟩ }^{o}] | [ ]

⟨term⟩ ::= ⟨factor⟩ | ⟨term⟩ ⟨multiplying operator⟩ ⟨factor⟩

⟨simple expression⟩ ::= ⟨term⟩ | ⟨simple expression⟩ ⟨adding⟩
⟨operator⟩ ⟨term⟩ ⟨adding operator⟩ ⟨term⟩

⟨expression⟩ ::= ⟨simple expression⟩ | ⟨simple expression⟩
⟨relational operator⟩ ⟨simple expression⟩

⟨statement⟩ ::= ⟨unlabelled statement⟩ | ⟨label⟩ : ⟨unlabelled statement⟩

⟨unlabelled statement⟩ ::= ⟨assignment⟩ | ⟨allocate statement⟩ |
⟨compound statement⟩ | ⟨procedure definition⟩ |
⟨procedure call⟩ | ⟨case statement⟩ | ⟨do statement⟩ |
⟨get statement⟩ | ⟨goto statement⟩ | ⟨if statement⟩ |
⟨loop statement⟩ | ⟨on statement⟩ | ⟨put statement⟩ |
⟨repeat statement⟩ | ⟨return statement⟩ |
⟨stop statement⟩ | ⟨empty statement⟩

⟨label⟩ ::= ⟨identifier⟩

⟨empty⟩ ::=

⟨assignment⟩ ::= ⟨variable⟩ { , ⟨variable⟩ }^{o} = ⟨expression⟩ |
⟨variable⟩ { , ⟨variable⟩ }^{o} := ⟨expression⟩

⟨allocate statement⟩ ::= ALLOCATE ⟨allocate item⟩ { , ⟨allocate item⟩ }^{o}

⟨allocate item⟩ ::= ⟨based variable⟩ SET(⟨pointer variable⟩) |
⟨based variable⟩

⟨compound statement⟩ :: = BEGIN ⟨declaration part⟩ ⟨statements⟩ END |
BEGIN ⟨statements⟩ END

⟨statements⟩ ::= ⟨statement⟩ { , ⟨statement⟩ }^{o}

⟨procedure definition⟩ ::= [PROC / PROCEDURE] ⟨procedure heading⟩ ⟨declaration part⟩
⟨body part⟩ END

⟨procedure heading⟩ ::= OPTIONS(⟨procedure option⟩) |
⟨formal parameter list⟩ | ⟨formal parameter list⟩ RETURNS (⟨type⟩)

⟨procedure option⟩ ::= MAIN

⟨formal parameter list⟩ ::= ⟨empty⟩ | (⟨formal parameter⟩ {, ⟨formal parameter⟩}^{o})

⟨body part⟩ ::= ⟨statements⟩

⟨procedure call⟩ ::= ⟨procedure identifier⟩ |
⟨procedure identifier⟩ (⟨actual parameter⟩ {, ⟨actual parameter⟩}^{o})

⟨procedure identifier⟩ ::= ⟨identifier⟩

⟨case statement⟩ ::= CASE ⟨expression⟩ OF
⟨case list element⟩ {; ⟨case list element⟩}' END

⟨case list element⟩ ::= ⟨case label list⟩ : ⟨statement⟩

⟨case label list⟩ ::= ⟨case label⟩ {, ⟨case label⟩}^{o}

⟨case label⟩ ::= ⟨constant⟩

⟨do statement⟩ ::= DO ⟨value-progression specification⟩ ; ⟨statements⟩ END

⟨value-progression specification⟩ ::= ⟨do index⟩ = ⟨do list⟩ | WHILE ⟨expression⟩
| ⟨empty⟩

⟨do index⟩ ::= ⟨identifier⟩

⟨do list⟩ ::= ⟨do item⟩ {, ⟨do item⟩}^{o}

⟨do item⟩ ::= ⟨item specification⟩ | ⟨item specification⟩ WHILE ⟨expression⟩

⟨item specification⟩ ::= ⟨value⟩ | ⟨value⟩ TO ⟨final value⟩ |
⟨value⟩ TO ⟨final value⟩ BY ⟨increment⟩ |
⟨value⟩ BY ⟨increment⟩ |
⟨value⟩ BY ⟨increment⟩ TO ⟨final value⟩

⟨value⟩ ::= ⟨expression⟩

⟨final value⟩ ::= ⟨expression⟩

⟨increment⟩ ::= ⟨constant⟩

⟨get statement⟩ ::= GET LIST (⟨variable⟩ {, ⟨variable⟩}^{o}) |
GET FILE(⟨filename⟩) LIST (⟨variable⟩ {, ⟨variable⟩}^{o})

⟨filename⟩ ::= ⟨identifier⟩

⟨goto statement⟩ ::= GO TO ⟨goto label⟩ | GOTO ⟨goto label⟩

⟨goto label⟩ ::= ⟨label variable⟩ | ⟨label constant⟩

⟨label variable⟩ ::= ⟨variable⟩

⟨label constant⟩ ::= ⟨label⟩

⟨if statement⟩ ::= IF ⟨expression⟩ THEN ⟨statement⟩ |
IF ⟨expression⟩ THEN ⟨statement⟩; ELSE ⟨statement⟩

⟨loop statement⟩ ::= LOOP ⟨statements⟩ EXIT IF ⟨expression⟩ ;
⟨statements⟩ END

⟨on statement⟩ ::= ON ⟨condition⟩ ⟨statement⟩

⟨condition⟩ ::= ENDFILE(⟨filename⟩)

⟨put statement⟩ ::= PUT ⟨put list⟩

PUT FILE (⟨filename⟩) ⟨put list⟩

⟨put list⟩ ::= $\{$⟨put options⟩$\}^{o}$ LIST (⟨list of expressions⟩)

$\{$⟨put options⟩$\}^{o}$ | ⟨put options⟩

⟨put options⟩ ::= SKIP | PAGE

⟨list of expressions⟩ ::= ⟨expression⟩ $\{$,⟨expression⟩$\}^{o}$

⟨repeat statement⟩ ::= REPEAT ⟨statements⟩ UNTIL ⟨expression⟩

⟨return statement⟩ ::= RETURN ⟨expression⟩

⟨stop statement⟩ ::= STOP

⟨empty statement⟩ ::=

⟨block⟩ ::= ⟨procedure definition⟩ | ⟨compound statement⟩

⟨program⟩ ::= ⟨procedure definition⟩

APPENDIX B

STANDARD PROCEDURES AND BUILTIN FUNCTIONS

APPENDIX B

STANDARD PROCEDURES

(1) PACK(X,Y) - pack the character elements in array X into the single variable Y.

(2) UNPACK(X,Y) - the complement of pack. Place each character in variable X into an element in array Y.

(3) MARK(X) - marks the heap by placing the current address in the execution-time heap pointer into X. This 'saves' the current value of the pointer so that variables allocated after this time can be effectively 'freed' by restoring this value to the heap pointer.

(4) RELEASE(X) - the instruction which releases heap storage by restoring the heap pointer value from X. It is the complement of MARK.

BUILTIN FUNCTIONS

(1) ABS(X) - computes the absolute value of X. X may be FIXED or FLOAT. The type of the result is the type of X.

(2) CHR(X) - returns the CHARACTER which has the ordinal value (rank) of X. X must be an integer in the range 0_8 , 77_8.

(3) MAX(X,Y) - finds the highest-valued expression of the two arguments. Arguments may be FIXED or FLOAT.

(4) MIN(X,Y) - is similar to MAX except that it finds the lowest-valued argument.

(5) ORD(X) - is the opposite of CHR in that it returns the rank of character X. X is a single character.

(6) PRED(X) - gives the value of the predecessor of X in an ordered set. Thus in any set of n values, the predecessor of value i is value $i-1$.

(7) SIGN(X) - returns the value -1 , 0 , or + 1 , depending on whether X is negative, zero, or positive. The value returned is FIXED BINARY. X may be FIXED or FLOAT.

(8) SUCC(X) - gives the value of the successor to X in an ordered set. The successor of V_i is V_{i+1} . Thus X = SUCC(PRED(X))
= PRED(SUCC(X))

(9) SUM(X) - produces the sum of all the elements in the array X.

(10) TIME - returns a FLOAT value of the current time in seconds.

(11) TRUNC(X) - returns a value of type FIXED , such that if $X \geq 0$, then $X - 1 < TRUNC(X) \leq X$. X must be FLOAT.

MATHEMATICAL BUILTIN FUNCTIONS

(1) COS(X) - returns the trigonometric cosine of the argument X . X must be a FLOAT value in radians.

(2) EXP(X) - returns the value of e raised to the Xth power.

(3) LOG(X) - finds the natural logarithm (base e) of the argument X. X must be greater than zero.

(4) SIN(X) - gives the trigonometric sine of X when X is expressed in radians. X must be of type FLOAT.

(5) SQRT(X) - finds the positive value of the square root of X. X must not be negative.

For all the above mathematical functions, the type of the result is FLOAT.

```
114             DESC = RECORD TYPTR: STP;
115                       CASE KIND: DESCKIND OF
116                         CST:   (CVAL: VALU);
117                         VARBL: (CASE ACCESS: VACCESS OF
118                                   DRCT: (VLEVEL: LEVRANGE; DPLMT: ADDRRANGE);
119                                   INDRCT: (IDPLMT: ADDRRANGE))
120                       END;
121
122           /*LABELS*/
123            LBP = ! LABL;
124            LABL = RECORD NEXTLAB: LBP;
125                          LABNAME: ALFA;  LABLEV: INTEGER;
126                          CASE DEFINED: BOOLEAN OF
127                            TRUE: (LABADDR: ADDRRANGE)
128                    END;
129
130
131       /*-----------------------------------------------------------------*/
132
133
134       VAR
135                                          /*RETURNED BY SCANNER *INSYMBOL* */
136
137           SY: SYMBOL;                    /*LAST SYMBOL*/
138           OP: OPERATOR;                  /*CLASSIFICATION OF LAST SYMBOL*/
139           VAL: VALU;                     /*VALUE OF LAST CONSTANT*/
140           LGTH: INTEGER;                 /*LENGTH OF LAST STRING CONSTANT*/
141           ID: ARRAY [1..ALFALENG] OF CHAR;/*LAST IDENTIFIER (POSSIBLY TRUNCATED)*/
142           KK: 1..ALFALENG;               /*NR OF CHARS IN LAST IDENTIFIER*/
143           CH: CHAR;                      /*LAST CHARACTER*/
144           LINE: ARRAY [1..80] OF CHAR;   /*CURRENT SOURCE LINE*/
145
146           LLABEL: ARRAY [1..ALFALENG] OF CHAR;   /*LAST LABEL*/
147           STATLAB,                       /*LABELLED STATEMENT*/
148           NXTSTLAB: BOOLEAN;             /*NEXT STATEMENT TO BE LABELLED*/
149           LASTSY: SYMBOL;
150           POSV: BOOLEAN;
151
152                                          /*USED BY PROCEDURE TYP*/
153           VBASE: BASE;
154           VBASED: BOOLEAN;
155           RECIDP: IDP;
156                                          /*COUNTERS:*/
157
158           CHCNT: 1..80;                  /*CHARACTER COUNTER*/
159           GLC,GIC,
160           LC,IC: ADDRRANGE;              /*DATA LOCATION AND INSTRUCTION COUNTER*/
161           LINENO,                        /*LINE COUNTER*/
162           FILENO: INTEGER;               /*FILE UNIT NUMBER*/
163
164
165                                          /*SWITCHES:*/
166
167           DP,                            /*DECLARATION PART*/
168           BODY,                          /*BODY PART*/
169           PRTERR,                        /*TO ALLOW FORWARD REFERENCES IN POINTER TY
170                                           DECLARATION BY SUPPRESSING ERROR MESSAGE*/
```

```
171         ATTRLIST,ERRORS,LOCOPT,PRTIME,
172         SUBSCRIPT,TRACE,XREF,LIST,
173         PRCODE,PRSYMB: BOOLEAN;            /*OUTPUT OPTIONS FOR
174                                              -- ATTRIBUTE LISTINGS
175                                              -- NO EXECUTION
176                                              -- LOCAL OPTIMISATION
177                                              -- PRINT TIMING ANALYSIS
178                                              -- DON'T CHECK SUBSCRIPTS
179                                              -- ALLOW EXECUTION TIME TRACE
180                                              -- CROSS-REFERENCE LISTING
181                                              -- SOURCE PROGRAM LISTING
182                                              -- PRINTING ASSEMBLED CODE
183                                              -- PRINTING SYMBOLIC CODE
184                                              --> PROCEDURE OPTION*/
185
186
187                                            /*POINTERS:*/
188         FIXPTR,FLTPTR,CHARPTR,
189         BOOLPTR,NILPTR,TEXTPTR: STP;       /*POINTERS TO ENTRIES OF STANDARD IDS*/
190         UTYPPTR,UCSTPTR,UVARPTR,
191         UFLDPTR,UPRCPTR,UFIDPTR,           /*POINTERS TO ENTRIES FOR UNDECLARED IDS*/
192         FWPTR: IDP;                        /*HEAD OF CHAIN OF FORW DECL TYPE IDS*/
193         FSTLABP: LBP;                      /*HEAD OF LABEL CHAIN*/
194         MAINP: IDP;                        /*MAIN PROCEDURE*/
195
196
197                                            /*BOOKKEEPING OF DECLARATION LEVELS:*/
198
199         LEVEL: LEVRANGE;                   /*CURRENT STATIC LEVEL*/
200         TREEX,                             /*LEVEL OF LAST ID SEARCHED BY SEARCHID*/
201         TOP: TREERANGE;                    /*TOP OF IDTREE*/
202
203         IDTREE: ARRAY[TREERANGE] OF IDP; /*SYMBOL TABLE TREE ROOTS*/
204
205         BLOCKS: ARRAY[1..20] OF            /*BLOCK NAMES AND LEVELS*/
206                   PACKED RECORD
207                     BEGLEV: INTEGER;
208                     BNAME: ARRAY[1..ALFALENG] OF CHAR;
209                     BTYPE: BLKTYPE
210                   END;
211
212         BIX: TREERANGE;
213         BLMAX,ENDLEV: INTEGER;
214         BDELIM,EDELIM: BOOLEAN;
215         MBLOCKTYPE: BLKTYPE;
216
217                                            /*ERROR MESSAGES:*/
218
219         FATALERR: BOOLEAN;
220         ERRTOT,                            /*TOTAL NO OF ERRORS*/
221         ERRINX: INTEGER;                   /*NR OF ERRORS IN CURRENT SOURCE LINE*/
222         ERRLIST:                           /*LIST OF ERROR NUMBERS AND THEIR*/
223           ARRAY [1..10] OF                 /*POSITIONS IN THE SOURCE LINE*/
224             PACKED RECORD POS: 1..80;
225                           NMR: 1..900
226                    END;
227
```

```
228
229                                     /*FORWARD REFERENCES OF PROCEDURES:*/
230         JMPIX: 2..JMPMAX;           /*ARE SATISFIED BY INDIRECT JUMPS */
231         JMPTAB: ARRAY ¢3..JMPMAX¢ OF /*VIA THE JUMP AREA, WHICH IS WRITTEN */
232                   ADDRRANGE;        /*OUT AT THE END OF COMPILATION AND*/
233                                     /*MUST BE LOADED WITH ITS VIRTUAL */
234                                     /*ORIGIN JMPTAB¢0¢ AT ABS ADDRESS ZERO*/
235
236
237                                     /*EXPRESSION COMPILATION:*/
238
239         GDESC: DESC;                /*DESCRIBES THE EXPR CURRENTLY COMPILED*/
240
241
242                                     /*STRUCTURED CONSTANTS:*/
243
244         LETTERS,DIGITS: SET OF CHAR;
245         CONSTBEGSYS,SIMPTYPEBEGSYS,TYPEBEGSYS,BLOCKBEGSYS,SELECTSYS,
246         FACBEGSYS,STATBEGSYS,DECLBEGSYS,SVATTRSYS,VATTRSYS,ATTRSYS,
247         STATMENTSYS,SEMICOMMA,LISTDELIMS,STATDELIMS: SYSET;
248         RW:  ARRAY ¢1..71/*NR. OF RES. WORDS*/¢ OF ALFA;
249         FRW: ARRAY ¢1..11/*ALFALENG+1*/¢ OF 1..72/*NR. OF RES. WORDS + 1*/;
250         RSY: ARRAY ¢1..71/*NR. OF RES. WORDS*/¢ OF SYMBOL;
251         SSY: ARRAY ¢'¢'..'_'¢ OF SYMBOL;
252         PSSY: ARRAY ¢1..6/*NR OF PSEUDO OPERATORS*/¢ OF ALFA;
253         ROP: ARRAY ¢1..71/*NR. OF RES. WORDS*/¢ OF OPERATOR;
254         SOP: ARRAY ¢'¢'..'_'¢ OF OPERATOR;
255         NA:  ARRAY ¢1..29¢ OF ALFA;
256         MN:  ARRAY ¢0..57¢ OF PACKED ARRAY ¢1..3¢ OF CHAR;
257         SNA: ARRAY ¢1..22¢ OF PACKED ARRAY ¢1..3¢ OF CHAR;
258         PSOP: ARRAY ¢1..6¢ OF OPERATOR;
259
260
261       /*-----------------------------------------------------------*/
262
263
264       PROCEDURE ENDOFLINE;
265               /*CALLED BY INSYMBOL TO PRINT THE
266                 SOURCE LINE (WITH ERRORS, IF ANY)*/
267         LABEL 1; VAR K,F,LEV: INTEGER;
268
269         PROCEDURE DIAGNOSTIC;
270               /*THIS MODULE WILL EVENTUALLY PRODUCE DIAGNOSTIC
271                 MESSAGES AND SUGGESTED CORRECTIONS*/
272         BEGIN
273           WRITE(' ':2);
274           WRITE(ERRLIST¢K¢:NMR:3);
275         END /*DIAGNOSTIC*/ ;
276
277       BEGIN
278         IF ^LIST THEN IF ERRINX = 0 THEN GOTO 1;
279             WRITE(' ':2);
280             IF BDELIM THEN
281               BEGIN  LEV := BLOCKS¢BIX¢.BEGLEV;
282                 WRITE('B':1);
283                 F := 1;
284                 IF LEV > 9 THEN F := 2;
```

```
285               WRITE(LEV:F,' ':3-F)
286             END
287           ELSE WRITE(' ':4);
288           IF EDELIM THEN
289             BEGIN WRITE('E':1);
290               F := 1;
291               IF ENDLEV > 9 THEN F := 2;
292               WRITE(ENDLEV:F,' ':3-F)
293             END
294            ELSE WRITE(' ':4);
295           IF BOD THEN WRITE(GIC:5) ELSE WRITE(' ':5);
296           IF OP THEN WRITE(GLC:5) ELSE WRITE(' ':5);
297           WRITE(LINENO:4,'* ':2);
298           WRITE(LINE:CHCNT,EOL);
299           IF ERRINX > 0 THEN
300             BEGIN
301               FOR K := 1 TO ERRINX DO
302                 BEGIN
303                   WRITE(' ':5,' **ERROR**':10);
304                   DIAGNOSTIC;
305                   WRITE(LINENO:4,'* ':2);
306                   WRITE(' ':ERRLIST^[K].POS-2,'*':1,EOL);
307                 END;
308               ERRTOT := ERRTOT + ERRINX; ERRINX := 0;
309             END;
310       1: CHCNT := 0; LINENO := LINENO + 1; GLC := LC; GIC := IC;
311       BDELIM := FALSE; EDELIM := FALSE;
312     END  /*ENDOFLINE*/ ;
313
314       PROCEDURE ERROR(FERRNR: INTEGER);
315             /*PLACE AN ERROR NUMBER IN ARRAY ERRLIST FOR
316               LATER USE BY ENDOFLINE*/
317       BEGIN
318         IF ERRINX >= 9 THEN
319           BEGIN ERRLIST^[10].NMR := 700; ERRINX := 10 END
320         ELSE
321           BEGIN ERRINX := ERRINX + 1;
322             ERRLIST^[ERRINX].NMR := FERRNR
323           END;
324         ERRLIST^[ERRINX].POS := CHCNT
325       END /*ERROR*/ ;
326
327       PROCEDURE INSYMBOL;
328         /*READ NEXT BASIC SYMBOL OF SOURCE PROGRAM AND RETURN ITS
329         DESCRIPTION IN THE GLOBAL VARIABLES SY, OP, ID, VAL AND LGTH*/
330         LABEL 1,2;
331         CONST DIGMAX = 15; MAX8 = 34359738367;
332               MAX10 = 34359738367;
333         VAR I,K,SCALE,EXP,IVAL: INTEGER;
334             RVAL,R,FAC: REAL; SIGN: BOOLEAN;
335             DIGIT: ARRAY [1..DIGMAX] OF 0..9;
336             STRING: ARRAY [1..STRGLGTH] OF CHAR;
337             LVP: CSP;
338
339         PROCEDURE NEXTCH;
340         BEGIN READ(CH);
341           IF CH # ECL THEN
```

```
342             BEGIN
343               CHCNT := CHCNT + 1;
344               LINE¢CHCNT¡ := CH
345             END;
346         END /*NEXTCH*/ ;
347
348         PROCEDURE OPTIONS;
349         BEGIN
350           REPEAT NEXTCH;
351             IF CH = 'A' THEN
352               BEGIN NEXTCH; ATTRLIST := CH = '+' END
353             ELSE
354             IF CH = 'S' THEN
355               BEGIN NEXTCH; LIST := CH = '+' END
356             ELSE
357             IF CH = 'C' THEN
358               BEGIN NEXTCH; PRCODE := CH = '+' END
359             ELSE
360             IF CH = 'T' THEN
361               BEGIN NEXTCH; TRACE := CH = '+' END
362             ELSE
363             IF CH = 'L' THEN
364               BEGIN NEXTCH; PRSYMB := CH = '+' END;
365             IF CH = EOL THEN ENDOFLINE;
366             NEXTCH
367           UNTIL CH # ',';
368         END /*OPTIONS*/ ;
369
370       BEGIN /*INSYMBOL*/
371       1:
372         LOOP WHILE CH = ' ' DO NEXTCH;
373         EXIT IF CH # EOL;
374           ENDOFLINE; NEXTCH
375         END;
376         CASE CH OF                              /*IDENTIFIERS*/
377           'A','B','C','D','E','F','G','H','I',
378           'J','K','L','M','N','O','P','Q','R',
379           'S','T','U','V','W','X','Y','Z':
380             BEGIN K := 0;
381               REPEAT
382                 IF K < ALFALENG THEN
383                   BEGIN K := K + 1; ID¢K¡ := CH END ;
384                 NEXTCH
385               UNTIL ¬(CH IN LETTERS OR DIGITS );
386               IF K >= KK THEN KK := K
387               ELSE
388                 REPEAT ID¢KK¡ := ' '; KK := KK - 1
389                 UNTIL KK = K;
390               FOR I := FRW¢K¡ TO FRW¢K+1¡ - 1 DO
391                 IF RW¢I¡ = ID THEN
392                   BEGIN SY := RSY¢I¡; OP := ROP¢I¡; GOTO 2 END;
393               FOR I := 1 TO 6 DO
394                  IF PSSY¢I¡ = ID THEN
395                    BEGIN SY := RELOP;  OP := PSOP¢I¡;  GOTO 2  END;
396               SY := IDENT; OP := NOOP;
397       2:    END;
398           '0','1','2','3','4','5','6','7','8','9':   /*NUMBERS*/
```

```
399             BEGIN SY := FIXCONST; OP := NOOP;
400               I := 0;
401               REPEAT I := I + 1;
402                 IF I <= DIGMAX THEN DIGIT¢I! := ORD(CH) - ORD("0");
403                 NEXTCH
404               UNTIL ~(CH IN DIGITS);
405               IVAL := 0;
406               IF CH = "8" THEN              /*OCTAL*/
407                 BEGIN
408                   FOR K := 1 TO I DO
409                     IF IVAL <= MAX8 THEN
410                       BEGIN IF DIGIT¢K! IN ¢8,9! THEN ERROR(120);
411                         IVAL := 8*IVAL + DIGIT¢K!
412                       END
413                     ELSE BEGIN ERROR(121); IVAL := 0 END;
414                   VAL.IVAL := IVAL;
415                   NEXTCH
416                 END
417               ELSE
418                 BEGIN
419                   FOR K := 1 TO I DO
420                     IF IVAL <= MAX10 THEN
421                       IVAL := 10*IVAL + DIGIT¢K!
422                     ELSE BEGIN ERROR(121); IVAL := 0 END;
423                   SCALE := 0;
424                   IF CH = "." THEN
425                     BEGIN NEXTCH;
426                       IF CH = "." THEN CH := ":"
427                       ELSE
428                         BEGIN RVAL := IVAL; SY := FLTCONST;
429                           IF ~(CH IN DIGITS) THEN ERROR(122)
430                           ELSE
431                             REPEAT RVAL := 10.0*RVAL + (ORD(CH) - ORD("0"));
432                               SCALE := SCALE - 1; NEXTCH
433                             UNTIL ~(CH IN DIGITS)
434                         END
435                     END;
436                   IF CH = "E" THEN
437                     BEGIN
438                       IF SCALE = 0 THEN
439                         BEGIN RVAL := IVAL; SY := FLTCONST END;
440                       SIGN := FALSE; NEXTCH;
441                       IF CH = "+" THEN NEXTCH
442                       ELSE
443                         IF CH = "-" THEN
444                           BEGIN SIGN := TRUE; NEXTCH END;
445                       EXP := 0;
446                       IF ~(CH IN DIGITS) THEN ERROR(122)
447                       ELSE
448                         REPEAT EXP := 10*EXP + (ORD(CH) - ORD("0"));
449                           NEXTCH
450                         UNTIL ~(CH IN DIGITS);
451                       IF SIGN THEN SCALE := SCALE - EXP
452                       ELSE SCALE := SCALE + EXP
453                     END;
454                   IF SCALE # 0 THEN
455                     BEGIN R := 1.0;   /*NOTE POSSIBLE OVERFLOW OR UNDERFLOW*/
```

```
456                     IF SCALE < 0 THEN
457                       BEGIN FAC := 0.1; SCALE := -SCALE END
458                     ELSE FAC := 10.0;
459                     REPEAT IF ODD(SCALE) THEN R := R*FAC;
460                       FAC := SQR(FAC); SCALE := SCALE DIV 2
461                     UNTIL SCALE = 0;   (*NOW R = 10!SCALE*)
462                     RVAL := RVAL*R
463                   END;
464                 IF SY = FIXCONST THEN VAL.IVAL := IVAL
465                 ELSE
466                   BEGIN NEW(LVP,REEL);
467                     LVP!.RVAL := RVAL; VAL.VALP := LVP
468                   END
469               END
470           END;
471         '"':                              (*CHARACTER STRINGS*)
472           BEGIN LGTH := 0; SY := STRINGCONST;  OP := NOOP;
473             REPEAT
474           REPEAT NEXTCH; LGTH := LGTH + 1; STRING[LGTH] := CH
475               UNTIL (CH = EOL) OR (CH = '"');
476               IF CH = EOL THEN ERROR(125) ELSE NEXTCH
477             UNTIL CH # '"';
478             LGTH := LGTH - 1;   (*NOW LGTH = NR OF CHARS IN STRING*)
479             IF LGTH = 1 THEN VAL.IVAL := ORD(STRING[1])
480             ELSE
481               BEGIN NEW(LVP,STRG:LGTH);
482                 WITH LVP! DO
483                   BEGIN SLGTH := LGTH;
484                     FOR I := 1 TO LGTH DO SVAL[I] := STRING[I]
485                   END;
486                 VAL.VALP := LVP
487               END
488           END;
489         ':':
490           BEGIN OP := NOOP; NEXTCH;
491             IF CH = '=' THEN
492               BEGIN SY := BECOMES; NEXTCH END
493             ELSE SY := COLON
494           END;
495         '.':
496           BEGIN OP := NOOP; NEXTCH;
497             IF CH = '.' THEN
498               BEGIN SY := COLON; NEXTCH END
499             ELSE SY := PERIOD
500           END;
501         '/':
502           BEGIN NEXTCH;
503             IF CH = '*' THEN            /*   COMMENTS   */
504               BEGIN NEXTCH;
505                 IF CH = '$' THEN OPTIONS;
506                 REPEAT
507                   LOOP IF CH = EOL THEN ENDOFLINE;
508                   EXIT IF CH = '*';
509                     NEXTCH
510                   END;
511                   NEXTCH
512                 UNTIL CH = '/';
```

```
513                   NEXTCH; GOTO 1
514                 END;
515               SY := MULOP; OP := RDIV
516             END;
517           '-':
518             BEGIN NEXTCH;
519               IF CH = '>' THEN
520                 BEGIN SY := ARROW; OP := NOOP; NEXTCH END
521               ELSE
522                 BEGIN SY := ADDOP; OP := MINUS END;
523             END;
524           '<':
525             BEGIN NEXTCH; SY := RELOP;
526               IF CH = '=' THEN
527                 BEGIN OP := LEOP; NEXTCH END
528               ELSE
529                 IF CH = '>' THEN
530                   BEGIN OP := NEOP; NEXTCH END
531                 ELSE OP := LTOP
532             END;
533           '>':
534             BEGIN NEXTCH; SY := RELOP;
535               IF CH = '=' THEN
536                 BEGIN OP := GEOP; NEXTCH END
537               ELSE OP := GTOP
538             END;
539           '¬':
540             BEGIN NEXTCH; SY := RELOP;
541               IF CH = '='  THEN
542                   BEGIN OP := NEOP; NEXTCH END
543               ELSE IF CH = '>'  THEN
544                   BEGIN OP := LEOP; NEXTCH END
545               ELSE IF CH = '<'  THEN
546                   BEGIN OP := GEOP; NEXTCH END
547               ELSE OP := NOOP
548             END;
549           '|':
550             BEGIN NEXTCH; OP := NOOP;
551               IF CH = '|'  THEN
552                   BEGIN SY := CATSY; NEXTCH END
553               ELSE SY := RBRACK;
554             END;
555           '&','+','*',
556           '#','=','(',')',
557           '¢',',',';',':','!':
558             BEGIN SY := SSY¢CH1; OP := SOP¢CH1;
559               NEXTCH
560             END;
561           '$':
562             BEGIN SY := ENDPROG; OP := NOOP; CHCNT := CHCNT - 1
563             END
564         END /*CASE*/
565       END /*INSYMBOL*/ ;
566
567       PROCEDURE ENTERID(FIP: IDP);
568         /*ENTER ID POINTED AT BY FIP INTO THE NAME-TABLE,
569           WHICH ON EACH DECLARATION LEVEL IS ORGANISED AS
```

```
570           AN UNBALANCED BINARY TREE*/
571         VAR NAM: ALFA; LIP, LIP1: IDP; LLEFT: BOOLEAN;
572       BEGIN NAM := FIP!.NAME;
573         LIP := IDTREE¢TOP!;
574         IF LIP = NIL THEN IDTREE¢TOP! := FIP
575         ELSE
576           BEGIN
577             REPEAT LIP1 := LIP;
578               IF LIP!.NAME = NAM THEN   /*NAME CONFLICT, FOLLOW RIGHT LINK*/
579                 BEGIN
580                   IF LIP!.NEXT # LIP THEN ERROR(130)
581                   ELSE
582                   IF LIP!.NEXT = FIP!.NEXT THEN ERROR(130);
583                   LIP := LIP!.RLINK; LLEFT := FALSE ;
584                 END
585               ELSE
586                 IF LIP!.NAME < NAM THEN
587                   BEGIN LIP := LIP!.RLINK; LLEFT := FALSE END
588                 ELSE BEGIN LIP := LIP!.LLINK; LLEFT := TRUE END
589             UNTIL LIP = NIL;
590             IF LLEFT THEN LIP1!.LLINK := FIP ELSE LIP1!.RLINK := FIP
591           END;
592         FIP!.LLINK := NIL; FIP!.RLINK := NIL
593       END /*ENTERID*/ ;
594
595       PROCEDURE SEARCHSECTION(FIP: IDP; VAR FIP1: IDP);
596             /*FIND PREVIOUSLY DECLARED ENTRY NAMES
597          --> PROCEDURE PROCEDUREDECLARATION, PROCEDUREDEFINE*/
598           LABEL 1;
599       BEGIN
600         WHILE FIP # NIL DO
601           IF FIP!.NAME = ID THEN
602             IF FIP!.NEXT = FIP THEN FIP := FIP!.RLINK ELSE GOTO 1
603           ELSE IF FIP!.NAME < ID THEN FIP := FIP!.RLINK
604             ELSE FIP := FIP!.LLINK;
605     1:  FIP1 := FIP
606       END /*SEARCHSECTION*/ ;
607
608       PROCEDURE SEARCHID(FIDCLS: SETOFIDS; VAR FIP: IDP);
609             /*SEARCH FOR AN IDENTIFIER IN THE SYMBOL TABLE*/
610         LABEL 1;
611         VAR LIP: IDP;
612       BEGIN
613         FOR TREEX := TOP DOWNTO 0 DO
614           BEGIN LIP := IDTREE¢TREEX!;
615             WHILE LIP # NIL DO
616               IF LIP!.NAME = ID THEN
617                 BEGIN
618                   IF LIP!.NEXT = LIP THEN LIP := LIP!.RLINK
619                   ELSE
620                   IF LIP!.KLASS IN FIDCLS THEN
621                     BEGIN
622                       IF LIP!.KLASS # FIELD THEN GOTO 1
623                       ELSE
624                       IF RECIDP = NIL THEN GOTO 1
625                       ELSE
626                       IF LIP!.NEXT = RECIDP THEN GOTO 1
```

```
627                       ELSE LIP := LIP!.RLINK
628                     END
629                   ELSE
630                     BEGIN IF PRTERR THEN ERROR(131);
631                       LIP := LIP!.RLINK
632                     END
633                 END
634               ELSE
635                 IF LIP!.NAME < ID THEN
636                   LIP := LIP!.RLINK
637                 ELSE LIP := LIP!.LLINK
638           END;
639         /*SEARCH NOT SUCCESSFUL; SUPPRESS ERROR MESSAGE IN CASE
640          OF FORWARD REFERENCED TYPE ID IN POINTER TYPE DEFINITION
641          --> PROCEDURE SIMPLETYPE,
642           OR LABEL CONSTANT      --> PROCEDURE GOTOSTATEMENT*/
643         IF PRTERR THEN
644           BEGIN ERROR(132);
645             /*TO AVOID RETURNING NIL, REFERENCE AN ENTRY
646              FOR AN UNDECLARED ID OF APPROPRIATE CLASS
647              --> PROCEDURE ENTERUNDECL*/
648             IF TYPES IN FIDCLS THEN LIP := UTYPPTR
649             ELSE
650               IF VARS IN FIDCLS THEN LIP := UVARPTR
651               ELSE
652                 IF FIELD IN FIDCLS THEN LIP := UFLDPTR
653                 ELSE
654                   IF KONST IN FIDCLS THEN LIP := UCSTPTR
655                   ELSE
656                     IF PROC IN FIDCLS THEN LIP := UPRCPTR
657                     ELSE LIP := UFIDPTR;
658           END;
659       1:  FIP := LIP
660       END /*SEARCHID*/ ;
661
662       PROCEDURE GETBOUNDS(FSP: STP; VAR FMIN,FMAX: INTEGER);
663         /*GET INTERNAL BOUNDS OF SUBRANGE OR SCALAR TYPE*/
664         /*ASSUME (FSP # NIL)&(FSP!.FORM <= SUBRANGE)&(FSP # FIXPTR)
665          &^COMPTYPES(FLIPTR,FSP)*/
666       BEGIN
667         WITH FSP! DO
668           IF FORM = SUBRANGE THEN
669             BEGIN FMIN := MIN.IVAL; FMAX := MAX.IVAL END
670           ELSE
671             BEGIN FMIN := 0;
672               IF FSP = CHARPTR THEN FMAX := 63
673               ELSE
674                 IF FSP!.FCONST # NIL THEN
675                   FMAX := FSP!.FCONST!.VALUES.IVAL
676                 ELSE FMAX := 0
677             END
678       END /*GETBOUNDS*/ ;
679
680       PROCEDURE ATTRIBUTELIST(FB: BOOLEAN);
681             /*LIST IDENTIFIERS AND DATA STRUCTURES*/
682         VAR I, LIM: TREERANGE;
683
```

```
684         PROCEDURE MARK;
685           /*MARK DATA STRUCTURE ENTRIES TO AVOID MULTIPLE PRINTOUT*/
686           VAR I: INTEGER;
687
688           PROCEDURE MARKSTP(FP: STP);
689             /*MARK DATA STRUCTURES, PREVENT CYCLES*/
690           BEGIN
691             IF FP # NIL THEN
692               WITH FP! DO
693                 BEGIN MARKED := TRUE;
694                   CASE FORM OF
695                   SCALAR:   ;
696                   SUBRANGE: MARKSTP(RANGETYPE);
697                   POINTER:  /*DON'T MARK ELTYPE: CYCLE POSSIBLE; WILL BE MARKED
698                             ANYWAY, IF FP = TRUE*/ ;
699                   SETS:     MARKSTP(ELSET) ;
700                   ARRAYS:   BEGIN MARKSTP(AELTYPE); MARKSTP(INXTYPE) END
701                   END /*CASE*/
702                 END /*WITH*/
703           END /*MARKSTP*/;
704
705           PROCEDURE MARKIDP(FP: IDP);
706           BEGIN
707             IF FP # NIL THEN
708               WITH FP! DO
709                 BEGIN MARKIDP(LLINK); MARKIDP(RLINK);
710                   MARKSTP(IDTYPE)
711                 END
712           END /*MARKIDP*/;
713
714         BEGIN /*MARK*/
715           FOR I := TOP DOWNTO LIM DO MARKIDP(IDTREE@[I])
716         END  /*MARK*/;
717
718         PROCEDURE FOLLOWSTP(FP: STP);
719         BEGIN
720           IF FP # NIL THEN
721             WITH FP! DO
722               IF MARKED THEN
723                 BEGIN MARKED := FALSE; WRITE(' ':4,ORD(FP):6,SIZE:10);
724                   CASE FORM OF
725                   SCALAR:   BEGIN WRITE('SCALAR':10);
726                               IF SCALKIND = STANDARD THEN WRITE('STANDARD':10)
727                               ELSE WRITE('DECLARED':10,' ':4,ORD(FCONST):6);
728                               WRITE(EOL)
729                             END;
730                   SUBRANGE: BEGIN WRITE('SUBRANGE':10,' ':4,ORD(RANGETYPE):6);
731                               IF RANGETYPE # FLTPTR THEN
732                                 WRITE(MIN.IVAL,MAX.IVAL)
733                               ELSE
734                                 IF (MIN.VALP # NIL)&(MAX.VALP # NIL) THEN
735                                   WRITE(' ',MIN.VALP!.RVAL:9,
736                                         ' ',MAX.VALP!.RVAL:9);
737                               WRITE(ECL); FOLLOWSTP(RANGETYPE);
738                             END;
739                   POINTER:  WRITE('POINTER':10,' ':4,ORD(ELTYPE):6,ECL);
740                   SETS:     BEGIN WRITE('SET':10,' ':4,ORD(ELSET):6,EOL);
```

```
                                FOLLOWSTP(ELSET)
                              END;
                ARRAYS:    BEGIN WRITE('ARRAY':10,' ':4,ORD(AELTYPE):6,' ':4,
                              ORD(INXTYPE):6,EOL);
                              FOLLOWSTP(AELTYPE); FOLLOWSTP(INXTYPE)
                            END;
                RECORDS:   WRITE('RECORD':10);
                FILES:     WRITE('FILE':10,' ':4,FILUNIT:2,EOL)
                END /*CASE*/
              END /*IF MARKED*/
          END /*FOLLOWSTP*/;

          PROCEDURE FOLLOWIDP(FP: IDP);
            VAR I: INTEGER;
          BEGIN
            IF FP # NIL THEN
              WITH FP! DO
                BEGIN WRITE(' ':4,ORD(FP):6,' ',NAME:9,' ':4,ORD(LLINK):6,
                  ' ':4,ORD(RLINK):6,' ':4,ORD(IDTYPE):6);
                  CASE KLASS OF
                    TYPES: WRITE('TYPE':10);
                    KONST: BEGIN WRITE('CONSTANT':10,' ':4,ORD(NEXT):6);
                           IF IDTYPE # NIL THEN
                               IF IDTYPE = FLTPTR THEN
                                 BEGIN
                                   IF VALUES.VALP # NIL THEN
                                     WRITE(' ',VALUES.VALP!.RVAL:9)
                                 END
                               ELSE
                                 IF IDTYPE!.FORM = ARRAYS THEN  /*STRINGCONST*/
                                   BEGIN
                                     IF VALUES.VALP # NIL THEN
                                       BEGIN WRITE(' ');
                                         WITH VALUES.VALP! DO
                                     FOR I := 1 TO SLGTH DO WRITE(SVAL@I])
                                         END
                                   END
                                 ELSE WRITE(VALUES.IVAL)
                             END;
                    VARS:  BEGIN WRITE('VARIABLE':10);
                             IF VKIND = ACTUAL THEN WRITE('ACTUAL':10)
                             ELSE WRITE('FORMAL':10);
                             WRITE(' ':4,ORD(NEXT):6,VLEV,' ':4,VADDR:6 );
                           END;
                    FIELD: WRITE('FIELD':10,' ':4,ORD(NEXT):6,' ':4,FLDADDR:6);
                    PROC,
                    FUNC:  BEGIN
                             IF KLASS = PROC THEN WRITE('PROCEDURE':10)
                             ELSE WRITE('FUNCTION':10);
                             IF PFDECKIND = STANDARD THEN WRITE('STANDARD':10,
                               KEY:10)
                             ELSE
                               BEGIN WRITE('DECLARED':10,' ':4,ORD(NEXT):6);
                                 WRITE(PFLEV,' ':4,PFADDR:6);
                                 IF PFKIND = ACTUAL THEN
                                   BEGIN WRITE('ACTUAL':10);
                                     IF FORWDECL THEN WRITE('FORWARD':10)
```

```
798                         ELSE WRITE('¬FORWARD':10);
799                                 IF EXTERN THEN WRITE('EXTERN':10)
800                                 ELSE WRITE('¬EXTERN':10);
801                               END
802                             ELSE WRITE('FORMAL':10)
803                           END
804                       END
805               END /*CASE*/;
806               WRITE(EOL); FOLLOWIDP(LLINK); FOLLOWIDP(RLINK);
807               FOLLOWSTP(IDTYPE)
808             END /*WITH*/
809         END /*FOLLOWIDP*/;
810
811       BEGIN /*ATTRIBUTELIST*/
812         WRITE(EOL,EOL,EOL);
813         IF FB THEN LIM := 0
814         ELSE BEGIN LIM := TOP; WRITE(' LOCAL') END;
815         WRITE(' TABLES ',EOL,EOL);
816         MARK;
817         FOR I := TOP DOWNTO LIM DO FOLLOWIDP(IDTREE¢I);
818         WRITE(EOL);
819         IF CH # EOL THEN WRITE(' ':CHCNT+8)
820       END /*ATTRIBUTELIST*/;
821
822       PROCEDURE BLOCK(FSYS: SYSET; FBTYP:BLKTYPE; FSY: SYMBOL; FPROCP: IDP);
823             /*PROCESS A PL/I BLOCK*/
824         VAR MARKMARK: INTEGER; FLABP: LBP; LINITP,FSTINITP,LSTINITP: INITP;
825
826         PROCEDURE BEGBLOCK(FBTYPE:BLKTYPE);
827             /*MAINTAIN BLOCK INFORMATION AND PRINT
828               LEVELS ON SOURCE LINE AT THE
829               BEGINNING OF EACH BLOCK*/
830         BEGIN
831           BLMAX := BLMAX + 1; BIX := BIX + 1;
832           BLOCKS¢BIX!.BEGLEV := BLMAX;
833           IF FBTYPE # BEGINBLK THEN BLOCKS¢BIX!.BNAME := LLABEL
834           ELSE BLOCKS¢BIX!.BNAME := '          ';
835           BLOCKS¢BIX!.BTYPE := FBTYPE; BDELIM := TRUE
836         END /*BEGBLOCK*/;
837
838         PROCEDURE ENDBLOCK(FBTYPE:BLKTYPE);
839             /*BLOCK HOUSEKEEPING AND LEVEL-PRINTING
840                 AT THE END OF EACH BLOCK*/
841           VAR I,J: INTEGER;
842         BEGIN
843           INSYMBOL;
844           IF FBTYPE # BEGINBLK THEN
845             BEGIN
846             WITH BLOCKS¢BIX! DO
847               IF SY = IDENT THEN
848                 BEGIN
849                   IF (BNAME = ID) AND (BTYPE = FBTYPE) THEN BIX := BIX - 1
850                   ELSE
851                     BEGIN
852                       REPEAT BIX := BIX - 1; ERROR(600)
853                       UNTIL (BNAME = ID) OR (BTYPE =FBTYPE);
854                       INSYMBOL; BIX := BIX - 1
```

```
855                   END;
856                 INSYMBOL
857               END
858             ELSE
859               BEGIN J := 0;
860                 IF BNAME # '           ' THEN
861                   BEGIN
862                     FOR I := CHCNT + 1 TO CHCNT + ALFALENG DO
863                       BEGIN J := J + 1; LINE^[I] := BNAME^[J] END;
864                     CHCNT := CHCNT + ALFALENG
865                   END;
866                 BIX := BIX - 1
867               END;
868           END
869         ELSE BIX := BIX - 1;
870         EOELIM := TRUE; ENDLEV := BLOCKS^[BIX+1].BEGLEV
871       END /*ENDBLOCK*/;
872
873       PROCEDURE SKIP(FSYS: SYSET);
874         /*SKIP INPUT STRING UNTIL RELEVANT SYMBOL FOUND*/
875       BEGIN WHILE ~(SY IN FSYS) DO INSYMBOL END;
876
877       PROCEDURE ERRSKIP(ERRNR: INTEGER; FSYS: SYSET);
878       BEGIN ERROR(ERRNR); SKIP(FSYS) END;
879
880       PROCEDURE CONSTANT(FSYS: SYSET; VAR FSP: STP; VAR FVALU: VALU);
881            /*SET UP THE DATA STRUCTURES FOR STRINGS AND CONSTANTS*/
882         VAR LSP: STP; LIP: IDP; SIGN: (NONE,POS,NEG);
883       BEGIN LSP := NIL; FVALU.IVAL := 0;
884         IF ~(SY IN CONSTBEGSYS) THEN ERRSKIP(123,FSYS OR CONSTBEGSYS);
885         IF SY IN CONSTBEGSYS THEN
886           BEGIN
887             IF SY = STRINGCONST THEN
888               BEGIN
889                 IF LGTH = 1 THEN LSP := CHARPTR
890                 ELSE
891                   BEGIN
892                     NEW(LSP,ARRAYS);
893                     WITH LSP^ DO
894                       BEGIN AELTYPE := CHARPTR; INXTYPE := NIL; SIZE := LGTH
895                       END
896                   END;
897                 FVALU := VAL; INSYMBOL
898               END
899             ELSE
900               BEGIN
901                 SIGN := NONE;
902                 IF (SY = ADDOP)&(OP IN [PLUS,MINUS]) THEN
903                   BEGIN IF OP = PLUS THEN SIGN := POS ELSE SIGN := NEG;
904                     INSYMBOL
905                   END;
906                 IF SY = IDENT THEN
907                   BEGIN SEARCHID([KONST],LIP);
908                     WITH LIP^ DO
909                       BEGIN LSP := IDTYPE; FVALU := VALUES END;
910                     IF SIGN # NONE THEN
911                       IF LSP = FIXPTR THEN
```

```
            IF FSP1!.FORM = SUBRANGE THEN
              COMPTYPES := COMPTYPES(FSP1!.RANGETYPE,FSP2)
            ELSE
              IF FSP2!.FORM = SUBRANGE THEN
                COMPTYPES := COMPTYPES(FSP1,FSP2!.RANGETYPE)
              ELSE COMPTYPES := FALSE
          ELSE COMPTYPES := TRUE
      END /*COMPTYPES*/ ;

      FUNCTION STRING(FSP: STP) : BOOLEAN;
      BEGIN STRING := FALSE;
        IF FSP # NIL THEN
          IF FSP!.FORM = ARRAYS THEN
            IF COMPTYPES(FSP!.AELTYPE,CHARPTR) THEN STRING := TRUE
      END /*STRING*/ ;


    PROCEDURE TYP(FSYS:SYSET;VAR FSP:STP;VAR FSIZE:ADDRRANGE;VAR FINITP:INITP);
          /*ANALYZE TYPES AND ATTRIBUTES
            AND SET UP THEIR STRUCTURE CELLS*/
        VAR LSP,LSP1,LSP2: STP; LINITP: INITP; LIP: IDP; LVP:VP;
            LSIZE: ADDRRANGE; LMIN,LMAX: INTEGER;

        FUNCTION REPEATCOUNT: INTEGER;
           /*USED FOR THE INITIAL ATTRIBUTE*/
          VAR LSP: STP; LVAL: VALU;
        BEGIN
          REPEATCOUNT := 1;
          IF SY = LPARENT THEN
            BEGIN INSYMBOL;
              IF SY = MULOP THEN REPEATCOUNT := 0   /*IE. (*) */
              ELSE
                BEGIN CONSTANT(FSYS OR [RPARENT],LSP,LVAL);
                  IF LSP # FIXPTR THEN ERROR(126)
                  ELSE IF LVAL.IVAL < 1 THEN ERROR(126)
                  ELSE REPEATCOUNT := LVAL.IVAL
                END;
              IF SY = RPARENT THEN INSYMBOL ELSE ERROR(24)
            END;
        END /*REPEATCOUNT*/ ;

        PROCEDURE SIMPLETYPE(FSYS: SYSET; VAR FSP: STP; VAR FSIZE: ADDRRANGE);
          VAR LSP,LSP1: STP; LIP,LIP1: IDP; LCNT: INTEGER; LVALU: VALU;
        BEGIN FSIZE := 1;
          IF ~(SY IN SIMPTYPEBEGSYS) THEN ERRSKIP(101,FSYS OR SIMPTYPEBEGSYS);
          IF SY IN SIMPTYPEBEGSYS THEN
            BEGIN
              IF SY = LBRACK THEN
                BEGIN NEW(LSP,SCALAR,DECLARED);
                  LSP!.SIZE := 1;
                  LIP1 := NIL; LCNT := 0;
                  REPEAT INSYMBOL;
                    IF SY = IDENT THEN
                      BEGIN NEW(LIP,KONST);
                        WITH LIP! DO
                          BEGIN NAME := ID; IDTYPE := LSP; NEXT := LIP1;
                            VALUES.IVAL := LCNT;
```

```
1026                          END;
1027                        ENTERID(LIP);
1028                        LCNT := LCNT + 1;
1029                        LIP1 := LIP; INSYMBOL
1030                      END
1031                    ELSE ERROR(102);
1032                    IF ~(SY IN FSYS OR LISTDELIMS) THEN
1033                      ERRSKIP(10,FSYS OR LISTDELIMS)
1034                  UNTIL SY # COMMA;
1035                  LSP!.FCONST := LIP1;
1036                  IF SY = RBRACK THEN INSYMBOL ELSE ERROR(22)
1037                END
1038              ELSE IF (SY IN SVATTRSYS) THEN
1039                BEGIN LSP1 := NIL; VBASE := NULL;
1040                  WHILE (SY IN SVATTRSYS) DO
1041                    BEGIN
1042                      IF SY = FIXEDSY THEN LSP1 := FIXPTR
1043                      ELSE
1044                      IF SY = FLOATSY THEN LSP1 := FLTPTR
1045                      ELSE
1046                      IF SY = BINARYSY THEN VBASE := BINARY
1047                      ELSE
1048                      IF SY = DECIMALSY THEN VBASE := DECIMAL
1049                      ELSE
1050                      IF SY = LIKESY THEN
1051                        BEGIN INSYMBOL; SEARCHID(¢VARS!,LIP);
1052                          LSP1 := LIP!.IDTYPE
1053                        END
1054                      ELSE
1055                      IF SY = INITIALSY THEN
1056                       IF VBASED THEN ERRSKIP(106,¢RPARENT!)
1057                       ELSE
1058                        BEGIN INSYMBOL; FINITP := LSTINITP;
1059                          IF SY = LPARENT THEN INSYMBOL ELSE ERROR(25);
1060                          LOOP NEW(LINITP);
1061                          WITH LINITP! DO
1062                            BEGIN NEXTINIT := NIL; REPEATF := REPEATCOUNT;
1063                              CONSTANT(FSYS OR LISTDELIMS,LSP,LVALU);
1064                              INITVAL := LVALU;
1065                              LSTINITP!.NEXTINIT := LINITP; LSTINITP := LINITP;
1066                            END;
1067                          EXIT IF SY # COMMA;
1068                            INSYMBOL
1069                          END;
1070                          IF SY # RPARENT THEN ERROR(24)
1071                        END /*SY = INITIALSY*/
1072                      ELSE
1073                      IF SY = BASEDSY THEN VBASED := TRUE
1074                      ELSE
1075                      IF (SY IN ¢AUTOSY,STATICSY,EXTERNSY!) THEN ERROR(403);
1076                      INSYMBOL;
1077                    END;
1078                  LSP := LSP1;
1079                  IF LSP # NIL THEN FSIZE := LSP!.SIZE;
1080                END  /*SY IN SVATTRSYS*/
1081              ELSE
1082                BEGIN
```

```
1083                    IF SY = IDENT THEN
1084                      BEGIN SEARCHID([TYPES,KONST],LIP);
1085                       IF LIP!.IDTYPE # NIL THEN
1086                       BEGIN INSYMBOL;
1087                        IF LIP!.KLASS = KONST THEN
1088                          BEGIN NEW(LSP,SUBRANGE);
1089                            WITH LSP!, LIP! DO
1090                              BEGIN RANGETYPE := IDTYPE;
1091                                IF STRING(RANGETYPE) THEN
1092                                 BEGIN ERROR(154); RANGETYPE := NIL END;
1093                                MIN := VALUES; SIZE := 1
1094                              END;
1095                            IF SY = COLON THEN
1096                              BEGIN INSYMBOL;
1097                                CONSTANT(FSYS,LSP1,LVALU);
1098                                LSP!.MAX := LVALU;
1099                                IF LSP!.RANGETYPE # LSP1 THEN ERROR(151)
1100                              END
1101                            ELSE
1102                              BEGIN INSYMBOL;
1103                                LSP!.MAX := LSP!.MIN;
1104                                LSP!.MIN.IVAL := 1
1105                              END;
1106                        END
1107                       ELSE
1108                         BEGIN LSP := LIP!.IDTYPE;
1109                           IF LSP # NIL THEN FSIZE := LSP!.SIZE
1110                         END
1111                       END  /*IDTYPE # NIL*/
1112                      END /*SY = IDENT*/
1113                    ELSE
1114                      BEGIN NEW(LSP,SUBRANGE);
1115                        CONSTANT(FSYS OR [COLON],LSP1,LVALU);
1116                        IF STRING(LSP1) THEN
1117                          BEGIN ERROR(154); LSP1 := NIL END;
1118                        WITH LSP! DO
1119                          BEGIN RANGETYPE := LSP1; MIN := LVALU; SIZE := 1 END;
1120                        IF SY = COLON THEN
1121                          BEGIN INSYMBOL;
1122                            CONSTANT(FSYS,LSP1,LVALU);
1123                            LSP!.MAX := LVALU;
1124                            IF LSP!.RANGETYPE # LSP1 THEN ERROR(151)
1125                          END
1126                        ELSE
1127                          BEGIN
1128                            LSP!.MAX := LSP!.MIN;
1129                            LSP!.MIN.IVAL := 1
1130                          END;
1131                      END;
1132                    IF LSP # NIL THEN
1133                      WITH LSP! DO
1134                        IF FORM = SUBRANGE THEN
1135                          IF RANGETYPE # NIL THEN
1136                            IF RANGETYPE = FLTPTR THEN
1137                              BEGIN
1138                                IF MIN.VALP!.RVAL > MAX.VALP!.RVAL THEN ERROR(152)
1139                              END
```

```
1140                          ELSE
1141                            IF MIN.IVAL > MAX.IVAL THEN ERROR(152)
1142                    END;
1143                  FSP := LSP;
1144                  IF ~(SY IN FSYS) THEN ERRSKIP(10,FSYS)
1145                END
1146                  ELSE FSP := NIL
1147              END /*SIMPLETYPE*/ ;
1148
1149              PROCEDURE FIELDLIST(VAR FSIZE: ADDRRANGE; FIP: IDP);
1150                 /*ANALYZE A RECORD DECLARATION
1151                   AND SET UP THE FIELD CELL STRUCTURES*/
1152                VAR LIP: IDP; LSP,LSP1,LSP2: STP;
1153                    DISPL,LSIZE,LSIZE1: ADDRRANGE; LEVL: INTEGER;
1154              BEGIN
1155                LEVL := VAL.IVAL; LSIZE := 0; DISPL := 0; LIP := NIL;
1156                LOOP
1157                  IF ~(SY = FIXCONST) THEN
1158                    ERRSKIP(200,FSYS OR [FIXCONST,COMMA,SEMICOLON]) ;
1159                  IF SY = FIXCONST THEN
1160                    IF VAL.IVAL >= LEVL THEN
1161                      BEGIN
1162                        IF VAL.IVAL > LEVL THEN
1163                          BEGIN FIELDLIST(LSIZE1,LIP);
1164                            IF LSIZE1 > LSIZE THEN LSIZE := LSIZE1;
1165                          END
1166                        ELSE
1167                        IF VAL.IVAL = LEVL THEN
1168                          BEGIN INSYMBOL;
1169                            DISPL := DISPL + LSIZE;
1170                            IF ~VBASED THEN GLC := GLC + LSIZE;
1171                            IF SY # IDENT THEN
1172                              ERRSKIP(10,FSYS OR [IDENT,COMMA,SEMICOLON]);
1173                            IF SY = IDENT THEN
1174                              BEGIN NEW(LIP,FIELD);
1175                                LIP!.NAME := ID;
1176                                IF (ID[1] < 'I') OR (ID[1] > 'N') THEN
1177                                  LIP!.BD := FLT ELSE LIP!.BD := FIX;
1178                                LIP!.IDTYPE := NIL;
1179                                LIP!.FLEV := LEVL;
1180                                LIP!.NEXT := FIP;
1181                                LIP!.FLDADDR := DISPL;
1182                                ENTERID(LIP);
1183                                INSYMBOL;
1184                                LSP := NIL; LSIZE := 1;
1185                                IF SY = LPARENT THEN
1186                                TYP(FSYS OR VATTRSYS OR LISTDELIMS,LSP,LSIZE,LINITP);
1187                                IF LSP # NIL THEN LIP!.IDTYPE := LSP;
1188                                VBASE := NULL;
1189                                IF ~(SY IN SEMICOMMA OR VATTRSYS) THEN
1190                                  ERRSKIP(10,FSYS OR SEMICOMMA);
1191                                IF (SY IN VATTRSYS) THEN
1192                                TYP(FSYS OR SEMICOMMA,LSP,LSIZE,LINITP);
1193                                IF LSP = NIL THEN
1194                                  BEGIN
1195                                    IF LIP!.BD = FIX THEN LSP := FIXPTR
1196                                    ELSE
```

```
1197                                IF LIP!.BD = FLT THEN LSP := FLTPTR
1198                              END;
1199                            IF ~(VBASE IN ¢BINARY,DECIMAL¡) THEN
1200                              BEGIN
1201                                IF LSP = FIXPTR THEN VBASE := BINARY
1202                                ELSE
1203                                IF LSP = FLTPTR THEN VBASE := DECIMAL
1204                              END;
1205                          IF (LSP = FIXPTR) OR (LSP = FLTPTR) THEN LIP!.BD := VBASE;
1206                              WITH LIP! DO
1207                                IF IDTYPE = NIL THEN IDTYPE := LSP
1208                                ELSE
1209                                  BEGIN LSP2 := IDTYPE;
1210                                      REPEAT LSP1 := LSP2;
1211                                      LSP1!.SIZE := LSP1!.SIZE*LSIZE;
1212                                      LSP2 := LSP1!.AELTYPE;
1213                                      UNTIL LSP2 = NIL;
1214                                      LSP1!.AELTYPE := LSP;
1215                                  END;
1216                              LSIZE := LIP!.IDTYPE!.SIZE;
1217                            END;
1218                          END;
1219                        IF ~(SY IN SEMICOMMA) THEN
1220                          ERRSKIP(10,FSYS OR ¢COMMA,FIXCONST,SEMICOLON¡)
1221                        END /*IF IVAL >= LEVL*/
1222                  EXIT IF (SY = SEMICOLON) OR (VAL.IVAL < LEVL);
1223                    INSYMBOL;
1224                  END;
1225                  FSIZE := DISPL + LSIZE;
1226                END /*FIELDLIST*/ ;
1227
1228              BEGIN /*TYP*/
1229                FINITP := LSTINITP;
1230                IF ~(SY IN TYPEBEGSYS OR SEMICOMMA) THEN
1231                   ERRSKIP(103,FSYS OR TYPEBEGSYS);
1232                IF SY IN TYPEBEGSYS THEN
1233                  IF SY IN SIMPTYPEBEGSYS THEN SIMPLETYPE(FSYS,FSP,FSIZE)
1234                  ELSE
1235                    BEGIN
1236           /*POINTER*/ IF SY = POINTERSY THEN
1237                        BEGIN NEW(LSP,POINTER); FSP := LSP;
1238                          WITH LSP! DO
1239                            BEGIN ELTYPE := NIL; SIZE := 1 END;
1240                          INSYMBOL;
1241                        END
1242                      ELSE
1243                 /*ARRAY*/     IF SY = LPARENT THEN
1244                               BEGIN
1245                                 LSP1 := NIL; INSYMBOL;
1246                                 LOOP NEW(LSP,ARRAYS);
1247                                   WITH LSP! DO
1248                                     BEGIN AELTYPE := LSP1; INXTYPE := NIL; SIZE := 1 END;
1249                                   LSP1 := LSP;
1250                                   SIMPLETYPE(FSYS OR LISTDELIMS,LSP2,LSIZE);
1251                                   IF LSP2 # NIL THEN
1252                                     BEGIN
1253                                       IF LSP2!.FORM <= SUBRANGE THEN
```

```
1254                                BEGIN
1255                                  IF LSP2 = FLTPTR THEN
1256                                    BEGIN ERROR(153); LSP2 := NIL END
1257                                  ELSE
1258                                    IF LSP2 = FIXPTR THEN
1259                                      BEGIN ERROR(153); LSP2 := NIL END;
1260                                  LSP!.INXTYPE := LSP2
1261                                END
1262                              ELSE BEGIN ERROR(154); LSP2 := NIL END;
1263                            END
1264                          ELSE
1265                            BEGIN SEARCHID(¢VARS¢,LIP);
1266                              NEW(LSP,SUBRANGE);
1267                              WITH LSP!, LIP! DO
1268                                BEGIN
1269                                  IF VKIND # FORMAL THEN ERROR(155);
1270                                  IF IDTYPE # FIXPTR THEN ERROR(156);
1271                                  ERROR(403);
1272                                  RANGETYPE := IDTYPE
1273                                END
1274                            END
1275                        EXIT IF SY # COMMA;
1276                          INSYMBOL
1277                        END;
1278                        IF SY = RPARENT THEN INSYMBOL ELSE ERROR(24);
1279                        LSP := NIL; LSIZE := 1;
1280                        REPEAT
1281                          WITH LSP1! DO
1282                            BEGIN LSP2 := AELTYPE; AELTYPE := LSP;
1283                              IF INXTYPE # NIL THEN
1284                                BEGIN GETBOUNDS(INXTYPE,LMIN,LMAX);
1285                                  LSIZE := LSIZE*(LMAX - LMIN + 1);
1286                                  SIZE := LSIZE
1287                                END
1288                            END;
1289                          LSP := LSP1; LSP1 := LSP2
1290                        UNTIL LSP1 = NIL
1291                      END
1292                    ELSE
1293      /*CHARACTER*/   IF SY = CHARSY THEN
1294                        BEGIN INSYMBOL;
1295                          IF SY = LPARENT THEN
1296                            BEGIN TYP(FSYS OR LISTDELIMS,LSP,LSIZE,LINITP);
1297                              WITH LSP! DO
1298                                BEGIN
1299                                  SIZE := LSIZE;
1300                                  IF AELTYPE = NIL THEN AELTYPE := CHARPTR
1301                                  ELSE ERROR(159);
1302                                END;
1303                            END
1304                          ELSE LSP := CHARPTR;
1305                          IF SY = INITIALSY THEN SIMPLETYPE(FSYS,FSP,FSIZE)
1306                        END
1307                      ELSE
1308      /*BIT*/           IF SY = BITSY THEN
1309                        BEGIN INSYMBOL; ERROR(403);
1310                          IF SY = LPARENT THEN
```

```
1311                              BEGIN TYP(FSYS OR LISTDELIMS,LSP,LSIZE,LINITP);
1312                                WITH LSP! DO
1313                                  IF AELTYPE = NIL THEN AELTYPE := BOOLPTR
1314                                  ELSE ERROR(159);
1315                              END
1316                            ELSE LSP := BOOLPTR
1317                          END
1318                        ELSE
1319          /*LABEL*/         IF SY = LABELSY THEN
1320                            BEGIN
1321                              NEW(LSP,LLABL); LSP!.SIZE := 1; FSP := LSP;
1322                              INSYMBOL
1323                            END
1324                          ELSE
1325          /*RECORD*/          IF SY = RECORDSY THEN
1326                            BEGIN
1327                              LSIZE := 0; SY := FIXCONST;
1328                              FIELDLIST(LSIZE,RECIDP);
1329                              NEW(LSP,RECORDS);
1330                              LSP!.SIZE := LSIZE;
1331                              IF SY # SEMICOLON THEN ERROR(27)
1332                            END
1333                          ELSE
1334          /*SET*/               IF SY = SETSY THEN
1335                              BEGIN INSYMBOL;
1336                                IF SY = OFSY THEN INSYMBOL ELSE ERROR(16);
1337                                SIMPLETYPE(FSYS,LSP1,LSIZE);
1338                                IF LSP1 # NIL THEN
1339                                  IF LSP1!.FORM > SUBRANGE THEN
1340                                    BEGIN ERROR(170); LSP1 := NIL END
1341                                  ELSE
1342                                    IF LSP1 = FLTPTR THEN ERROR(171);
1343                                NEW(LSP,SETS);
1344                                WITH LSP! DO
1345                                  BEGIN ELSET := LSP1; SIZE := 2 END;
1346                              END
1347                            ELSE
1348          /*FILE*/              IF SY = FILESY THEN
1349                              BEGIN INSYMBOL;
1350                                IF ?(SY IN SEMICOMMA) THEN
1351                                  SKIP(SEMICOMMA);
1352                                FILENO := FILENO + 1;
1353                                NEW(LSP,FILES);
1354                                WITH LSP! DO
1355                                  BEGIN LOFADDR := 0;
1356                                    FILUNIT := FILENO; SIZE := BUFFSIZE
1357                                  END;
1358                                NEW(LINITP); FINITP := LSTINITP;
1359                                WITH LINITP! DO
1360                                  BEGIN NEXTINIT := NIL; TYPOINTER := FIXPTR;
1361                                    INITVAL.IVAL := FILENO;
1362                                IF LSTINITP # NIL THEN LSTINITP!.NEXTINIT := LINITP;
1363                                    LSTINITP := LINITP
1364                                  END;
1365                              END
1366                                ELSE LSP := NIL;
1367                      FSP := LSP
```

```
1368                    END;
1369                  IF ~(SY IN FSYS) THEN ERRSKIP(10,FSYS);
1370                IF FSP = NIL THEN FSIZE := 1 ELSE FSIZE := FSP!.SIZE
1371              END /*TYP*/ ;
1372
1373              PROCEDURE CONSTANTDECLARATION;
1374                VAR LIP: IDP; LSP: STP; LVALU: VALU;
1375              BEGIN
1376                IF SY # IDENT THEN ERRSKIP(102,FSYS OR ¢IDENT);
1377                WHILE SY = IDENT DO
1378                  BEGIN NEW(LIP,KONST);
1379                    WITH LIP! DO
1380                      BEGIN NAME := ID; IDTYPE := NIL; NEXT := NIL END;
1381                    INSYMBOL;
1382                    IF SY = BECOMES THEN INSYMBOL ELSE ERROR(105);
1383                    CONSTANT(FSYS OR ¢SEMICOLON,LSP,LVALU);
1384                    ENTERID(LIP);
1385                    LIP!.IDTYPE := LSP; LIP!.VALUES := LVALU;
1386                    IF SY = SEMICOLON THEN
1387                      BEGIN INSYMBOL;
1388                        IF ~(SY IN FSYS OR ¢IDENT) THEN ERRSKIP(10,FSYS OR ¢IDENT)
1389                      END
1390                    ELSE ERROR(27)
1391                  END
1392              END /*CONSTANTDECLARATION*/ ;
1393
1394              PROCEDURE TYPEDECLARATION;
1395                VAR LINITP: INITP; LIP,LIP1,LIP2: IDP; LSP: STP; LSIZE: ADDRRANGE;
1396              BEGIN
1397                IF SY # IDENT THEN ERRSKIP(102,FSYS OR ¢IDENT);
1398                WHILE SY = IDENT DO
1399                  BEGIN NEW(LIP,TYPES);
1400                    WITH LIP! DO
1401                      BEGIN NAME := ID; BASED := FALSE; IDTYPE := NIL END;
1402                    INSYMBOL;
1403                    IF SY = BECOMES THEN INSYMBOL ELSE ERROR(105);
1404                    TYP(FSYS OR ¢SEMICOLON,¢RBRACK,LSP,LSIZE,LINITP);
1405                    ENTERID(LIP);
1406                    LIP!.IDTYPE := LSP;
1407                    /*HAS ANY FORWARD REFERENCE BEEN SATISFIED:*/
1408                    LIP1 := FWPTR;
1409                    WHILE LIP1 # NIL DO
1410                      BEGIN
1411                        IF LIP1!.NAME = LIP!.NAME THEN
1412                          BEGIN LIP1!.IDTYPE!.ELTYPE := LIP!.IDTYPE;
1413                            IF LIP1 # FWPTR THEN
1414                              LIP2!.NEXT := LIP1!.NEXT
1415                            ELSE FWPTR := LIP1!.NEXT;
1416                          END;
1417                        LIP2 := LIP1; LIP1 := LIP1!.NEXT
1418                      END;
1419                    IF SY = SEMICOLON THEN
1420                      BEGIN INSYMBOL;
1421                        IF ~(SY IN FSYS OR ¢IDENT) THEN ERRSKIP(10,FSYS OR ¢IDENT)
1422                      END
1423                    ELSE ERROR(27)
1424                  END;
```

```
1425            IF FWPTR # NIL THEN
1426              BEGIN ERROR(106); WRITE(EOL);
1427                REPEAT WRITE(' TYPE-ID ',FWPTR!.NAME,EOL);
1428                  FWPTR := FWPTR!.NEXT
1429                UNTIL FWPTR = NIL;
1430                IF CH # EOL THEN WRITE(' *: CHCNT+8)
1431              END
1432          END /*TYPEDECLARATION*/ ;
1433
1434          PROCEDURE DECLARATION;
1435               /*DEAL WITH A 'DECLARE' STATEMENT*/
1436            VAR LIP,LIP1,LIPP,NXT,NXT1: IDP; LSP,LSP1,LSP2,LSPP,LASTYPE: STP;
1437                LADDR,SIZE,LSIZE: ADDRRANGE; FIRST: BOOLEAN;
1438                LINITP1,LINITP2: INITP;
1439
1440          PROCEDURE PROCEDUREDECLARATION(FIP: IDP);
1441               /*HANDLE ENTRY DECLARATIONS*/
1442            VAR LIP,LIP1: IDP; LSP: STP; FORW: BOOLEAN;
1443
1444            PROCEDURE PARAMTYPELIST(FSYS: SYSET; VAR FPAR: IDP);
1445               /*SET UP CELLS AND
1446                 STORE ATTRIBUTES FOR EACH FORMAL PARAMETER*/
1447              VAR LIP,LIP1,LIP2,LIP3: IDP; LSP,LSP1,LSP2,LSP3: STP;
1448                  OFFSET: ADDRRANGE;
1449            BEGIN LIP1 := NIL; OFFSET := 4;
1450              IF ~(SY IN FSYS OR [LPARENT]) THEN ERRSKIP(28,FSYS OR [LPARENT]);
1451              IF SY = LPARENT THEN
1452                BEGIN INSYMBOL;
1453                  LOOP
1454                  IF ~(SY IN [PROCEDURESY,LPARENT] OR VATTRSYS) THEN
1455                    ERRSKIP(320,FSYS OR [IDENT,RPARENT]);
1456                  IF SY = PROCEDURESY THEN
1457                    BEGIN NEW(LIP,PROC,DECLARED,FORMAL);
1458                      WITH LIP! DO
1459                        BEGIN NAME := '          '; IDTYPE := NIL; NEXT := LIP1;
1460                          PFLEV := LEVEL; PFADDR := OFFSET
1461                        END;
1462                      LIP1 := LIP; OFFSET := OFFSET + 1;
1463                      INSYMBOL
1464                    END
1465                  ELSE
1466                      IF (SY IN VATTRSYS OR [LPARENT]) THEN
1467                        BEGIN NEW(LIP,VARS);
1468                          WITH LIP! DO
1469                            BEGIN NAME := '          ';
1470                              VKIND := FORMAL; NEXT := LIP1; VLEV := LEVEL + 1;
1471                              VADDR := OFFSET
1472                            END;
1473                          LIP1 := LIP; OFFSET := OFFSET + 1;
1474                          VBASE := NULL; VBASED := FALSE;
1475                          TYP(FSYS OR [RPARENT],LSP,LSIZE,LINITP1);
1476                          IF LSP = NIL THEN ERRSKIP(322,FSYS OR LISTDELIMS)
1477                          ELSE
1478                            IF ~(LSP!.FORM IN [SCALAR,ARRAYS,SUBRANGE,POINTER]) THEN
1479                              BEGIN ERROR(321); LIP!.IDTYPE := NIL END;
1480                          IF LSP!.FORM = ARRAYS THEN
1481                            BEGIN TYP(FSYS OR [RPARENT],LSP1,LSIZE,LINITP1);
```

```
1482                          IF ~(LSP!.FORM IN ¢SCALAR,SUBRANGE,POINTER!) THEN
1483                            BEGIN ERROR(321); LIP!.IDTYPE := NIL END;
1484                          LSP2 := LSP;
1485                          REPEAT LSP3 := LSP2;
1486                            LSP2 := LSP3!.AELTYPE;
1487                          UNTIL LSP2 = NIL;
1488                          LSP3!.AELTYPE := LSP1;
1489                        END;
1490                      IF ~(VBASE IN ¢BINARY,DECIMAL!) THEN
1491                        BEGIN
1492                          IF LSP = FIXPTR THEN VBASE := BINARY
1493                          ELSE
1494                          IF LSP = FLTPTR THEN VBASE := DECIMAL
1495                        END;
1496                    IF (LSP = FIXPTR) OR (LSP = FLTPTR) THEN LIP!.BD := VBASE;
1497                      LIP!.IDTYPE := LSP;
1498                      IF ~(SY IN LISTDELIMS OR FSYS) THEN
1499                        ERRSKIP(322,FSYS OR ¢COMMA,SEMICOLON,RPARENT!);
1500                    END;
1501                EXIT IF SY # COMMA;
1502                  INSYMBOL
1503                END;
1504                IF SY = RPARENT THEN
1505                  BEGIN INSYMBOL;
1506                    IF ~(SY IN FSYS) THEN ERRSKIP(10,FSYS)
1507                  END
1508                ELSE ERROR(29);
1509                LIP3 := NIL;
1510             /*REVERSE POINTERS*/
1511                WHILE LIP1 # NIL DO
1512                  WITH LIP1! DO
1513                    BEGIN LIP2 := NEXT; NEXT := LIP3;
1514                      LIP3 := LIP1; LIP1 := LIP2
1515                    END;
1516                FPAR := LIP3
1517              END /*IF SY = LPARENT*/
1518            ELSE FPAR := NIL
1519        END /*PARAMTYPELIST*/ ;
1520
1521        BEGIN /*PROCEDUREDECLARATION*/
1522          NXT := FIP; NXT1 := NIL;
1523          WHILE NXT # NIL DO
1524            WITH NXT! DO
1525              BEGIN  ID := NAME;  LIP := NIL;
1526                SEARCHSECTION(IDTREE¢TOP!,LIP);   /*ALREADY DECLARED*/
1527                IF LIP # NIL THEN
1528                  IF LIP!.KLASS IN ¢PROC,FUNC! THEN
1529                    FORW := LIP!.FORWDECL&(LIP!.PFKIND = ACTUAL)
1530                ELSE FORW := FALSE;
1531                IF ~FORW THEN
1532                  BEGIN
1533                    NEW(LIP,PROC,DECLARED,ACTUAL);
1534                    IF IDTYPE # NIL THEN ERROR(107);
1535                    LIP!.NEXT := NXT1;
1536                    LIP1 := NXT; NXT := NEXT; NEXT := LIP1;
1537                    LIP!.NAME := NAME;
1538                    LIP!.IDTYPE := NIL;
```

```
1539                    LIP!.FORWDECL := TRUE;
1540                    LIP!.EXTERN := FALSE;
1541                    LIP!.PFADDR := 0;
1542                    ENTERID(LIP);
1543                    NXT1 := LIP;
1544                  END
1545                ELSE ERROR(300)
1546              END;
1547            PARAMTYPELIST([SEMICOLON,COMMA,RETURNSSY],LIP1);
1548            IF SY = RETURNSSY THEN
1549              BEGIN INSYMBOL;
1550                IF SY # LPARENT THEN ERRSKIP(25,FSYS OR [SEMICOLON])
1551                ELSE
1552                BEGIN INSYMBOL;
1553                  TYP(FSYS OR [RPARENT],LSP,LSIZE,LINITP1);
1554                  IF LSP # NIL THEN
1555                    IF ~(LSP!.FORM IN [SCALAR,SUBRANGE,POINTER]) THEN
1556                      BEGIN ERROR(301); LIP!.IDTYPE := NIL END;
1557                END
1558              END;
1559            WHILE NXT1 # NIL DO
1560              BEGIN
1561                NXT := LIP!.NEXT;
1562                LIP!.NEXT := LIP1;
1563                IF SY # SEMICOLON THEN
1564                  BEGIN
1565                    IF LSP = NIL THEN
1566                      BEGIN
1567                        IF LIP!.BD = FIX THEN LSP := FIXPTR
1568                        ELSE
1569                        IF LIP!.BD = FLT THEN LSP := FLTPTR
1570                      END;
1571                    IF ~(VBASE IN [BINARY,DECIMAL]) THEN
1572                      BEGIN
1573                        IF LSP = FIXPTR THEN VBASE := BINARY
1574                        ELSE
1575                        IF LSP = FLTPTR THEN VBASE := DECIMAL
1576                      END;
1577                    IF (LSP = FIXPTR) OR (LSP = FLTPTR) THEN LIP!.BD := VBASE;
1578                    LIP!.IDTYPE := LSP;
1579                    LIP!.KLASS := FUNC;
1580                  END;
1581                NXT1 := NXT;
1582              END;
1583              IF SY = RPARENT THEN INSYMBOL;
1584          END /*PROCEDUREDECLARATION*/ ;
1585
1586          PROCEDURE IDLIST(VAR FIP: IDP);
1587               /*SET UP IDENTIFIER CELLS
1588                 AND ENTER THEM INTO THE SYMBOL TABLE*/
1589            VAR NXT,NXT1,LIP: IDP; LSP: STP; EXHAUSTED: BOOLEAN;
1590          BEGIN NXT := NIL;
1591            IF SY = LPARENT THEN
1592              BEGIN INSYMBOL; IDLIST(NXT);
1593                IF SY = RPARENT THEN INSYMBOL ELSE ERROR(24)
1594              END
1595            ELSE
```

```
1596              LOOP
1597                IF SY = IDENT THEN
1598                  BEGIN NEW(LIP,VARS);
1599                    WITH LIP! DO
1600                     BEGIN NAME := ID; NEXT := NXT; NXT1 := NXT;
1601                        IDTYPE := NIL; VKIND := ACTUAL; VLEV := LEVEL
1602                        IF (ID¢1! < 'I') OR (ID¢1! > 'N') THEN BD := FLT
1603                        ELSE BD := FIX;
1604                      END;
1605                    ENTERID(LIP);
1606                    NXT := LIP;
1607                    INSYMBOL;
1608                  END;
1609                LSP := NIL;
1610                IF SY = LPARENT  THEN
1611                  TYP(FSYS OR VATTRSYS OR LISTDELIMS,LSP,SIZE,LINITP1);
1612                IF LSP # NIL THEN
1613                  BEGIN NXT1 := NXT; EXHAUSTED := FALSE;
1614                    REPEAT LIP := NXT1;
1615                      LIP!.IDTYPE := LSP;
1616                      NXT1 := LIP!.NEXT;
1617                      IF NXT1 = NIL THEN EXHAUSTED := TRUE
1618                      ELSE EXHAUSTED := NXT1!.IDTYPE # NIL;
1619                    UNTIL EXHAUSTED;
1620                  END;
1621                IF ~(SY IN FSYS OR ¢COMMA,SEMICOLON,RPARENT! OR ATTRSYS) THEN
1622                 ERRSKIP(10,FSYS OR ¢COMMA,COLON,SEMICOLON! OR ATTRSYS)
1623              EXIT IF (SY # COMMA) OR (LASTSY = RECORDSY);
1624                INSYMBOL
1625              END;
1626            FIP := NXT
1627          END /*IDLIST*/ ;
1628
1629          BEGIN  /*DECLARATION*/
1630            DP := TRUE;
1631            REPEAT VBASED := FALSE;
1632              IF SY = FIXCONST THEN
1633                BEGIN IF VAL.IVAL > 1 THEN ERROR(201); LASTSY := RECORDSY;
1634                  INSYMBOL; IDLIST(LIP); RECIDP := LIP;
1635                  IF SY = BASEDSY THEN
1636                    BEGIN VBASED := TRUE; INSYMBOL; IDLIST(NXT);
1637                      NEW(LSPP,POINTER);
1638                      WITH NXT! DO
1639                        BEGIN IDTYPE := LSPP;
1640                          VADDR := LC; LC := LC + 1;
1641                          IDTYPE!.SIZE := 1; LIP!.KLASS := TYPES;
1642                          LIP!.BASED := TRUE; LIP!.POINTERID := NXT
1643                        END;
1644                    END;
1645                  IF SY = COMMA THEN
1646                    BEGIN INSYMBOL;
1647                      IF SY = FIXCONST THEN
1648                        BEGIN SY := RECORDSY;
1649                          TYP(FSYS OR SEMICOMMA,LSP,SIZE,LINITP1);
1650                          LIP!.IDTYPE := LSP;
1651                          IF VBASED THEN NXT!.IDTYPE!.ELTYPE := LSP
1652                          ELSE
```

```
1653                        BEGIN LIP!.VADDR := LC; LC := LC + LIP!.IDTYPE!.SIZE END
1654                    END
1655                  ELSE ERRSKIP(200,¢SEMICOLON¢);
1656                END
1657              ELSE ERRSKIP(10,FSYS OR SEMICOMMA);
1658              LASTSY := OTHERSY;  RECIDP := NIL;
1659            END /*SY = FIXCONST*/
1660          ELSE
1661            BEGIN IDLIST(NXT);
1662              LSP := NIL; LASTYPE := NIL;
1663              IF ~(SY IN ATTRSYS OR SEMICOMMA) THEN
1664                ERRSKIP(103,FSYS OR SEMICOMMA);
1665              IF (SY IN ATTRSYS OR SEMICOMMA) THEN
1666              BEGIN
1667              IF SY = ENTRYSY THEN
1668                BEGIN INSYMBOL; PROCEDUREDECLARATION(NXT)  END
1669              ELSE
1670              BEGIN
1671               VBASE := NULL; LINITP := NIL; FIRST := TRUE;
1672             TYP(FSYS OR ¢SEMICOLON,COMMA,LPARENT OR ATTRSYS,LSP,LSIZE,LINITP1);
1673               IF VBASED THEN
1674                 BEGIN IDLIST(LIPP);
1675                   NEW(LSPP,POINTER);
1676                   WITH LIPP! DO
1677                     BEGIN IDTYPE := LSPP;
1678                       VADDR := LC;  LC := LC + 1;
1679                       IDTYPE!.SIZE := 1; IDTYPE!.ELTYPE := LSP;
1680                     END;
1681                 END /*IF VBASED*/ ;
1682               WHILE NXT # NIL DO
1683                 WITH  NXT! DO
1684                   BEGIN
1685                     IF LSP = NIL THEN
1686                       BEGIN
1687                         IF BD = FIX THEN LSP := FIXPTR
1688                         ELSE
1689                         IF BD = FLT THEN LSP := FLTPTR
1690                       END;
1691                     IF ~(VBASE IN ¢BINARY,DECIMAL¢) THEN
1692                       BEGIN
1693                         IF LSP = FIXPTR THEN VBASE := BINARY
1694                         ELSE
1695                         IF LSP = FLTPTR THEN VBASE := DECIMAL
1696                       END;
1697                     IF (LSP = FIXPTR) OR (LSP = FLTPTR) THEN BD := VBASE;
1698                     IF IDTYPE = NIL THEN IDTYPE := LSP
1699                     ELSE
1700                     IF IDTYPE # LASTYPE THEN
1701                       BEGIN LSP2 := IDTYPE; LASTYPE := IDTYPE;
1702                           REPEAT LSP1 := LSP2;
1703                           LSP1!.SIZE := LSP1!.SIZE*LSIZE;
1704                           LSP2 := LSP1!.AELTYPE;
1705                           UNTIL LSP2 = NIL;
1706                           LSP1!.AELTYPE := LSP;
1707                       END;
1708                     IF VBASED THEN
1709                       BEGIN KLASS := TYPES; BASED := VBASED;
```

```
1710                          POINTERID := LIPP
1711                        END
1712                      ELSE
1713                        BEGIN
1714                          VADDR := LC;
1715                          IF LINITP1!.NEXTINIT # NIL THEN
1716                            BEGIN LINITP := LINITP1!.NEXTINIT;
1717                              LADDR := VADDR;
1718                              WHILE LINITP # NIL DO
1719                                BEGIN
1720                                  IF FIRST THEN LINITP!.INITADDR := LADDR
1721                                  ELSE
1722                                    BEGIN NEW(LINITP2);
1723                                      WITH LINITP2! DO
1724                                        BEGIN NEXTINIT := NIL;
1725                                          REPEATF := LINITP!.REPEATF;
1726                                          INITVAL := LINITP!.INITVAL;
1727                                          INITADDR := LADDR; LINITP := LINITP2
1728                                        END;
1729                                      LSTINITP!.NEXTINIT := LINITP2;
1730                                      LSTINITP := LINITP2
1731                                    END;
1732                                  LADDR := LADDR + LINITP!.REPEATF;
1733                                  LINITP!.TYPOINTER := LSP;
1734                                  LINITP := LINITP!.NEXTINIT
1735                                END /* WHILE*/ ;
1736                              FIRST := FALSE;
1737                            END /* LINITP1 # NIL*/ ;
1738                          LC := LC + IDTYPE!.SIZE;
1739                        END;
1740                      NXT := NEXT;
1741                    END; /*WHILE*/
1742                END /*SY # ENTRYSY*/
1743              END; /*SY IN ATTRSYS OR SEMICOMMA*/
1744            END;
1745            IF ~(SY IN SEMICOMMA) THEN ERRSKIP(14,FSYS OR SEMICOMMA);
1746            IF (SY IN SEMICOMMA) THEN
1747              IF SY = COMMA THEN
1748              BEGIN INSYMBOL;
1749                IF ~(SY IN FSYS OR ¢IDENT¢) THEN ERRSKIP(10,FSYS OR ¢IDENT¢)
1750              END;
1751          UNTIL (SY = SEMICOLON);
1752          INSYMBOL; DP := FALSE;
1753        END /*DECLARATION*/ ;
1754
1755        PROCEDURE PROCEDUREDEFINE(FSYS: SYSET);
1756            /*HANDLE THE DEFINITION OF EACH BLOCK, INCLUDING
1757              THE OUTERMOST ONE (IE. THE MAIN PROGRAM)*/
1758          VAR OLDLEV: 0..MAXLEVEL;  LIP,LIP1: IDP; LSP: STP;
1759              FORW: BOOLEAN; BLOCKTYPE: BLKTYPE;  OLDTOP: TREERANGE;
1760              LSIZE,LLC: ADDRRANGE;
1761
1762          PROCEDURE PARAMETERLIST(FSYS: SYSET);
1763             /*INSERT FORMAL PARAMETER NAMES INTO THE IDENTIFIER
1764               CELLS ALREADY SET UP BY PROCEDURE PARAMTYPELIST*/
1765            VAR LIP1: IDP;
1766          BEGIN
```

```
1767            IF ~(SY IN FSYS OR ¢LPARENT!) THEN ERRSKIP(323,FSYS OR ¢LPARENT!);
1768            IF SY = LPARENT THEN
1769              BEGIN INSYMBOL; LIP1 := LIP!.NEXT;
1770                LOOP
1771                  IF (SY # IDENT) OR (LIP1 = NIL) THEN
1772                    ERRSKIP(323,FSYS OR ¢IDENT,RPARENT!);
1773                  IF (SY = IDENT)&(LIP1 # NIL) THEN
1774                    BEGIN LIP1!.NAME := ID; ENTERID(LIP1) END;
1775                  INSYMBOL;
1776                  IF ~(SY IN ¢COMMA! OR FSYS) THEN
1777                    ERRSKIP(323,FSYS OR ¢COMMA,SEMICOLON,RPARENT!);
1778                EXIT IF SY # COMMA;
1779                  INSYMBOL;
1780                  LIP1 := LIP1!.NEXT;
1781                END;
1782                WITH LIP1! DO LC := VALDR + IDTYPE!.SIZE;
1783                IF LIP1!.NEXT # NIL THEN ERROR(324);
1784                IF SY = RPARENT THEN
1785                  BEGIN INSYMBOL;
1786                    IF ~(SY IN (¢FSY! OR FSYS)) THEN ERRSKIP(10,FSYS OR ¢FSY!)
1787                  END
1788                ELSE ERROR(24);
1789            END;
1790        END /*PARAMETERLIST*/ ;
1791
1792        BEGIN /*PROCEDUREDEFINE*/
1793          LLC:= LC; IF LEVEL > 0 THEN LC := 4;
1794          IF (SY IN ¢PROCEDURESY,ENTRYSY!) THEN
1795            BEGIN
1796              SEARCHSECTION(IDTREE¢TOP!,LIP);   /*DECLARED ?*/
1797              IF LIP # NIL THEN
1798                FORW := LIP!.FORWDECL&(LIP!.KLASS # NILL)&(LIP!.PFKIND = ACTUAL)
1799              ELSE FORW := FALSE;
1800              IF ~FORW THEN ERROR(302);
1801              IF SY = PROCEDURESY THEN
1802                BEGIN
1803                  IF LIP # MAINP THEN
1804                    IF JMPIX = JMPMAX THEN ERROR(703)
1805                      ELSE BEGIN JMPIX := JMPIX + 1; JMPTAB¢JMPIX! := 0 END;
1806                  LIP!.PFADDR := JMPIX;
1807                   OLDLEV := LEVEL; OLDTOP := TOP;
1808                   IF LEVEL < MAXLEVEL THEN LEVEL := LEVEL + 1 ELSE ERROR(704);
1809                   LIP!.PFLEV := LEVEL;
1810                   IF TOP < TREELIMIT THEN
1811                     BEGIN TOP := TOP + 1; IDTREE¢TOP! := NIL END
1812                   ELSE ERROR(704);
1813                END;
1814                INSYMBOL; BLOCKTYPE := PROCBLK;
1815               IF SY = OPTIONSY  THEN
1816                 BEGIN INSYMBOL;
1817                   IF SY = LPARENT THEN INSYMBOL ELSE ERROR(25);
1818                   IF ID = 'MAIN      ' THEN INSYMBOL ELSE ERROR(402);
1819                   IF SY = RPARENT THEN INSYMBOL ELSE ERROR(24);
1820                 END;
1821               PARAMETERLIST(¢RPARENT,RETURNSSY,SEMICOLON!);
1822               IF SY = RETURNSSY THEN
1823                 BEGIN INSYMBOL; BLOCKTYPE := FUNCBLK;
```

```
1824                IF SY # LPARENT THEN ERRSKIP(25,FSYS OR [LPARENT,SEMICOLON])
1825                ELSE
1826                BEGIN INSYMBOL;
1827                  TYP(FSYS OR [RPARENT],LSP,LSIZE,LINITP);
1828                  IF LSP # NIL THEN
1829                    IF ~(LSP!.FORM IN [SCALAR,SUBRANGE,POINTER]) THEN
1830                      BEGIN ERROR(301); LIP!.IDTYPE := NIL END;
1831                  IF ~COMPTYPES(LSP,LIP!.IDTYPE) THEN ERROR(303);
1832                  IF LIP!.KLASS # FUNC THEN ERROR(304);
1833                  IF SY = RPARENT THEN INSYMBOL ELSE ERROR(24);
1834                END
1835              END;
1836            BEGBLOCK(BLOCKTYPE);
1837            IF SY = SEMICOLON THEN INSYMBOL ELSE ERROR(27);
1838            IF LIP # MAINP THEN
1839              BEGIN
1840                BLOCK(FSYS,BLOCKTYPE,SEMICOLON,LIP);
1841                IF SY = SEMICOLON THEN
1842                  BEGIN INSYMBOL;
1843                    IF ~(SY IN STATMENTSYS) THEN
1844                      ERRSKIP(10,FSYS)
1845                  END
1846                ELSE ERROR(27);
1847              IF ATTRLIST THEN ATTRIBUTELIST(FALSE);
1848              LEVEL := OLDLEV; TOP := OLDTOP;
1849            END;
1850          END;
1851          LC := LLC;
1852      END /*PROCEDUREDEFINE*/ ;
1853
1854      PROCEDURE BODY(FSYS: SYSET);
1855          /*DRIVE THE PROCEDURES WHICH HANDLE THE BODY PART
1856            OF A PROCEDURE, AND ALSO THE WRITING TO DISC
1857            OF GENERATED CODE*/
1858        CONST CIXMAX =  1000; CSTOCCMAX = 60; LABMAX = 20;
1859        TYPE OPRANGE = 0..63;  CODERANGE = 0..CIXMAX;
1860        VAR CODE: ARRAY [CODERANGE] OF PACKED
1861                  RECORD OP: OPRANGE; P1: 0..63; P2: INTEGER END;
1862            CIX,IX: INTEGER;
1863            FRETNIX: 1..10;
1864            FUNCRETN: ARRAY[1..10] OF ADDRRANGE;
1865            CSTPTR: ARRAY [1..CSTOCCMAX] OF CSP;
1866            CSTPTRIX: 0..CSTOCCMAX;
1867            /*ALLOWS REFERENCING OF NONINTEGER CONSTANTS BY AN INDEX
1868             (INSTEAD OF A POINTER), WHICH CAN BE STORED IN THE P2-FIELD
1869             OF THE INSTRUCTION RECORD UNTIL WRITEOUT.
1870             --> PROCEDURE LOAD WRITEOUT*/
1871            LABPTR: ARRAY[1..LABMAX] OF LBP;
1872            LABPTRIX: 1..LABMAX;
1873            I: INTEGER; EPMAIN: ADDRRANGE;
1874            LCMAX,LLC1: ADDRRANGE; LIP: IDP;
1875            LLP: LBP;
1876
1877        PROCEDURE GENC(FOP: OPRANGE);
1878           /*GENERATE 1 INSTRUCTION, NO PARAMETERS*/
1879        BEGIN CIX := CIX + 1;
1880          IF CIX > CIXMAX THEN
```

```
1881                BEGIN ERROR(702); CIX := 0 END;
1882              WITH CODE¢CIX! DO
1883                BEGIN OP := FOP; P1 := 0; P2 := 0 END;
1884              IC := IC + 1
1885            END /*GEN0*/ ;
1886
1887            PROCEDURE GEN1(FOP: OPRANGE; FP2: INTEGER);
1888               /*GENERATE 1 INSTRUCTION, 1 PARAMETER*/
1889            BEGIN CIX := CIX + 1;
1890              IF CIX > CIXMAX THEN
1891                BEGIN ERROR(702); CIX := 0 END;
1892              WITH CODE¢CIX! DO
1893                BEGIN OP := FOP; P2 := FP2; P1 := 0 END;
1894              IC := IC + 1
1895            END /*GEN1*/ ;
1896
1897            PROCEDURE GEN2(FOP: OPRANGE; FP1,FP2: INTEGER);
1898               /*GENERATE 1 INSTRUCTION, 1 PARAMETER, 1 ADDRESS*/
1899            BEGIN CIX := CIX + 1;
1900             IF CIX > CIXMAX THEN
1901                BEGIN ERROR(702); CIX := 0 END;
1902             WITH CODE¢CIX! DO
1903               BEGIN OP := FOP; P1 := FP1; P2 := FP2 END;
1904              IC := IC + 1
1905            END /*GEN2*/ ;
1906
1907            PROCEDURE LOAD;
1908               /*GENERATE A LOAD INSTRUCTION OF A CONSTANT OR VARIABLE*/
1909            BEGIN
1910              WITH GOESC DO
1911                IF TYPTR # NIL THEN
1912                  BEGIN
1913                    CASE KIND OF
1914                      CST:   IF (TYPTR!.FORM = SCALAR)&(TYPTR # FLTPTR) THEN
1915                               IF TYPTR = BOOLPTR THEN GEN2(51/*LDC*/,3,CVAL.IVAL)
1916                               ELSE GEN2(51/*LDC*/,1,CVAL.IVAL)
1917                             ELSE
1918                               IF TYPTR = NILPTR THEN GEN2(51/*LDC*/,4,0)
1919                               ELSE
1920                                 IF CSTPTRIX >= CSTOCCMAX THEN ERROR(701)
1921                                 ELSE
1922                                   BEGIN CSTPTRIX := CSTPTRIX + 1;
1923                                     CSTPTR¢CSTPTRIX! := CVAL.VALP;
1924                                     IF TYPTR = FLTPTR THEN
1925                                       GEN2(51/*LDC*/,2,CSTPTRIX)
1926                                     ELSE GEN2(51/*LDC*/,5,CSTPTRIX)
1927                                   END;
1928                      VARBL: CASE ACCESS OF
1929                               DRCT:   IF VLEVEL <= 1 THEN GEN1(39/*LDO*/,DPLMT)
1930                                       ELSE GEN2(54/*LOD*/,LEVEL-VLEVEL,DPLMT);
1931                               INDRCT: GEN1(35/*IND*/,IDPLMT);
1932                               INXD:   BEGIN GEN1(36/*IXA*/,TYPTR!.SIZE);
1933                                         GEN1(35/*IND*/,0)
1934                                       END
1935                             END;
1936                      EXPR:
1937                    END;
```

```
1938                KIND := EXPR
1939              END
1940          END /*LOAD*/ ;
1941
1942          PROCEDURE STORE(VAR FDESC: DESC);
1943            /*GENERATE A STORE INSTRUCTION*/
1944          BEGIN
1945            WITH FDESC DO
1946              IF TYPTR # NIL THEN
1947                CASE ACCESS OF
1948                  DRCT:   IF VLEVEL <= 1 THEN GEN1(43/*SRO*/,DPLMT)
1949                          ELSE GEN2(56/*STR*/,LEVEL-VLEVEL,DPLMT);
1950                  INDRCT: IF IDPLMT # 0 THEN ERROR(802)
1951                          ELSE GEN0(26/*STO*/);
1952                  INXD:   ERROR(800)
1953                END
1954          END /*STORE*/ ;
1955
1956          PROCEDURE LOADADDRESS;
1957            /*GENERATE A LOAD-ADDRESS INSTRUCTION FOR USE WITH
1958              STRING OR FLOAT CONSTANTS, AND BASE ADDRESSES
1959              OF ARRAYS AND RECORDS, ALSO FOR LOADING
1960              OF PARAMETERS*/
1961          BEGIN
1962            WITH GDESC DO
1963              IF TYPTR # NIL THEN
1964                BEGIN
1965                  CASE KIND OF
1966                    CST:    IF STRING(TYPTR) THEN
1967                              IF CSTPTRIX >= CSTOCCMAX THEN ERROR(701)
1968                              ELSE
1969                                BEGIN CSTPTRIX := CSTPTRIX + 1;
1970                                  CSTPTR^[CSTPTRIX] := CVAL.VALP;
1971                                  GEN2(38/*LCA*/,CSTPTR^[CSTPTRIX]^.SLGTH,CSTPTRIX)
1972                                END
1973                            ELSE ERROR(801);
1974                    VARBL: CASE ACCESS OF
1975                             DRCT:   IF VLEVEL <= 1 THEN GEN1(37/*LAO*/,DPLMT)
1976                                     ELSE GEN2(50/*LDA*/,LEVEL-VLEVEL,DPLMT);
1977                             INDRCT: IF IDPLMT # 0 THEN GEN1(34/*INC*/,IDPLMT);
1978                             INXD:   GEN1(36/*IXA*/,TYPTR^.SIZE)
1979                           END;
1980                    EXPR:  ERROR(803)
1981                  END;
1982                  KIND := VARBL; ACCESS := INDRCT; IDPLMT := 0
1983                END
1984          END /*LOADADDRESS*/ ;
1985
1986          PROCEDURE INSERT(FCIX: CODERANGE; FIC: ADDRRANGE);
1987            /*INSERT AN ADDRESS FIC INTO
1988              THE INSTRUCTION AT ADDRESS FCIX*/
1989          BEGIN CODE^[FCIX].P2 := FIC
1990          END /*INSERT*/ ;
1991
1992          PROCEDURE GENFJP(FADDR: ADDRRANGE);
1993            /*GENERATE A FALSE JUMP INSTRUCTION*/
1994          BEGIN LOAD
```

```
1995              IF GDESC.TYPTR # NIL THEN
1996                IF GDESC.TYPTR # BOOLPTR THEN ERROR(500);
1997              GEN1(33/*FJP*/,FADDR)
1998            END /*GENFJP*/ ;
1999
2000            PROCEDURE WRITEOUT;
2001              /*FOR EACH PROCEDURE, TRANSLATE THE CODE INTO
2002                MNEMONIC FORM, AND WRITE OUT THE CODE, IN
2003                NUMERIC AND MNEMONIC FORM, TO DISC*/
2004              VAR LOP: OPRANGE; LP1,LP2,I,K: INTEGER; LIC: ADDRRANGE; LCH: CHAR;
2005                  LLP,LLP1: LBP; FORWLAB: BOOLEAN; LABLNAM: ALFA;
2006            BEGIN LIC := IC - CIX - 1;
2007              FOR I := 0 TO CIX DO
2008                BEGIN
2009                  IF LIC MOD 10 = 0 THEN WRITE(LIC:7)
2010                    ELSE WRITE(' ':7);
2011                  WITH CODE^[I] DO
2012          /*UJP*/     BEGIN FORWLAB := FALSE;
2013                      IF OP = 57 THEN FORWLAB := ((P1 = 0) OR (P1 = 99));
2014                      IF OP = 51 THEN FORWLAB := P1 = 7;
2015                      IF FORWLAB THEN
2016                        BEGIN LLP := FSTLABP;
2017                          LLP1 := LABPTR^[P2];
2018                          LABLNAM := LLP1^.LABNAME;
2019                          WHILE LLP # FLABP DO
2020                            WITH LLP^ DO
2021                              IF LABNAME = LABLNAM THEN
2022                                BEGIN
2023                                  IF OP = 51 THEN P1 := 6
2024                                  ELSE
2025                                  IF LABLEV > P1 THEN ERROR(404) ELSE P1 := 0;
2026                                  P2:=LABADDR; LLP:=FLABP
2027                                END
2028                              ELSE LLP := NEXTLAB
2029                        END;
2030                      IF PRCODE THEN WRITE(' ':7,OP:3,P1:3,P2:12);
2031                    END;
2032                  IF PRSYMB THEN BEGIN
2033                  LOP := CODE^[I].OP;
2034                  WRITE(' ',MN^[LOP]:3);
2035                  IF LOP >= 30 THEN
2036                    BEGIN LP2 := CODE^[I].P2;
2037                      IF LOP < 45 THEN BEGIN
2038                        IF LOP = 30 THEN WRITE(' ',SNA^[LP2]:3,'   ',CODE^[I].P1:3)
2039                        ELSE
2040                        IF LOP = 32 /*ENT*/ THEN WRITE(' ',LP2:5,' ':20,FPROCP^.NAME)
2041                        ELSE
2042                        IF LOP = 38 THEN WRITE(LP1:6)
2043                        ELSE
2044            /*RET*/       IF LOP = 42 THEN WRITE(CHR(LP2))
2045                        ELSE WRITE(LP2: 6) END
2046                      ELSE
2047                        BEGIN LP1 := CODE^[I].P1;
2048                          CASE LOP OF
2049            /*CHK,CUP*/       45,46,
2050            /*LDA,LOD,STR*/   50,54,56:
2051                              WRITE(' ',LP1:5,' ',LP2:5);
```

```
2052                        /*NO CHK INSTRUCTION IS
2053                          GENERATED BY THIS COMPILER*/
2054          /*EQU,GEQ,GRT*/   47,48,49,
2055          /*LEQ,LES,NEQ*/   52,53,55:
2056                          BEGIN WRITE(CHR(LP1));
2057                            IF CHR(LP1) = 'M' THEN WRITE(' ',LP2:4)
2058                          END;
2059          /*LDC*/           51:
2060                          CASE LP1 OF
2061                          /*I*/ 1: WRITE('I ':2,LP2:12);
2062                          /*R*/ 2: WRITE('R ':2,CSTPTR@LP2!!.RVAL);
2063                          /*B*/ 3: WRITE('B',LP2:5);
2064                          /*N*/ 4: WRITE('N');
2065                          /*S*/ 5:  WRITE('S ':2,CSTPTR@LP2!!.PVAL);
2066                          /*L*/ 6:  WRITE('L ':2,LP2:12)
2067                          END /*CASE LP1*/ ;
2068          /*UJP*/           57:
2069                          IF LP1 = 0 THEN WRITE(' ',LP2:5)
2070                          ELSE IF LP1 = 1 THEN WRITE('     *')
2071                                          ELSE WRITE(' ':6,LP1:6)
2072                      END /*CASE*/
2073                    END /*ELSE*/
2074                END /*IF LOP >= 30*/
2075              END /*IF PRSYMB*/ ;
2076              WRITE(EOL);
2077              IF PRCODE THEN BEGIN
2078              WITH CODE@[I] DO
2079                IF OP = 38 THEN
2080                  BEGIN
2081                    WITH CSTPTR@P2!! DO
2082                      FOR K := 1 TO SLGTH DO WRITE(SVAL@[K]);
2083                    WRITE(EOL)
2084                  END
2085                ELSE
2086                  IF OP = 51 THEN
2087                    IF P1 = 2 THEN BEGIN WRITE(' ',CSTPTR@P2!!.RVAL);
2088                                         WRITE(EOL) END
2089                    ELSE IF P1 = 5 THEN BEGIN WRITE(' ',CSTPTR@P2!!.PVAL);
2090                                              WRITE(EOL) END;
2091              END;
2092               LIC := LIC + 1;
2093          END /*FOR*/ ;
2094        END /*WRITEOUT*/ ;
2095
2096        PROCEDURE STATEMENT(FSYS: SYSET);
2097          /*ANALYZE LABELS AND DRIVE THE STATEMENT PROCESSORS*/
2098          LABEL 1;
2099          VAR LDP: IDP; LLP: LBP;
2100
2101          PROCEDURE EXPRESSION(FSYS: SYSET); FORWARD;
2102
2103          PROCEDURE SELECTOR(FSYS: SYSET; FIP: IDP);
2104          /*ANALYZE EACH VARIABLE NAME IN A STATEMENT,
2105            GENERATING INSTRUCTIONS IF NECESSARY. DEAL WITH
2106            COMPOUND NAMES EG. SUBSCRIPTED VARIABLES, FIELDS,
2107            BASED VARIABLES, AND FUNCTION NAMES. */
2108           VAR LOPLMT: ADDRRANGE; LDESC: DESC; LIP: IDP; LMIN,LMAX: INTEGER;
```

```
2109            BEGIN LDPLMT := 0; LIP := FIP;
2110              WITH GDESC DO
2111                BEGIN TYPTR := LIP!.IDTYPE; KIND := VARBL;
2112                  IF LIP!.KLASS = FIELD THEN
2113                    BEGIN
2114                      WHILE LIP!.KLASS = FIELD DO
2115                      BEGIN LDPLMT := LDPLMT + LIP!.FLDADDR; LIP := LIP!.NEXT END;
2116                      IF LIP!.IDTYPE!.FORM # RECORDS THEN ERROR(804);
2117                    END;
2118                  CASE LIP!.KLASS OF
2119                    TYPES:
2120                      IF ~LIP!.BASED THEN ERROR(470)
2121                      ELSE
2122                        BEGIN LASTSY := SY; SY := IDENT;
2123                          SELECTOR(¢IDENT¢,LIP!.POINTERID); LOAD;
2124                          SY := LASTSY; LASTSY := OTHERSY;
2125                          WITH GDESC, TYPTR! DO
2126                            BEGIN KIND := VARBL; ACCESS := INDRCT;
2127                              IDPLMT := LDPLMT
2128                            END;
2129                        END;
2130                    VARS:
2131                      WITH LIP! DO
2132                        IF VKIND = ACTUAL THEN
2133                          BEGIN ACCESS := DRCT; VLEVEL := VLEV;
2134                            DPLMT := VADDR + LDPLMT
2135                          END
2136                        ELSE
2137                          BEGIN GEN2(54/*LOD*/,LEVEL-VLEV,VADDR);
2138                            ACCESS := INDRCT; IDPLMT := LDPLMT
2139                          END;
2140                    FIELD:
2141                      BEGIN LDPLMT := 0;
2142                        WHILE LIP!.KLASS = FIELD DO
2143                          BEGIN
2144                            LDPLMT := LDPLMT + LIP!.FLDADDR;
2145                            LIP := LIP!.NEXT
2146                          END;
2147                        WITH LIP! DO
2148                          IF VKIND = ACTUAL THEN
2149                            BEGIN ACCESS := DRCT; VLEVEL := VLEV;
2150                              DPLMT := VADDR + LDPLMT
2151                            END
2152                          ELSE
2153                            BEGIN GEN2(54/*LOD*/,LEVEL-VLEV,VADDR);
2154                              ACCESS := INDRCT; IDPLMT := LDPLMT
2155                            END;
2156                      END;
2157                    FUNC:
2158                      WITH LIP! DO
2159                        IF PFDECKIND = STANDARD THEN ERROR(150)
2160                        ELSE
2161                        IF PFLEV = 0 THEN ERROR(305)  /*EXTERNAL FCT*/
2162                        ELSE
2163                        IF PFKIND = FORMAL THEN ERROR(306)
2164                        ELSE
2165                          BEGIN ACCESS := DRCT; VLEVEL := PFLEV + 1;
```

```
2166                          DPLMT := 0   /*IMPL. RELAT. ADDR. OF FCT. RESULT*/
2167                        END
2168                    END /*CASE*/ ;
2169                  END /*WITH*/;
2170                IF ~(SY IN SELECTSYS OR FSYS) THEN ERRSKIP(133,FSYS OR SELECTSYS);
2171                WHILE SY IN SELECTSYS DO
2172                  BEGIN
2173            /*(*/   IF SY = LPARENT THEN
2174                    BEGIN
2175                      REPEAT LDESC := GDESC;
2176                        WITH LDESC DO
2177                          IF TYPTR # NIL THEN
2178                            IF TYPTR!.FORM # ARRAYS THEN
2179                              BEGIN ERROR(158); TYPTR := NIL END;
2180                        LOADADDRESS;
2181                        INSYMBOL; EXPRESSION(FSYS OR LISTDELIMS);
2182                        LOAD;
2183                        IF GDESC.TYPTR # NIL THEN
2184                          IF GDESC.TYPTR!.FORM # SCALAR THEN ERROR(154);
2185                        IF LDESC.TYPTR # NIL THEN
2186                          WITH LDESC.TYPTR! DO
2187                            BEGIN
2188                              IF COMPTYPES(INXTYPE,GDESC.TYPTR) THEN
2189                                BEGIN
2190                                  IF INXTYPE # NIL THEN
2191                                    BEGIN GETBOUNDS(INXTYPE,LMIN,LMAX);
2192                                      IF LMIN > 0 THEN GEN1(31/*DEC*/,LMIN)
2193                                      ELSE IF LMIN < 0 THEN GEN1(34/*INC*/,-LMIN)
2194                                      /*OR SIMPLY GEN1(31,LMIN)*/
2195                                    END
2196                                END
2197                              ELSE ERROR(157);
2198                              WITH GDESC DO
2199                                BEGIN TYPTR := AELTYPE; KIND := VARBL;
2200                                  ACCESS := INDRCT; IDPLMT := 0
2201                                END;
2202                              IF GDESC.TYPTR # NIL THEN
2203                                GEN1(36/*IXA*/,GDESC.TYPTR!.SIZE)
2204                            END
2205                      UNTIL SY # COMMA;
2206                      IF SY = RPARENT THEN INSYMBOL ELSE ERROR(24)
2207                    END /*IF SY = LPARENT*/
2208                  ELSE
2209            /*.*/     IF SY = PERIOD THEN
2210                      BEGIN
2211                        WITH GDESC DO
2212                          BEGIN TYPTR := LIP!.IDTYPE;
2213                            IF TYPTR # NIL THEN
2214                              IF TYPTR!.FORM # RECORDS THEN
2215                                BEGIN ERROR(202); TYPTR := NIL END;
2216                            INSYMBOL;
2217                            IF SY = IDENT THEN
2218                              BEGIN
2219                                IF TYPTR # NIL THEN
2220                                  BEGIN RECIDP := LIP; SEARCHID(@FIELD!,LIP);
2221                                    IF (LIP = UFLDPTR) OR (LIP!.NEXT # RECIDP) THEN
2222                                      BEGIN ERROR(203); TYPTR := NIL END
```

```
2223                                ELSE
2224                                  WITH LIP! DO
2225                                    BEGIN TYPTR := IDTYPE;
2226                                      CASE ACCESS OF
2227                                        DRCT:   DPLMT := DPLMT + FLDADDR;
2228                                        INDRCT: IDPLMT := IDPLMT + FLDADDR;
2229                                        INXD:   BEGIN LOADADDRESS;
2230                                                  IDPLMT := FLDADDR
2231                                                END
2232                                      END
2233                                    END;
2234                                  RECIDP := NIL;
2235                              END;
2236                            INSYMBOL
2237                          END /*SY = IDENT*/
2238                        ELSE ERROR(21)
2239                      END /*WITH GDESC*/
2240                  END /*IF SY = PERIOD*/
2241              ELSE
2242 /*!*/           BEGIN
2243                  IF GDESC.TYPTR # NIL THEN
2244                    WITH GDESC.TYPTR! DO
2245                      IF FORM = POINTER THEN
2246                        BEGIN TYPTR := ELTYPE; LOAD;
2247                          WITH GDESC DO
2248                            BEGIN KIND := VARBL; ACCESS := INDRCT;
2249                              IDPLMT := 0
2250                            END
2251                        END
2252                      ELSE ERROR(181);
2253                  INSYMBOL;
2254                  IF SY = IDENT THEN
2255                    BEGIN SEARCHID([TYPES,FIELD],LIP); RECIDP := LIP;
2256                      IF LIP!.KLASS = FIELD THEN
2257                        WITH GDESC DO
2258                          WHILE LIP!.KLASS = FIELD DO
2259                            BEGIN IDPLMT := IDPLMT + LIP!.FLDADDR;
2260                              LIP := LIP!.NEXT
2261                            END;
2262                      IF ~LIP!.BASED THEN ERROR(470);
2263                      IF LIP # UTYPPTR THEN
2264                        WITH GDESC DO
2265                        IF TYPTR # NIL THEN
2266                          IF ~COMPTYPES(LIP!.IDTYPE,TYPTR) THEN ERROR(472);
2267                      GDESC.TYPTR := RECIDP!.IDTYPE; RECIDP := NIL
2268                    END /*SY = IDENT*/
2269                  ELSE ERROR(471);
2270                  INSYMBOL
2271                END;
2272              IF ~(SY IN FSYS OR SELECTSYS) THEN ERRSKIP(10,FSYS OR SELECTSYS)
2273            END /*WHILE*/
2274        END /*SELECTOR*/ ;
2275
2276        PROCEDURE LABELCONSTANT(VAR P1,P2: INTEGER);
2277          LABEL 1;
2278          VAR LLP:LBP;
2279        BEGIN LLP := FSTLABP;
```

```
2280                WHILE LLP # FLABP DO
2281                  WITH LLP! DO
2282                    IF LABNAME = ID THEN
2283                      BEGIN
2284                        IF LASTSY = GOTOSY THEN P1 := 0 ELSE P1 := 6;
2285                        P2 := LABADDR; GOTO 1
2286                      END
2287                    ELSE LLP := NEXTLAB;
2288                NEW(LLP);
2289                WITH LLP! DO
2290                  BEGIN
2291                    LABNAME := ID; LABLEV := 0;
2292                    LABADDR := 0; NEXTLAB := FSTLABP
2293                  END;
2294                FSTLABP := LLP; LLP!.DEFINED := FALSE;
2295                LABPTRIX := LABPTRIX + 1;
2296                IF LABPTRIX > LABMAX THEN ERROR(705)
2297                ELSE LABPTR¢LABPTRIX! := LLP;
2298                IF LASTSY = GOTOSY THEN P1 := BLOCKS¢BIX!.BEGLEV
2299                ELSE P1 := 7; P2 := LABPTRIX;
2300          1:    INSYMBOL; LASTSY := OTHERSY
2301              END /*LABELCONSTANT*/ ;
2302
2303              PROCEDURE VARIABLE(FSYS: SYSET);
2304                VAR LIP: IDP;
2305                BEGIN
2306                  IF SY = IDENT THEN
2307                    BEGIN SEARCHID(¢VARS,TYPES,FIELD!,LIP); INSYMBOL END
2308                  ELSE BEGIN ERROR(21); LIP := UVARPTR END;
2309                  SELECTOR(FSYS,LIP)
2310                END /*VARIABLE*/ ;
2311
2312              PROCEDURE FILENAME(VAR FIP: IDP);
2313                VAR LIP: IDP;
2314                BEGIN LIP := UVARPTR;
2315                  IF SY = FILESY THEN
2316                    BEGIN INSYMBOL;
2317                      IF SY = LPARENT THEN INSYMBOL ELSE ERROR(25);
2318                      IF SY = IDENT THEN
2319                        BEGIN SEARCHID(¢VARS!,LIP);
2320                          IF LIP!.IDTYPE!.FORM # FILES THEN
2321                            ERRSKIP(183,FSYS OR ¢SKIPSY,PAGESY,LISTSY!)
2322                          ELSE INSYMBOL
2323                        END
2324                      ELSE ERROR(21);
2325                      IF SY = RPARENT THEN INSYMBOL ELSE ERROR(24)
2326                    END;
2327                  FIP := LIP
2328                END /*FILENAME*/ ;
2329
2330              PROCEDURE CALL(FSYS: SYSET; FIP: IDP);
2331              /*ANALYZE A PROCEDURE CALL FOR STANDARD
2332                AND DECLARED INTERNAL AND EXTERNAL PROCEDURES*/
2333                VAR LKEY: 1..15;
2334
2335            /*STANDARD PROCEDURES*/
2336
```

```
2337              PROCEDURE PACK;
2338                VAR LSP,LSP1: STP;
2339              BEGIN ERROR(403); VARIABLE(FSYS OR LISTDELIMS);
2340                LSP := NIL; LSP1 := NIL;
2341                IF GDESC.TYPTR # NIL THEN
2342                  WITH GDESC.TYPTR! DO
2343                    IF FORM = ARRAYS THEN
2344                      BEGIN LSP := INXTYPE; LSP1 := AELTYPE END
2345                    ELSE ERROR(326);
2346                IF SY = COMMA THEN INSYMBOL ELSE ERROR(29);
2347                EXPRESSION(FSYS OR LISTDELIMS);
2348                IF GDESC.TYPTR # NIL THEN
2349                  IF GDESC.TYPTR!.FORM # SCALAR THEN ERROR(326)
2350                  ELSE
2351                    IF ~COMPTYPES(LSP,GDESC.TYPTR) THEN ERROR(325);
2352                IF SY = COMMA THEN INSYMBOL ELSE ERROR(29);
2353                VARIABLE(FSYS OR [RPARENT]);
2354                IF GDESC.TYPTR # NIL THEN
2355                  WITH GDESC.TYPTR! DO
2356                    IF FORM = ARRAYS THEN
2357                      BEGIN
2358                        IF ~COMPTYPES(AELTYPE,LSP1) OR ~COMPTYPES(INXTYPE,LSP) THEN
2359                          ERROR(326)
2360                      END
2361                    ELSE ERROR(326)
2362              END /*PACK*/ ;
2363
2364              PROCEDURE UNPACK;
2365                VAR LSP,LSP1: STP;
2366              BEGIN ERROR(403); VARIABLE(FSYS OR LISTDELIMS);
2367                LSP := NIL; LSP1 := NIL;
2368                IF GDESC.TYPTR # NIL THEN
2369                  WITH GDESC.TYPTR! DO
2370                    IF FORM = ARRAYS THEN
2371                      BEGIN LSP := INXTYPE; LSP1 := AELTYPE END
2372                    ELSE ERROR(326);
2373                IF SY = COMMA THEN INSYMBOL ELSE ERROR(29);
2374                VARIABLE(FSYS OR LISTDELIMS);
2375                IF GDESC.TYPTR # NIL THEN
2376                  WITH GDESC.TYPTR! DO
2377                    IF FORM = ARRAYS THEN
2378                      BEGIN
2379                        IF ~COMPTYPES(AELTYPE,LSP1) OR ~COMPTYPES(INXTYPE,LSP) THEN
2380                          ERROR(326)
2381                      END
2382                    ELSE ERROR(326);
2383                IF SY = COMMA THEN INSYMBOL ELSE ERROR(29);
2384                EXPRESSION(FSYS OR [RPARENT]);
2385                IF GDESC.TYPTR # NIL THEN
2386                  IF GDESC.TYPTR!.FORM # SCALAR THEN ERROR(326)
2387                  ELSE
2388                    IF ~COMPTYPES(LSP,GDESC.TYPTR) THEN ERROR(326);
2389              END /*UNPACK*/ ;
2390
2391              PROCEDURE MARK;
2392              BEGIN VARIABLE(FSYS OR [RPARENT]);
2393                IF COMPTYPES(FIXPTR,GDESC.TYPTR) THEN
```

```
2394                    BEGIN LOADADDRESS; GEN1(30/*CSP*/,13/*SAV*/) END
2395                    ELSE ERROR(327)
2396                  END /*MARK*/ ;
2397
2398                  PROCEDURE RELEASE;
2399                  BEGIN EXPRESSION(FSYS OR [RPARENT]);
2400                    IF GDESC.TYPTR # NIL THEN
2401                      IF GDESC.TYPTR = FIXPTR THEN
2402                        BEGIN LOAD; GEN1(30/*CSP*/,14/*RST*/) END
2403                      ELSE ERROR(327)
2404                  END /*RELEASE*/ ;
2405
2406            /*BUILTIN FUNCTIONS*/
2407
2408                  PROCEDURE ABS;
2409                  BEGIN
2410                    IF GDESC.TYPTR # NIL THEN
2411                      IF GDESC.TYPTR = FIXPTR THEN GEN0(0/*ABI*/)
2412                      ELSE
2413                        IF GDESC.TYPTR = FLTPTR THEN GEN0(1/*ABR*/)
2414                        ELSE BEGIN ERROR(327); GDESC.TYPTR := FIXPTR END
2415                  END /*ABS*/ ;
2416
2417                  PROCEDURE TRUNC;
2418                  BEGIN
2419                    IF GDESC.TYPTR # NIL THEN
2420                      IF GDESC.TYPTR # FLTPTR THEN ERROR(327);
2421                    GEN0(27/*TRC*/);
2422                    GDESC.TYPTR := FIXPTR
2423                  END /*TRUNC*/ ;
2424
2425                  PROCEDURE ORD;
2426                  BEGIN
2427                    IF GDESC.TYPTR # NIL THEN
2428                      IF GDESC.TYPTR!.FORM > SETS THEN ERROR(327);
2429                    GDESC.TYPTR := FIXPTR
2430                  END /*ORD*/ ;
2431
2432                  PROCEDURE CHR;
2433                  BEGIN
2434                    IF GDESC.TYPTR # NIL THEN
2435                      IF GDESC.TYPTR # FIXPTR THEN ERROR(327);
2436                    GDESC.TYPTR := CHARPTR
2437                  END /*CHR*/ ;
2438
2439                  PROCEDURE PREDSUCC;
2440                  BEGIN ERROR(403);
2441                    IF GDESC.TYPTR # NIL THEN
2442                      IF GDESC.TYPTR!.FORM # SCALAR THEN ERROR(327);
2443                  END /*PREDSUCC*/ ;
2444
2445                  PROCEDURE SGN;
2446                  BEGIN EXPRESSION(FSYS OR [RPARENT]);
2447                    IF GDESC.TYPTR # NIL THEN
2448                      IF GDESC.TYPTR!.FORM # SCALAR THEN ERROR(327);
2449                    LOAD; GEN1(30/*CSP*/,2/*SGN*/);
2450                      GDESC.TYPTR := FIXPTR
```

```
2451                END (*SGN*) ;
2452
2453                PROCEDURE MINMAX;
2454                  VAR I: INTEGER;
2455                BEGIN
2456                  FOR I := 1 TO 2 DO
2457                    BEGIN EXPRESSION(FSYS OR LISTDELIMS);
2458                      IF GDESC.TYPTR # NIL THEN
2459                        IF GDESC.TYPTR!.FORM # SCALAR THEN ERROR(327)
2460                        ELSE LOAD;
2461                      IF SY = COMMA THEN INSYMBOL;
2462                    END;
2463                  GEN1(30/*CSP*/,LKEY+13/*MIN,MAX*/)
2464                END (*MINMAX*) ;
2465
2466                PROCEDURE SUM;
2467                BEGIN VARIABLE(FSYS OR [RPARENT]); GDESC.KIND := VARBL;
2468                  IF GDESC.TYPTR # NIL THEN
2469                    IF GDESC.TYPTR!.FORM # ARRAYS THEN ERROR(327)
2470                    ELSE
2471                      BEGIN GEN2(51/*LDC*/,1,GDESC.TYPTR!.SIZE);
2472                        LOADADDRESS; GEN1(30/*CSP*/,1/*SUM*/)
2473                      END;
2474                END (*SUM*) ;
2475
2476                PROCEDURE CALLNONSTANDARD;
2477              (*DEAL WITH A CALL OF A DECLARED PROCEDURE*)
2478                  VAR NXT,LIP: IDP; LSP: STP; LKIND: IDKIND; LB: BOOLEAN;
2479                  NROFPAR: INTEGER;
2480                BEGIN NROFPAR := 0;
2481                  WITH FIP! DO
2482                    BEGIN NXT := NEXT; LKIND := PFKIND;
2483                      IF PEXTERN THEN
2484                        IF KLASS = PROC THEN GEN2(41/*MST*/,LEVEL-PFLEV,0)
2485                        ELSE GEN2(41/*MST*/,LEVEL-PFLEV,1);
2486                    END;
2487                  IF SY = LPARENT THEN
2488                    BEGIN
2489                      REPEAT LB := FALSE;   /*DECIDE WHETHER PROC/FUNC MUST BE PASSED
2490                        IF LKIND = ACTUAL THEN
2491                          BEGIN
2492                            IF NXT = NIL THEN ERROR(324)
2493                            ELSE LB := NXT!.KLASS IN [PROC,FUNC]
2494                          END ELSE ERROR(325);
2495                        (*FOR FORMAL PROC/FUNC LB IS FALSE AND EXPRESSION
2496                          WILL BE CALLED, WHICH WILL ALLWAYS INTERPRET A PROC/FUNC ID
2497                        AT ITS BEGINNING AS A CALL RATHER THAN A PARAMETER PASSING.
2498                        IN THIS IMPLEMENTATION, PARAMETER PROCEDURES/FUNCTIONS
2499                        ARE THEREFORE NOT ALLOWED TO HAVE PROCEDURE/FUNCTION
2500                        PARAMETERS*)
2501                        INSYMBOL;
2502                        IF LB THEN   /*PASS FUNCTION OR PROCEDURE*/
2503                          BEGIN
2504                            IF SY # IDENT THEN ERRSKIP(21,FSYS OR LISTDELIMS)
2505                            ELSE
2506                              BEGIN
2507                                IF NXT!.KLASS = PROC THEN SEARCHID([PROC],LIP)
```

```
2508                        ELSE
2509                          BEGIN SEARCHID([FUNC],LIP);
2510                            IF ~COMPTYPES(LIP!.IDTYPE,NXT!.IDTYPE) THEN
2511                              ERROR(328)
2512                          END;
2513                        INSYMBOL;
2514                        IF ~(SY IN FSYS OR LISTDELIMS) THEN
2515                          ERRSKIP(10,FSYS OR LISTDELIMS)
2516                      END
2517                  END /*IF LB*/
2518                ELSE
2519                  BEGIN EXPRESSION(FSYS OR LISTDELIMS);
2520                    IF GDESC.TYPTR # NIL THEN
2521                      IF LKIND = ACTUAL THEN
2522                        BEGIN
2523                          IF NXT # NIL THEN
2524                            BEGIN LSP := NXT!.IDTYPE;
2525                              IF LSP # NIL THEN
2526                                BEGIN
2527                                  IF (NXT!.VKIND = ACTUAL)
2528                                  &((LSP!.SIZE = 1) OR (LSP!.FORM = SETS)) THEN
2529                                    BEGIN LOAD;
2530                                      IF COMPTYPES(FLTPTR,LSP)
2531                                        &(GDESC.TYPTR = FIXPTR) THEN
2532                                        BEGIN GEN0(10/*FLT*/);
2533                                          GDESC.TYPTR := FLTPTR
2534                                        END
2535                                    END
2536                                  ELSE
2537                                    IF GDESC.KIND = VARBL THEN
2538                                      LLOADADDRESS
2539                                    ELSE ERROR(329);
2540                                  IF ~COMPTYPES(LSP,GDESC.TYPTR) THEN
2541                                    ERROR(330)
2542                                END
2543                            END
2544                        END
2545                      ELSE /*LKIND = FORMAL*/
2546                        BEGIN /*PASS FORMAL PARAM*/
2547                        END
2548                  END;
2549                NROFPAR := NROFPAR + 1;
2550                IF (LKIND = ACTUAL)&(NXT # NIL) THEN NXT := NXT!.NEXT
2551              UNTIL SY # COMMA;
2552            IF SY = RPARENT THEN INSYMBOL ELSE ERROR(24)
2553          END /*IF LPARENT*/;
2554          IF LKIND = ACTUAL THEN
2555            BEGIN IF NXT # NIL THEN ERROR(324);
2556              WITH FIP! DO
2557                BEGIN
2558                  IF EXTERN THEN GEN1(30/*CSP*/,PFADDR)
2559                  ELSE GEN2(46/*CUP*/,NROFPAR,PFADDR);
2560                  IF PFADDR <= JMPIX THEN   /*NOTE FORWARD REFERENCE*/
2561                    JMPTAB[PFADDR] := 1
2562                END
2563            END;
2564          GDESC.TYPTR := FIP!.IDTYPE
```

```
2565            END /*CALLNONSTANDARD*/ ;
2566
2567          BEGIN /*CALL*/
2568            IF FIP!.PFDECKIND = STANDARD THEN
2569              BEGIN IF SY = LPARENT THEN INSYMBOL ELSE ERROR(25);
2570                LKEY := FIP!.KEY;
2571                IF FIP!.KLASS = PROC THEN
2572                  CASE LKEY OF
2573                    1:    PACK;
2574                    2:    UNPACK;
2575                    3:    MARK;
2576                    4:    RELEASE
2577                  END
2578                ELSE
2579                IF LKEY < 7 THEN
2580                  BEGIN EXPRESSION([FSYS OR [RPARENT]); LOAD;
2581                    CASE LKEY OF
2582                      1:    ABS;
2583                      2:    TRUNC;
2584                      3:    ORD;
2585                      4:    CHR;
2586                    5,6:    PREDSUCC
2587                    END
2588                  END
2589                ELSE
2590                  CASE LKEY OF
2591                    7:    SGN;
2592                  8,9:    MINMAX;
2593                   10:    SUM
2594                  END;
2595                IF SY = RPARENT THEN INSYMBOL ELSE ERROR(24)
2596              END /*STANDARD PROCEDURES AND FUNCTIONS*/
2597            ELSE CALLNONSTANDARD
2598          END /*CALL*/ ;
2599
2600          PROCEDURE EXPRESSION;
2601          /*ANALYZE ANY PL/I EXPRESSION*/
2602            VAR LDESC: DESC; LOP: OPERATOR; TYPIND: CHAR; LSIZE: ADDRRANGE;
2603
2604            PROCEDURE SIMPLEEXPRESSION(FSYS: SYSET);
2605              VAR LDESC: DESC; LOP: OPERATOR; SIGNED: BOOLEAN;
2606
2607              PROCEDURE TERM(FSYS: SYSET);
2608                VAR LDESC: DESC; LOP: OPERATOR;
2609
2610                PROCEDURE FACTOR(FSYS: SYSET);
2611                  VAR LIP,LIP1: IDP; LVP: CSP; VARPART: BOOLEAN;
2612                CSTPART: SET OF 0..71; LSP: STP;
2613                BEGIN
2614                  IF ~(SY IN FACBEGSYS) THEN
2615                    BEGIN ERRSKIP(501,FSYS OR FACBEGSYS);
2616                      GDESC.TYPTR := NIL
2617                    END;
2618                  IF (SY IN FACBEGSYS) THEN
2619                    BEGIN
2620                      CASE SY OF
2621                  /*ID*/    IDENT:
```

```
2622                    BEGIN SEARCHID([TYPES,KONST,VARS,FIELD,FUNC],LIP);
2623                      INSYMBOL;
2624                      IF LIP^.KLASS = FUNC THEN
2625                        BEGIN CALL(FSYS,LIP); GDESC.KIND := EXPR END
2626                      ELSE
2627                        IF LIP^.KLASS = KONST THEN
2628                          WITH GDESC, LIP^ DO
2629                            BEGIN TYPTR := IDTYPE; KIND := CST;
2630                              CVAL := VALUES
2631                            END
2632                        ELSE
2633                          BEGIN
2634                            SELECTOR(FSYS,LIP);
2635                          IF GDESC.TYPTR # NIL THEN /*ELIM. SUBR. TYPES*/
2636                            WITH GDESC, TYPTR^ DO   /*SIMPLIFY LATER TEST*/
2637                              IF FORM = SUBRANGE THEN
2638                                  TYPTR := RANGETYPE
2639                          END
2640                    END;
2641          /*CST*/   FIXCONST:
2642                    BEGIN
2643                      WITH GDESC DO
2644                        BEGIN TYPTR := FIXPTR; KIND := CST;
2645                          CVAL := VAL
2646                        END;
2647                      INSYMBOL
2648                    END;
2649                  FLTCONST:
2650                    BEGIN
2651                      WITH GDESC DO
2652                        BEGIN TYPTR := FLTPTR; KIND := CST;
2653                          CVAL := VAL
2654                        END;
2655                      INSYMBOL
2656                    END;
2657                  STRINGCONST:
2658                    BEGIN
2659                      WITH GDESC DO
2660                        BEGIN
2661                          IF LGTH = 1 THEN TYPTR := CHARPTR
2662                          ELSE
2663                            BEGIN NEW(LSP,ARRAYS);
2664                              WITH LSP^ DO
2665                                BEGIN AELTYPE := CHARPTR;
2666                                  INXTYPE := NIL; SIZE := LGTH
2667                                END;
2668                              TYPTR := LSP
2669                            END;
2670                          KIND := CST; CVAL := VAL
2671                        END;
2672                      INSYMBOL
2673                    END;
2674          /*(*/     LPARENT:
2675                    BEGIN INSYMBOL; EXPRESSION([FSYS, CR, RPARENT]);
2676                      IF SY = RPARENT THEN INSYMBOL ELSE ERROR(4)
2677                    END;
2678          /*~*/     NOTSY:
```

```
2679                    BEGIN INSYMBOL; FACTOR(FSYS);
2680                      LOAD; GEN0(19/*NOT*/);
2681                      IF GDESC.TYPTR # NIL THEN
2682                        IF GDESC.TYPTR # BOOLPTR THEN
2683                          BEGIN ERROR(502); GDESC.TYPTR := NIL END;
2684                    END;
2685          /*¢*/     LBRACK:
2686                    BEGIN INSYMBOL; CSTPART := ¢ 1; VARPART := FALSE;
2687                      NEW(LSP,SETS);
2688                      WITH LSP! DO
2689                    BEGIN ELSET := NIL; SIZE := 2 END;
2690                      IF SY = RBRACK THEN
2691                        BEGIN
2692                          WITH GDESC DO
2693                            BEGIN TYPTR := LSP; KIND := CST END;
2694                          INSYMBOL
2695                        END
2696                      ELSE
2697                        BEGIN
2698                          LOOP EXPRESSION(FSYS OR ¢COMMA,RBRACK);
2699                            IF GDESC.TYPTR # NIL THEN
2700                              IF GDESC.TYPTR!.FORM # SCALAR THEN
2701                                BEGIN ERROR(172); GDESC.TYPTR := NIL END
2702                              ELSE
2703                                IF COMPTYPES(LSP!.ELSET,GDESC.TYPTR) THEN
2704                                  BEGIN
2705                                    IF GDESC.KIND = CST THEN
2706                                  CSTPART := CSTPART OR ¢GDESC.CVAL.IVAL1
2707                                    ELSE
2708                                      BEGIN LOAD; GEN0(23/*SGS*/);
2709                                        IF VARPART THEN GEN0(28/*UNI*/)
2710                                        ELSE VARPART := TRUE
2711                                      END;
2712                                    LSP!.ELSET := GDESC.TYPTR;
2713                                    GDESC.TYPTR := LSP
2714                                  END
2715                                ELSE ERROR(173);
2716                            EXIT IF SY # COMMA;
2717                              INSYMBOL
2718                          END;
2719                          IF SY = RBRACK THEN INSYMBOL ELSE ERROR(22)
2720                        END;
2721                      IF VARPART THEN
2722                        BEGIN
2723                          IF CSTPART # ¢ 1 THEN
2724                            BEGIN NEW(LVP,PSET); LVP!.PVAL := CSTPART;
2725                              IF CSTPTRIX = CSTOCCMAX THEN ERROR(701)
2726                              ELSE
2727                                BEGIN CSTPTRIX := CSTPTRIX + 1;
2728                                  CSTPTR¢CSTPTRIX1 := LVP;
2729                                  GEN2(51/*LDC*/,5,CSTPTRIX);
2730                                  GEN0(28/*UNI*/); GDESC.KIND := EXPR
2731                                END
2732                           END
2733                          END
2734                        ELSE
2735                          BEGIN NEW(LVP,PSET); LVP!.PVAL := CSTPART;
```

```
2736                                GDESC.CVAL.VALP := LVP
2737                              END
2738                            END
2739                        END /*CASE*/ ;
2740                        IF ~(SY IN FSYS) THEN ERRSKIP(16,FSYS OR ¢SEMICOLON¢)
2741                      END /*SY IN FACBEGSYS*/
2742                  END /*FACTOR*/ ;
2743
2744                BEGIN /*TERM*/
2745                  FACTOR(FSYS OR ¢MULOP¢);
2746                  WHILE SY = MULOP DO
2747                          BEGIN LOAD; LDESC := GDESC; LOP := OP;
2748                      INSYMBOL; FACTOR(FSYS OR ¢MULOP¢); LOAD;
2749                      IF (LDESC.TYPTR # NIL)&(GDESC.TYPTR # NIL) THEN
2750                        CASE LOP OF
2751 /***/                MUL:  IF (LDESC.TYPTR = FIXPTR)&(GDESC.TYPTR = FIXPTR)
2752                            THEN GEN0(15/*MPI*/)
2753                          ELSE
2754                            BEGIN
2755                              IF LDESC.TYPTR = FIXPTR THEN
2756                                BEGIN GEN0(9/*FLO*/);
2757                                  LDESC.TYPTR := FLTPTR
2758                                END
2759                              ELSE
2760                                IF GDESC.TYPTR = FIXPTR THEN
2761                                  BEGIN GEN0(10/*FLT*/);
2762                                    GDESC.TYPTR := FLTPTR
2763                                  END;
2764                              IF (LDESC.TYPTR = FLTPTR)
2765                                &(GDESC.TYPTR = FLTPTR) THEN GEN0(16/*MPR*/)
2766                              ELSE BEGIN ERROR(503); GDESC.TYPTR := NIL END
2767                            END;
2768 /*/*/               RDIV: BEGIN
2769                            IF LDESC.TYPTR = FIXPTR THEN
2770                              BEGIN GEN0(9/*FLO*/);
2771                                LDESC.TYPTR := FLTPTR
2772                              END;
2773                            IF GDESC.TYPTR = FIXPTR THEN
2774                                BEGIN GEN0(10/*FLT*/);
2775                                GDESC.TYPTR := FLTPTR
2776                              END;
2777                            IF (LDESC.TYPTR = FLTPTR)
2778                              &(GDESC.TYPTR = FLTPTR) THEN GEN0(7/*DVR*/)
2779                            ELSE BEGIN ERROR(503); GDESC.TYPTR := NIL END
2780                          END;
2781 /*DIV*/            IDIV: IF (LDESC.TYPTR = FIXPTR)
2782                            &(GDESC.TYPTR = FIXPTR) THEN GEN0(6/*DVI*/)
2783                          ELSE BEGIN ERROR(503); GDESC.TYPTR := NIL END;
2784 /*MOD*/            IMOD: IF (LDESC.TYPTR = FIXPTR)
2785                            &(GDESC.TYPTR = FIXPTR) THEN GEN0(14/*MOD*/)
2786                          ELSE BEGIN ERROR(503); GDESC.TYPTR := NIL END;
2787 /*&*/              ANDOP:IF (LDESC.TYPTR = BOOLPTR)
2788                            &(GDESC.TYPTR = BOOLPTR) THEN GEN0(4/*AND*/)
2789                          ELSE
2790                            IF (LDESC.TYPTR!.FORM = SETS)
2791                              &COMPTYPES(LDESC.TYPTR,GDESC.TYPTR) THEN
2792                              GEN0(12/*INT*/)
```

```
2793                          ELSE BEGIN ERROR(503); GDESC.TYPTR := NIL END
2794                    END /*CASE*/
2795                  ELSE GDESC.TYPTR := NIL
2796                END /*WHILE*/
2797            END /*TERM*/ ;
2798
2799          BEGIN /*SIMPLEEXPRESSION*/
2800            SIGNED := FALSE;
2801            IF (SY = ADDOP)&(OP IN ¢PLUS,MINUS¢) THEN
2802              BEGIN SIGNED := OP = MINUS; INSYMBOL END;
2803            POSV := ¬SIGNED;
2804            TERM(FSYS OR ¢ADDOP¢);
2805            IF SIGNED THEN
2806              BEGIN LOAD;
2807                IF GDESC.TYPTR = FIXPTR THEN GEN0(17/*NGI*/)
2808                ELSE
2809                  IF GDESC.TYPTR = FLTPTR THEN GEN0(18/*NGR*/)
2810                  ELSE BEGIN ERROR(503); GDESC.TYPTR := NIL END
2811              END;
2812            WHILE SY = ADDOP DO
2813              BEGIN LOAD; LDESC := GDESC; LOP := OP;
2814                INSYMBOL; TERM(FSYS OR ¢ADDOP¢); LOAD;
2815                IF (LDESC.TYPTR # NIL)&(GDESC.TYPTR # NIL) THEN
2816                  CASE LOP OF
2817      /***/          PLUS:
2818                      IF (LDESC.TYPTR = FIXPTR)&(GDESC.TYPTR = FIXPTR) THEN
2819                        GEN0(2/*ADI*/)
2820                      ELSE
2821                        BEGIN
2822                          IF LDESC.TYPTR = FIXPTR THEN
2823                            BEGIN GEN0(9/*FLO*/);
2824                              LDESC.TYPTR := FLTPTR
2825                            END
2826                          ELSE
2827                            IF GDESC.TYPTR = FIXPTR THEN
2828                              BEGIN GEN0(10/*FLT*/);
2829                                GDESC.TYPTR := FLTPTR
2830                              END;
2831                          IF (LDESC.TYPTR = FLTPTR)&(GDESC.TYPTR = FLTPTR)
2832                            THEN GEN0(3/*ADR*/)
2833                          ELSE BEGIN ERROR(503); GDESC.TYPTR := NIL END
2834                        END;
2835      /*-*/           MINUS:
2836                      IF (LDESC.TYPTR = FIXPTR)&(GDESC.TYPTR = FIXPTR) THEN
2837                        GEN0(21/*SBI*/)
2838                      ELSE
2839                        BEGIN
2840                          IF LDESC.TYPTR = FIXPTR THEN
2841                            BEGIN GEN0(9/*FLO*/);
2842                              LDESC.TYPTR := FLTPTR
2843                            END
2844                          ELSE
2845                            IF GDESC.TYPTR = FIXPTR THEN
2846                            BEGIN GEN0(10/*FLT*/);
2847                                GDESC.TYPTR := FLTPTR
2848                              END;
2849                          IF (LDESC.TYPTR = FLTPTR)&(GDESC.TYPTR = FLTPTR)
```

```
2850                              THEN GEN0(22/*SBR*/)
2851                            ELSE
2852                              IF (LDESC.TYPTR!.FORM = SETS)
2853                                &COMPTYPES(LDESC.TYPTR,GDESC.TYPTR) THEN
2854                                GEN0(5/*DIF*/)
2855                                ELSE BEGIN ERROR(503); GDESC.TYPTR := NIL END
2856                          END;
2857          /* OR */        OROP:
2858                        IF (LDESC.TYPTR = BOOLPTR)&(GDESC.TYPTR = BOOLPTR) THEN
2859                          GEN0(13/*IOR*/)
2860                        ELSE
2861                          IF (LDESC.TYPTR!.FORM = SETS)
2862                            &COMPTYPES(LDESC.TYPTR,GDESC.TYPTR) THEN
2863                            GEN0(28/*UNI*/)
2864                          ELSE BEGIN ERROR(503); GDESC.TYPTR := NIL END
2865                    END /*CASE*/
2866                  ELSE GDESC.TYPTR := NIL
2867                END /*WHILE*/
2868            END /*SIMPLEEXPRESSION*/ ;
2869
2870          BEGIN /*EXPRESSION*/
2871            SIMPLEEXPRESSION(FSYS OR [RELOP,BECOMES]);
2872            IF SY = BECOMES THEN SY := RELOP;
2873            IF SY = RELOP THEN
2874              BEGIN
2875                IF GDESC.TYPTR # NIL THEN
2876                  IF GDESC.TYPTR!.FORM <= SETS THEN LOAD
2877                  ELSE LOADADDRESS;
2878                  LDESC := GDESC; LOP := OP;
2879                INSYMBOL; SIMPLEEXPRESSION(FSYS);
2880                IF GDESC.TYPTR # NIL THEN
2881                  IF GDESC.TYPTR!.FORM <= SETS THEN LOAD
2882              ELSE LOADADDRESS;
2883                IF (LDESC.TYPTR # NIL)&(GDESC.TYPTR # NIL) THEN
2884                  IF LOP = INOP THEN
2885                    IF GDESC.TYPTR!.FORM = SETS THEN
2886                      IF COMPTYPES(LDESC.TYPTR,GDESC.TYPTR!.ELSET) THEN
2887                        GEN0(11/*INN*/)
2888                      ELSE BEGIN ERROR(504); GDESC.TYPTR := NIL END
2889                    ELSE BEGIN ERROR(174); GDESC.TYPTR := NIL END
2890                  ELSE
2891                    BEGIN
2892                      IF LDESC.TYPTR # GDESC.TYPTR THEN
2893                        IF LDESC.TYPTR = FIXPTR THEN
2894                          BEGIN GEN0(9/*FLO*/);
2895                            LDESC.TYPTR := FLTPTR
2896                          END
2897                        ELSE
2898                          IF GDESC.TYPTR = FIXPTR THEN
2899                            BEGIN GEN0(10/*FLT*/);
2900                              GDESC.TYPTR := FLTPTR
2901                            END;
2902                      IF COMPTYPES(LDESC.TYPTR,GDESC.TYPTR) THEN
2903                        BEGIN LSIZE := LDESC.TYPTR!.SIZE;
2904                          CASE LDESC.TYPTR!.FORM OF
2905                            SCALAR:
2906                  IF LDESC.TYPTR = FLTPTR THEN TYPIND := 'R'
```

```
2907                              ELSE
2908                                IF LDESC.TYPTR = BOOLPTR THEN TYPIND := 'B'
2909                                ELSE TYPIND := 'I';
2910                            POINTER:
2911                              BEGIN
2912                                IF LOP IN ¢LTOP,LEOP,GTOP,GEOP¡ THEN ERROR(520);
2913                                TYPIND := 'A'
2914                              END;
2915                            SETS:
2916                              BEGIN IF LOP IN ¢LTOP,GTOP¡ THEN ERROR(521);
2917                                TYPIND := 'S'
2918                            END;
2919                            ARRAYS:
2920                              BEGIN
2921                                IF ~STRING(LDESC.TYPTR)
2922                                &(LOP IN ¢LTOP,LEOP,GTOP,GEOP¡) THEN ERROR(522);
2923                                TYPIND := 'M'
2924                              END;
2925                            RECORDS:
2926                              BEGIN
2927                                IF LOP IN ¢LTOP,LEOP,GTOP,GEOP¡ THEN ERROR(520);
2928                                TYPIND := 'M'
2929                              END;
2930                            FILES:
2931                          BEGIN ERROR(522); TYPIND := 'F' END
2932                          END;
2933                          CASE LOP OF
2934                            LTOP: GEN2(53/*LES*/,ORD(TYPIND),LSIZE);
2935                            LEOP: GEN2(52/*LEQ*/,ORD(TYPIND),LSIZE);
2936                            GTOP: GEN2(49/*GRT*/,ORD(TYPIND),LSIZE);
2937                            GEOP: GEN2(48/*GEQ*/,ORD(TYPIND),LSIZE);
2938                            NEOP: GEN2(55/*NEQ*/,ORD(TYPIND),LSIZE);
2939                            EQOP: GEN2(47/*EQU*/,ORD(TYPIND),LSIZE);
2940                            NOOP: ERROR(523)
2941                          END
2942                        END
2943                      ELSE ERROR(504)
2944                    END;
2945                  GDESC.TYPTR := BOOLPTR; GDESC.KIND := EXPR
2946                END /*SY = RELOP*/
2947            END /*EXPRESSION*/ ;
2948
2949            /* FOLLOWING ARE THE STATEMENT PROCESSORS FOR
2950               ASSIGNMENT, RETURN STATEMENTS, GET, PUT,
2951               GOTO'S, COMPOUNDSTATEMENTS, IF, CASE, REPEAT,
2952               DO, LOOP, ALLOCATE, ON, AND STOP STATEMENTS.
2953
2954               STATEMENT PROCESSORS MAY BE CALLED INDIRECT-
2955               RECURSIVELY THROUGH PROCEDURES COMPOUNDSTATEMENT,
2956               IFSTATEMENT, CASESTATEMENT, REPEATSTATEMENT,
2957               DOSTATEMENT, LOOPSTATEMENT, AND ONSTATEMENT. */
2958
2959            PROCEDURE ASSIGNMENT(FIP: IDP);
2960              VAR LDESC: ARRAY¢1..10¡ OF DESC;
2961                  LIP: IDP; P1,P2,I,TIX: INTEGER;
2962            BEGIN TIX := 0; LIP := FIP;
2963              LOOP
```

```
2964                TIX := TIX + 1;
2965                SELECTOR(FSYS OR ¢BECOMES,COMMA!,LIP);
2966                IF GDESC.TYPTR # NIL THEN
2967                  IF (GDESC.ACCESS # DRCT) OR (GDESC.TYPTR!.FORM > SETS) THEN
2968                    LOADADDRESS;
2969                LDESC¢TIX! := GDESC;
2970              EXIT IF SY # COMMA;
2971                INSYMBOL;
2972                SEARCHID(¢VARS,FIELD,TYPES!,LIP); INSYMBOL
2973              END;
2974              IF SY = BECOMES THEN
2975                BEGIN INSYMBOL;
2976                  IF GDESC.TYPTR!.FORM = LLABL THEN
2977                    BEGIN
2978                      IF SY = IDENT THEN
2979                        BEGIN PRTERR := FALSE;
2980                          SEARCHID(¢VARS,TYPES!,LIP); PRTERR := TRUE;
2981                          IF LIP # NIL THEN SELECTOR(FSYS,LIP)
2982                          ELSE LABELCONSTANT(P1,P2)
2983                        END
2984                      ELSE ERROR(21)
2985                    END
2986                  ELSE EXPRESSION(FSYS);
2987                  FOR I := TIX DOWNTO 1 DO
2988                    BEGIN
2989                      IF GDESC.TYPTR # NIL THEN
2990                        IF GDESC.TYPTR!.FORM = LLABL THEN GEN2(51/*LDC*/,P1,P2)
2991                        ELSE
2992                        IF GDESC.TYPTR!.FORM <= SETS THEN LOAD
2993                        ELSE LOADADDRESS;
2994                      IF (LDESC¢I!.TYPTR # NIL)&(GDESC.TYPTR # NIL) THEN
2995                        BEGIN
2996                          IF COMPTYPES(FLTPTR,LDESC¢I!.TYPTR)
2997                                        &(GDESC.TYPTR = FIXPTR) THEN
2998                            BEGIN GEN0(10/*FLT*/);
2999                              GDESC.TYPTR := FLTPTR
3000                            END
3001                          ELSE
3002                          IF COMPTYPES(FIXPTR,LDESC¢I!.TYPTR)
3003                                        &(GDESC.TYPTR = FLTPTR) THEN
3004                            BEGIN GEN0(27/*TRC*/); GDESC.TYPTR := FIXPTR END;
3005                          IF COMPTYPES(LDESC¢I!.TYPTR,GDESC.TYPTR) THEN
3006                            CASE LDESC¢I!.TYPTR!.FORM OF
3007                              LLABL,
3008                              SCALAR,
3009                              SUBRANGE,
3010                              POINTER,
3011                              SETS:   STORE(LDESC¢I!);
3012                              ARRAYS,
3013                              RECORDS: GEN1(40/*MCV*/,LDESC¢I!.TYPTR!.SIZE);
3014                              FILES: ERROR(182)
3015                            END
3016                          ELSE ERROR(504)
3017                        END
3018                      IF I >= 2 THEN GDESC := LDESC¢I-1!;
3019                    END;
3020                END /*SY = BECOMES*/
```

```
3021                ELSE ERROR(480)
3022              END /*ASSIGNMENT*/ ;
3023
3024              PROCEDURE RETURNSTATEMENT;
3025                VAR LDESC: DESC; LIP: IDP;
3026              BEGIN
3027                IF FBTYP = FUNCBLK THEN
3028                  BEGIN
3029                    IF SY # SEMICOLON THEN
3030                      BEGIN
3031                        LIP := FPROCP; LDESC.TYPTR := LIP!.IDTYPE;
3032                        EXPRESSION(FSYS);
3033                        IF GDESC.TYPTR # NIL THEN
3034                          IF GDESC.TYPTR!.FORM <= SETS THEN LOAD
3035                          ELSE LOADADDRESS;
3036                        IF (LDESC.TYPTR # NIL)&(GDESC.TYPTR # NIL) THEN
3037                          BEGIN
3038                            IF COMPTYPES(FLTPTR,LDESC.TYPTR)
3039                                          &(GDESC.TYPTR = FIXPTR) THEN
3040                              BEGIN GEN0(10/*FLT*/);
3041                                GDESC.TYPTR := FLTPTR
3042                              END
3043                            ELSE
3044                            IF COMPTYPES(FIXPTR,LDESC.TYPTR)
3045                                          &(GDESC.TYPTR = FLTPTR) THEN
3046                              BEGIN GEN0(27/*TRC*/); GDESC.TYPTR := FIXPTR END;
3047                            IF COMPTYPES(LDESC.TYPTR,GDESC.TYPTR) THEN
3048                              CASE LDESC.TYPTR!.FORM OF
3049                                LLABL,
3050                                SCALAR,
3051                                SUBRANGE,
3052                                POINTER,
3053                                SETS:   GEN2(56/*STR*/,0,0);
3054                                ARRAYS,RECORDS,FILES: ERROR(307)
3055                              END
3056                            ELSE ERROR(504)
3057                          END
3058                      END;
3059                    GEN2(57/*UJP*/,0,0);
3060                    FRETNIX := FRETNIX + 1; FUNCRETN^[FRETNIX] := CIX;
3061                    NXTSTLAB := TRUE
3062                  END
3063                ELSE ERROR(400);
3064              END /*RETURNSTATEMENT*/;
3065
3066              PROCEDURE GETSTATEMENT;
3067                VAR LIP: IDP; BUFFADDR: ADDRRANGE;
3068              BEGIN BUFFADDR := 4;
3069                FILENAME(LIP);
3070                IF LIP # UVARPTR THEN
3071                  BEGIN BUFFADDR := LIP!.VADDR;
3072                    WITH LIP!.IDTYPE! DO
3073                      IF EOFADDR # 0 THEN
3074                        BEGIN GEN2(8/*EOF*/,LIP!.VLEV,BUFFADDR);
3075                          GEN1(33/*FJP*/,IC+2); GEN2(57/*UJP*/,0,EOFADDR)
3076                        END;
3077                  END;
```

```
3078                IF SY = LISTSY THEN
3079                  BEGIN INSYMBOL;
3080                    IF SY = LPARENT THEN INSYMBOL
3081                    ELSE ERRSKIP(25,FSYS OR ¢LPARENT,SEMICOLON¢);
3082                    IF SY = IDENT THEN
3083                      LOOP GEN2(50/*LDA*/,LEVEL-LIP!.VLEV,BUFFADDR);
3084                        VARIABLE(FSYS OR LISTDELIMS); LOADADDRESS;
3085                        IF GDESC.TYPTR # NIL THEN
3086                          IF GDESC.TYPTR!.FORM <= SUBRANGE THEN
3087                            IF COMPTYPES(FIXPTR,GDESC.TYPTR) THEN
3088                              GEN1(30/*CSP*/,3/*RDI*/)
3089                            ELSE
3090                              IF COMPTYPES(FLTPTR,GDESC.TYPTR) THEN
3091                                GEN1(30/*CSP*/,4/*RDR*/)
3092                              ELSE
3093                                IF COMPTYPES(CHARPTR,GDESC.TYPTR) THEN
3094                                  GEN1(30/*CSP*/,5/*RDC*/)
3095                                ELSE ERROR(460)
3096                          ELSE ERROR(326);
3097                        EXIT IF SY # COMMA;
3098                        INSYMBOL
3099                      END;
3100                  END
3101                ELSE
3102                IF SY = EDITSY THEN ERRSKIP(403,FSYS OR ¢SEMICOLON¢)
3103                ELSE ERRSKIP(10,FSYS OR ¢SEMICOLON¢);
3104                INSYMBOL
3105              END /*GETSTATEMENT*/ ;
3106
3107              PROCEDURE PUTSTATEMENT;
3108                VAR LIP: IDP; LSP: STP; BUFFADDR: ADDRRANGE; DEFAULT: BOOLEAN;
3109
3110                PROCEDURE PUTOPTIONS;
3111                BEGIN
3112                  IF SY = SKIPSY THEN
3113                    BEGIN INSYMBOL;
3114                      IF SY = LPARENT THEN ERRSKIP(403,FSYS OR ¢SEMICOLON,LISTSY¢);
3115                      GEN2(51/*LDC*/,1,0); GEN2(51/*LDC*/,1,2);
3116                      GEN1(30/*CSP*/,9/*WRC*/)
3117                    END
3118                  ELSE
3119                  IF SY = PAGESY THEN
3120                    BEGIN GEN2(51/*LDC*/,1,0); GEN2(51/*LDC*/,1,3);
3121                      GEN1(30/*CSP*/,9/*WRC*/); INSYMBOL
3122                    END;
3123                END /*PUTOPTIONS*/ ;
3124
3125              BEGIN BUFFADDR := 4*BUFFSIZE; FILENAME(LIP);
3126                IF LIP # UVARPTR THEN BUFFADDR := LIP!.VADDR;
3127                PUTOPTIONS;
3128                IF SY = LISTSY THEN
3129                  BEGIN INSYMBOL;
3130                    IF SY = LPARENT THEN INSYMBOL
3131                    ELSE ERRSKIP(25,FSYS OR ¢LPARENT,SEMICOLON¢);
3132                    IF (SY IN ¢IDENT,FIXCONST,FLTCONST,STRINGCONST¢) THEN
3133                      LOOP GEN2(50/*LDA*/,LEVEL-LIP!.VLEV,BUFFADDR);
3134                        EXPRESSION(FSYS OR ¢COMMA,COLON,RPARENT¢);
```

```
3135                    LSP := GDESC.TYPTR;
3136                    IF LSP # NIL THEN
3137                      IF LSP!.FORM <= SETS THEN LOAD ELSE LOADADDRESS;
3138                    IF SY = COLON THEN
3139                      BEGIN INSYMBOL; EXPRESSION(FSYS OR ¢COMMA,COLON,RPARENT);
3140                        IF GDESC.TYPTR # NIL THEN
3141                          IF GDESC.TYPTR # FIXPTR THEN ERROR(326);
3142                        LOAD; DEFAULT := FALSE
3143                      END
3144                    ELSE DEFAULT := TRUE;
3145                    IF SY = COLON THEN
3146                      BEGIN INSYMBOL; EXPRESSION(FSYS OR LISTDELIMS);
3147                        IF GDESC.TYPTR # NIL THEN
3148                          IF GDESC.TYPTR # FIXPTR THEN ERROR(324);
3149                        IF LSP # FLTPTR THEN ERROR(401);
3150                        LOAD; ERROR(403);
3151                      END
3152                    ELSE
3153                      IF LSP = FIXPTR THEN
3154                        BEGIN IF DEFAULT THEN GEN2(51/*LDC*/,1,10);
3155                          GEN1(30/*CSP*/,6/*WRI*/)
3156                        END
3157                    ELSE
3158                    IF LSP = FLTPTR THEN
3159                      BEGIN IF DEFAULT THEN GEN2(51/*LDC*/,1,20);
3160                        GEN1(30/*CSP*/,8/*WRR*/)
3161                      END
3162                    ELSE
3163                    IF LSP = CHARPTR THEN
3164                      BEGIN IF DEFAULT THEN GEN2(51/*LDC*/,1,1);
3165                        GEN1(30/*CSP*/,9/*WRC*/)
3166                      END
3167                    ELSE
3168                    IF LSP # NIL THEN
3169                      BEGIN
3170                        IF LSP!.FORM = SCALAR THEN ERROR(400)
3171                        ELSE
3172                          IF STRING(LSP) THEN
3173                            BEGIN IF DEFAULT THEN GEN2(51/*LDC*/,1,LSP!.SIZE);
3174                              GEN2(51/*LDC*/,1,LSP!.SIZE);
3175                              GEN1(30/*CSP*/,10/*WRS*/)
3176                            END
3177                          ELSE
3178                          IF LSP!.FORM = SETS THEN
3179                            BEGIN GEN2(51/*LDC*/,1,15); GEN1(30/*CSP*/,6/*WRI*/);
3180                              GEN2(51/*LDC*/,1,15); GEN1(30/*CSP*/,6/*WRI*/);
3181                            END
3182                        ELSE ERROR(326)
3183                      END;
3184                  EXIT IF SY # COMMA;
3185                    INSYMBOL
3186                  END;
3187                INSYMBOL
3188              END
3189            ELSE
3190            IF SY = EDITSY THEN ERRSKIP(403,FSYS OR ¢SEMICOLON)
3191            ELSE IF SY # SEMICOLON THEN ERRSKIP(10,FSYS OR ¢SEMICOLON);
```

```
3306                              GOTO 1
3307                            END;
3308                          LPT2 := LPT1; LPT1 := NEXT
3309                        END;
3310          1:          NEW(LPT3);
3311                      WITH LPT3! DO
3312                        BEGIN NEXT := LPT1; CSLAB := LVAL.IVAL;
3313                          CSSTART := IC; CSEND := 0
3314                        END;
3315                      IF LPT2 = NIL THEN FSTPTR := LPT3
3316                      ELSE LPT2!.NEXT := LPT3
3317                    END
3318                  ELSE ERROR(441);
3319              EXIT IF SY # COMMA;
3320                INSYMBOL;
3321              END;
3322              IF SY = COLON THEN INSYMBOL ELSE ERROR(2E);
3323              REPEAT STATEMENT(FSYS OR [SEMICOLON])
3324              UNTIL ~(SY IN STATMENTSYS);
3325              IF LPT3 # NIL THEN
3326                BEGIN GEN2(57/*UJP*/,0,0); LPT3!.CSEND := CIX END;
3327            EXIT IF SY # SEMICOLON;
3328              INSYMBOL
3329            END;
3330            IF FSTPTR # NIL THEN
3331              BEGIN LMAX := FSTPTR!.CSLAB;
3332                /*REVERSE POINTERS*/
3333                LPT1 := FSTPTR; FSTPTR := NIL;
3334                REPEAT LPT2 := LPT1!.NEXT; LPT1!.NEXT := FSTPTR;
3335                  FSTPTR := LPT1; LPT1 := LPT2
3336                UNTIL LPT1 = NIL;
3337                LMIN := FSTPTR!.CSLAB;
3338                INSERT(LCIX+IC-LMIN);
3339                IF LMAX - LMIN < CIXMAX THEN
3340                  BEGIN LADDR := IC + LMAX - LMIN + 1;
3341                    REPEAT
3342                      WITH FSTPTR! DO
3343                        BEGIN
3344                          WHILE CSLAB > LMIN DO
3345                            BEGIN GEN2(57/*UJP*/,0,LADDR); LMIN := LMIN + 1 END;
3346                          GEN2(57/*UJP*/,0,CSSTART);
3347                          IF CSEND # 0 THEN INSERT(CSEND,LADDR);
3348                          FSTPTR := NEXT; LMIN := LMIN + 1
3349                        END
3350                    UNTIL FSTPTR = NIL
3351                  END
3352                ELSE ERROR(442)
3353              END;
3354              IF SY = ENDSY THEN INSYMBOL ELSE ERROR(600)
3355          END /*CASESTATEMENT*/ ;
3356
3357          PROCEDURE REPEATSTATEMENT;
3358            VAR LADDR: ADDRRANGE;
3359          BEGIN LADDR := IC;
3360            LOOP
3361              REPEAT STATEMENT(FSYS OR [SEMICOLON,UNTILSY])
3362              UNTIL ~(SY IN STATMENTSYS);
```

```
3534                 FOR IX := 1 TO ITEMX DO INSERT(LCIX^IX],IC);
3535                 LOOP
3536                   REPEAT STATEMENT(FSYS OR STATDELIMS)
3537                   UNTIL ~(SY IN STATMENTSYS);
3538                 EXIT IF SY # SEMICOLON;
3539                   INSYMBOL
3540                 END;
3541                 IF NIX > 0 THEN
3542                   FOR IX := 1 TO NIX DO INSERT(NXTIX^IX],IC);
3543                 IF THIX > 1 THEN
3544                   BEGIN
3545                     GEN2(54/*LOD*/,0,THUNKX); GEN1(44/*XJP*/,IC);
3546                     FOR IX := 2 TO THIX DO GEN2(57/*UJP*/,0,LADDR^IX]);
3547                   END;
3548                 LC := THUNKX;
3549               END /*~SY IN STATBEGSYS*/ ;
3550             IF SY = ENDSY THEN ENDBLOCK(DOBLK) ELSE ERROR(600);
3551           END /*COSTATEMENT*/ ;
3552
3553           PROCEDURE LOOPSTATEMENT;
3554             VAR LADDR: ADDRRANGE; LCIX: CODERANGE;
3555           BEGIN LADDR := IC;
3556             LOOP
3557               REPEAT STATEMENT(FSYS OR ^SEMICOLON,EXITSY])
3558               UNTIL ~(SY IN STATMENTSYS);
3559             EXIT IF SY # SEMICOLON;
3560               INSYMBOL
3561             END;
3562             IF SY = EXITSY THEN
3563               BEGIN INSYMBOL;
3564                 IF SY = IFSY THEN
3565                   BEGIN INSYMBOL; EXPRESSION(FSYS OR STATDELIMS);
3566                     LOAD; GEN0(19/*NOT*/);
3567                   END
3568                 ELSE ERRSKIP(19,FSYS OR STATDELIMS);
3569                 GENFJP(0); LCIX := CIX;
3570                 LOOP
3571                   REPEAT STATEMENT(FSYS OR STATDELIMS)
3572                   UNTIL ~(SY IN STATMENTSYS);
3573                 EXIT IF SY # SEMICOLON;
3574                   INSYMBOL
3575                 END;
3576                 GEN2(57/*UJP*/,0,LADDR); INSERT(LCIX,IC)
3577               END
3578             ELSE ERROR(20);
3579             IF  SY = ENDSY THEN INSYMBOL ELSE ERROR(600)
3580           END /*LOOPSTATEMENT*/ ;
3581
3582           PROCEDURE ALLOCATESTATEMENT;
3583             VAR LIP,LIP1: IDP;
3584           BEGIN
3585             LOOP
3586               IF SY = IDENT THEN
3587                 BEGIN SEARCHID(^TYPES],LIP);
3588                   IF LIP = UTYPPTR THEN ERROR(131)
3589                   ELSE
3590                     BEGIN INSYMBOL;
```

```
3591                        IF SY = SETSY THEN
3592                          BEGIN INSYMBOL;
3593                            IF SY = LPARENT THEN INSYMBOL ELSE ERROR(25);
3594                            IF SY = IDENT THEN
3595                              BEGIN VARIABLE(FSYS OR [RPARENT]); LOADADDRESS END
3596                            ELSE ERROR(183);
3597                            IF SY = RPARENT THEN INSYMBOL ELSE ERROR(24)
3598                          END
3599                        ELSE
3600                         WITH LIP! DO
3601                          BEGIN
3602                            IF ~BASED THEN ERROR(470)
3603                            ELSE SELECTOR(FSYS OR SEMICOMMA,LIP!.POINTERID);
3604                            LOADADDRESS
3605                          END;
3606                        IF GDESC.TYPTR # NIL THEN
3607                          IF GDESC.TYPTR!.FORM # POINTER THEN ERROR(181)
3608                          ELSE
3609                          IF ~COMPTYPES(GDESC.TYPTR!.ELTYPE,LIP!.IDTYPE)
3610                          THEN ERROR(508);
3611                        GEN2(51/*LDC*/,1,0); GEN2(51/*LDC*/,1,LIP!.IDTYPE!.SIZE);
3612                        GEN1(30/*CSP*/,12/*NEW*/);
3613                      END /*LIP # UTYPPTR*/;
3614                  END /*SY = IDENT*/
3615                ELSE ERROR(21);
3616              EXIT IF SY # COMMA;
3617                INSYMBOL
3618              END;
3619            END /*ALLOCATESTATEMENT*/ ;
3620
3621            PROCEDURE UNSTATEMENT;
3622              VAR LCIX: CODERANGE; LIP: IDP;
3623            BEGIN
3624              IF ID = 'ENDFILE    ' THEN
3625                BEGIN SY := FILESY; FILENAME(LIP);
3626                  GEN2(57/*UJP*/,0,0); LCIX := CIX;
3627                  LIP!.IDTYPE!.EOFADDR := IC;
3628                  STATEMENT(FSYS); GEN0(29/*STP*/);
3629                  INSERT(LCIX,IC); NXTSTLAB := FALSE
3630                END
3631              ELSE ERRSKIP(403,FSYS OR STATDELIMS);
3632            END /*UNSTATEMENT*/ ;
3633
3634            PROCEDURE STOPSTATEMENT;
3635            BEGIN
3636              GEN0(29/*STP*/);
3637              NXTSTLAB := TRUE;
3638              INSYMBOL
3639            END /*STOPSTATEMENT*/;
3640
3641          BEGIN /*STATEMENT*/
3642            /*HANDLE LABELS, SETTING UP LABEL CELLS IF NECESSARY*/
3643            IF SY = IDENT THEN
3644              BEGIN
3645                IF ~STATLAB THEN
3646                  BEGIN IF LASTSY # OTHERSY THEN SY := LASTSY ELSE INSYMBOL;
3647                    IF SY = COLON THEN
```

```
3648                    BEGIN STATLAB := TRUE; LLABEL := ID; INSYMBOL END
3649                  ELSE
3650                    BEGIN LASTSY := SY; SY := IDENT END;
3651                END;
3652            END;
3653          IF SY # ENDSY THEN
3654            IF NXTSTLAB & ~STATLAB THEN ERROR(422);
3655          NXTSTLAB := FALSE;
3656          IF STATLAB THEN
3657            BEGIN LLP := FSTLABP;
3658              WHILE LLP # FLABP DO
3659                  IF LLP!.LABNAME = LLABEL THEN
3660                    IF LLP!.DEFINED = FALSE THEN
3661                      BEGIN LLP!.DEFINED := TRUE;
3662                        LLP!.LABADDR := IC;
3663                        LLP!.LABLEV := BLOCKS¢BIX!.BEGLEV; GOTO 1
3664                      END
3665                    ELSE
3666                    BEGIN ERROR(420); GOTO 1 END
3667                  ELSE LLP := LLP!.NEXTLAB;
3668              NEW(LLP);
3669              WITH LLP! DO
3670                BEGIN
3671                  LABNAME := LLABEL;
3672                  LABLEV := BLOCKS¢BIX!.BEGLEV;
3673                  LABADDR := IC;
3674                  NEXTLAB := FSTLABP;
3675                  DEFINED := TRUE
3676                END;
3677              FSTLABP := LLP;
3678            END
3679          ELSE LLABEL := '          ';
3680          /*DRIVE THE RESPECTIVE STATEMENT PROCESSOR,
3681            DEPENDING ON THE CURRENT STATEMENT-BEGIN-
3682            SYMBOL IN 'SY' */
3683      1:  IF ~(SY IN FSYS OR ¢IDENT!) THEN ERRSKIP(10,FSYS);
3684          IF SY IN STATMENTSYS THEN
3685            BEGIN
3686              IF SY = IDENT THEN
3687                BEGIN SEARCHID(¢TYPES,VARS,FIELD,FUNC,PROC!,LIP);
3688                  IF STATLAB THEN INSYMBOL ELSE SY := LASTSY;
3689                  STATLAB := FALSE; LASTSY := OTHERSY;
3690                  IF LIP!.KLASS = PROC THEN CALL(FSYS,LIP)
3691                  ELSE ASSIGNMENT(LIP)
3692                END
3693              ELSE
3694                BEGIN STATLAB := FALSE;
3695                CASE SY OF
3696                CALLSY:   BEGIN INSYMBOL; SEARCHID(¢PROC!,LIP);
3697                            IF LIP # UPRCPTR THEN
3698                              BEGIN INSYMBOL; CALL(FSYS,LIP) END;
3699                          END;
3700                BEGINSY:  BEGIN BEGBLOCK(BEGINBLK); INSYMBOL;
3701                            COMPOUNDSTATEMENT END;
3702                GETSY:    BEGIN INSYMBOL; GETSTATEMENT END;
3703                PUTSY:    BEGIN INSYMBOL; PUTSTATEMENT END;
3704                GOSY:     BEGIN INSYMBOL;
```

```
3705                              IF SY = TOSY THEN
3706                                BEGIN INSYMBOL;  GOTOSTATEMENT END;
3707                            END;
3708                    GOTOSY:   BEGIN INSYMBOL; GOTOSTATEMENT END;
3709                    IFSY:     BEGIN INSYMBOL; IFSTATEMENT END;
3710                    CASESY:   BEGIN INSYMBOL; CASESTATEMENT END;
3711                    REPEATSY: BEGIN INSYMBOL; REPEATSTATEMENT END;
3712                    LOOPSY:   BEGIN INSYMBOL; LOOPSTATEMENT END;
3713                    DOSY:     BEGIN INSYMBOL; DOSTATEMENT END;
3714                    RETURNSY: BEGIN INSYMBOL; RETURNSTATEMENT END;
3715                    ALLOCATESY: BEGIN INSYMBOL; ALLOCATESTATEMENT END;
3716                    FREESY,
3717                    ENTRYSY,
3718                    CHECKSY,
3719                    FORMATSY: ERRSKIP(401,FSYS OR ¢SEMICOLON¢);
3720                    ONSY:     BEGIN INSYMBOL; ONSTATEMENT END;
3721                    STOPSY:   STOPSTATEMENT
3722                  END;
3723                END;
3724                  IF ~(SY IN FSYS) THEN ERRSKIP(10,FSYS)
3725                END
3726            END /*STATEMENT*/ ;
3727
3728          BEGIN BOD := TRUE; /*BODY*/
3729            IF FBTYP NE BEGINBLK THEN
3730              IF FPROCP # MAINP THEN
3731                WITH FPROCP! DO
3732                  BEGIN
3733                    IF JMPTAB¢PFADDR¢ = 0 THEN   /*NO FORWARD REFERENCE*/
3734                      BEGIN IF JMPIX = PFADDR THEN JMPIX := JMPIX - 1 END
3735                    ELSE JMPTAB¢PFADDR¢ := IC;
3736                    PFADDR := IC;
3737              /*COPY MULTIPLE VALUES INTO LOCAL CELLS*/
3738                    LLC1 := 4;
3739                    LIP := FPROCP!.NEXT;
3740                    WHILE LIP # NIL DO
3741                      WITH LIP! DO
3742                        BEGIN
3743                          IF KLASS = VARS THEN
3744                            IF IDTYPE # NIL THEN
3745                              IF (VKIND = ACTUAL)&(IDTYPE!.SIZE # 1) THEN
3746                                BEGIN
3747                                  GEN2(50/*LDA*/,0,VADDR);
3748                                  GEN2(54/*LOD*/,0,LLC1);
3749                                  GEN1(40/*MOV*/,IDTYPE!.SIZE)
3750                                END;
3751                          LIP := LIP!.NEXT; LLC1 := LLC1 + 1
3752                        END;
3753                  END
3754              ELSE EPMAIN := IC;
3755            CSTPTRIX := 0; FRETNIX := 0; LABPTRIX := 0; CIX := -1;
3756            GEN1(32/*ENT*/,0); LCMAX := LC;
3757            IF FPROCP = MAINP THEN
3758              BEGIN GEN2(51/*LDC*/,1,0); GEN1(43/*SRO*/,4);
3759                GEN2(51/*LDC*/,1,0); GEN1(43/*SRO*/,4+BUFFSIZE)
3760              END;
3761            IF FSTINITP!.NEXTINIT # NIL THEN
```

```
3762              BEGIN LINITP := FSTINITP!.NEXTINIT;
3763                WHILE LINITP # NIL DO
3764                  WITH LINITP!. GDESC DO
3765                    BEGIN TYPTR := TYPOINTER; KIND := CST; CVAL := INITVAL;
3766                      IF TYPTR!.FORM <= SETS THEN
3767                        BEGIN I := REPEATF;
3768                          WHILE I > 0 DO
3769                            BEGIN I := I - 1; LOAD;
3770                              IF LEVEL = 1 THEN GEN1(43/*SRO*/,INITADDR)
3771                              ELSE GEN2(56/*STR*/,0,INITADDR)
3772                            END;
3773                        END
3774                      ELSE
3775                        BEGIN
3776                          IF LEVEL = 1 THEN GEN1(39/*LDO*/,INITADDR)
3777                          ELSE GEN2(54/*LCD*/,0,INITADDR);
3778                          LOADADDRESS; GEN1(40/*MOV*/,TYPTR!.SIZE)
3779                        END;
3780                      LINITP := LINITP!.NEXTINIT
3781                    END;
3782            END /*FSTINITP # NIL*/ ;
3783          LOOP
3784            REPEAT STATEMENT(FSYS OR STATDELIMS)
3785            UNTIL ~(SY IN STATMENTSYS);
3786          EXIT IF SY # SEMICOLON;
3787            INSYMBOL
3788          END;
3789          IF SY = ENDSY THEN ENDBLOCK(FBTYP) ELSE ERROR(600);
3790          LLP := FSTLABP; /*TEST FOR UNDEFINED LABELS*/
3791          WHILE LLP # FLABP DO
3792            WITH LLP! DO
3793              BEGIN
3794                IF ~DEFINED THEN
3795                  BEGIN ERROR(421);
3796                    WRITE(EOL,' LABEL ',LABNAME,' UNDEFINED',EOL)
3797                  END;
3798                LLP := NEXTLAB
3799              END;
3800          IF FPROCP # MAINP THEN
3801            BEGIN
3802              IF FBTYP = FUNCBLK THEN
3803                FOR IX := 1 TO FRETNIX DO INSERT(FUNCRETN@IX!,IC);
3804              IF FPROCP!.IDTYPE = NIL THEN GEN1(42/*RET*/,ORD('P'))
3805              ELSE GEN1(42/*RET*/,ORD('F'));
3806              INSERT(0,LCMAX-1);
3807              IF PRSYMB OR PRCODE THEN WRITEOUT
3808            END
3809          ELSE
3810            BEGIN GEN1(42/*RET*/,ORD('P')); INSERT(0,LCMAX-1);
3811              IF PRSYMB OR PRCODE THEN
3812                BEGIN WRITEOUT; WRITE(EOL) /*SIMULATES EOR*/ END;
3813              CIX := -1; GEN1(32/*ENT*/,0); IC := 0;
3814              /*GENERATE CALL OF MAIN PROGRAM; NOTE THAT THIS CALL MUST BE LOAD
3815                AT ABSOLUTE ADDRESS ZERO*/
3816              GEN1(41/*MST*/,0); GEN2(46/*CUP*/,0,EPMAIN); GEN0(29/*STP*/);
3817              FOR I := 3 TO JMPIX DO GEN2(57/*UJP*/,0,JMPTAB@I!);
3818              GEN1(42/*RET*/,ORD(' '));
```

```
3819                IF PRSYMB OR PRCODE THEN
3820                  BEGIN WRITEOUT; WRITE(EOL) /*SIMULATES EOR*/ END
3821              END;
3822            FSTLABP := FLABP; BOD := FALSE
3823          END /*BODY*/ ;
3824
3825        BEGIN /*BLOCK*/
3826          MARK(HEAPMARK);
3827          NEW(FSTINITP); FSTINITP^.NEXTINIT := NIL; LSTINITP := FSTINITP;
3828          FLABP := FSTLABP;  BOD := FALSE;
3829          REPEAT
3830            IF SY = CONSTSY THEN
3831              BEGIN INSYMBOL; CONSTANTDECLARATION END;
3832            IF SY = TYPESY THEN
3833              BEGIN INSYMBOL; TYPEDECLARATION END;
3834            IF SY = DECLARESY THEN
3835              BEGIN INSYMBOL; DECLARATION END;
3836            WHILE (SY = IDENT) AND NOT BOD DO
3837              BEGIN LLABEL := ID; INSYMBOL;
3838                IF SY = COLON THEN
3839                  BEGIN INSYMBOL;
3840                    IF (SY IN [PROCEDURESY,ENTRYSY]) THEN
3841                      BEGIN ID := LLABEL; PROCEDUREDEFINE(FSYS) END
3842                    ELSE BEGIN STATLAB := TRUE; BOD := TRUE END;
3843                  END
3844                ELSE
3845                  BEGIN BOD := TRUE; LASTSY := SY; SY := IDENT END;
3846              END
3847            UNTIL NOT(SY IN DECLBEGSYS) OR BOD;
3848            REPEAT BODY(FSYS OR [CASESY]);
3849              IF FSY = ENDPROG THEN
3850                IF SY = SEMICOLON THEN INSYMBOL;
3851              IF SY # FSY THEN ERRSKIP(10,FSYS OR [FSY])
3852            UNTIL (SY = FSY) OR (SY IN BLOCKBEGSYS);
3853            RELEASE(HEAPMARK); /*DELETE LOCAL ENTRIES IN THE RUNTIME **HEAP***/
3854        END /*BLOCK*/ ;
3855
3856              /*BELOW ARE THE PROCEDURES WHICH SET UP
3857                STANDARD IDENTIFIER AND STRUCTURE CELLS,
3858                AND WHICH ALSO INITIALIZE TABLES*/
3859
3860        PROCEDURE STANDARDNAMES;
3861        BEGIN
3862          NA[ 1] := 'FALSE     '; NA[ 2] := 'TRUE      '; NA[ 3] := 'SYSIN     ';
3863          NA[ 4] := 'SYSPRINT  '; NA[ 5] := 'PACK      '; NA[ 6] := 'UNPACK    ';
3864          NA[ 7] := 'MARK      '; NA[ 8] := 'RELEASE   '; NA[ 9] := 'ABS       ';
3865          NA[10] := 'TRUNC     '; NA[11] := 'ORD       '; NA[12] := 'CHR       ';
3866          NA[13] := 'PRED      '; NA[14] := 'SUCC      '; NA[15] := 'SIGN      ';
3867          NA[16] := 'MIN       '; NA[17] := 'MAX       '; NA[18] := 'SUM       ';
3868          NA[19] := 'SIN       '; NA[20] := 'COS       '; NA[21] := 'EXP       ';
3869          NA[22] := 'SQRT      '; NA[23] := 'LOG       '; NA[24] := 'TIME      ';
3870        END /*STANDARDNAMES*/ ;
3871
3872        PROCEDURE ENTERSTDTYPES;
3873        BEGIN                                                  /*TYPE UNDERLIEING:*/
3874
3875          NEW(FIXPTR,SCALAR,STANDARD);                           /*FIXED*/
```

```
3876          FIXPTR!.SIZE := 1;
3877          NEW(FLTPTR,SCALAR,STANDARD);                              /*FLOAT*/
3878          FLTPTR!.SIZE := 1;
3879          NEW(CHARPTR,SCALAR,STANDARD);                             /*CHAR*/
3880          CHARPTR!.SIZE := 1;
3881          NEW(BOOLPTR,SCALAR,DECLARED);                             /*BOOLEAN*/
3882          BOOLPTR!.SIZE := 1;
3883          NEW(NILPTR,POINTER);                                      /*NIL*/
3884          WITH NILPTR! DO
3885            BEGIN ELTYPE := NIL; SIZE := 1 END;
3886          NEW(TEXTPTR,FILES);                                       /*TEXT*/
3887          WITH TEXTPTR! DO
3888            BEGIN FILUNIT := 0; LCFADDR := 0; SIZE := BUFFSIZE END
3889        END /*ENTERSTDTYPES*/ ;
3890
3891        PROCEDURE ENTERSTDNAMES;
3892          VAR IP,IP1: IDP; I: INTEGER;
3893        BEGIN                                                       /*NAME:*/
3894
3895          NEW(IP,TYPES,FALSE);                                      /*BOOLEAN*/
3896          WITH IP! DO
3897            BEGIN NAME := 'BOOLEAN   '; IDTYPE := BOOLPTR END;
3898          ENTERID(IP);
3899          NEW(IP,TYPES,FALSE);                                      /*TEXT*/
3900          WITH IP! DO
3901            BEGIN NAME := 'TEXT      '; IDTYPE := TEXTPTR END;
3902          ENTERID(IP);
3903          NEW(IP,KONST);                                            /*NIL*/
3904          WITH IP! DO
3905            BEGIN NAME := 'NULL      '; IDTYPE := NILPTR;
3906              NEXT := NIL; VALUES.IVAL := 0
3907            END;
3908          ENTERID(IP);
3909          NEW(IP,KONST);                                            /*EOL*/
3910          WITH IP! DO
3911            BEGIN NAME := 'EOL       '; IDTYPE := CHARPTR;
3912              NEXT := NIL; VALUES.IVAL := 0
3913            END;
3914          ENTERID(IP);
3915          NEW(IP,KONST);                                            /*ALFALENG*/
3916          WITH IP! DO
3917            BEGIN NAME := 'ALFALENG  '; IDTYPE := FIXPTR;
3918              NEXT := NIL; VALUES.IVAL := ALFALENG
3919            END;
3920          ENTERID(IP);
3921          IP1 := NIL;
3922          FOR I := 1 TO 2 DO
3923            BEGIN NEW(IP,KONST);                                    /*FALSE,TRUE*/
3924              WITH IP! DO
3925                BEGIN NAME := NA[I]; IDTYPE := BOOLPTR;
3926                  NEXT := IP1; VALUES.IVAL := I - 1
3927                END;
3928              ENTERID(IP); IP1 := IP
3929            END;
3930          BOOLPTR!.FCONST := IP;
3931          FOR I := 3 TO 4 DO
3932            BEGIN NEW(IP,VARS);                                     /*INPUT,OUTPUT*/
```

```
3933              WITH IP! DO
3934                BEGIN NAME := NA$I; IDTYPE := TEXTPTR;
3935                  VKIND := ACTUAL; NEXT := NIL; VLEV := 0; VADDR := I + 1
3936              END;
3937              ENTERID(IP)
3938            END;
3939          FOR I := 5 TO 8 DO
3940            BEGIN NEW(IP,PROC,STANDARD);                          /*PACK,UNPACK*/
3941              WITH IP! DO                                         /*MARK,RELEASE*/
3942                BEGIN NAME := NA$I; IDTYPE := NIL;
3943                  NEXT := NIL; KEY := I - 4;
3944                END;
3945              ENTERID(IP)
3946            END;
3947          FOR I := 9 TO 18 DO
3948            BEGIN NEW(IP,FUNC,STANDARD);                          /*ABS,TRUNC,ORD*/
3949              WITH IP! DO                                         /*CHAR,PRED*/
3950                BEGIN NAME := NA$I; IDTYPE := NIL;                /*SUCC,EOF,MIN*/
3951                  NEXT := NIL; KEY := I - 8;                      /*MAX,SUM*/
3952              END;
3953              ENTERID(IP)
3954            END;
3955          NEW(IP,VARS);                       /*PARAMETER OF PREDECLARED FUNCTIONS*/
3956          WITH IP! DO
3957            BEGIN NAME := '          '; IDTYPE := FLTPTR;
3958              VKIND := ACTUAL; NEXT := NIL; VLEV := 1; VADDR := 0
3959            END;
3960          FOR I := 19 TO 24 DO
3961            BEGIN NEW(IP1,FUNC,DECLARED,ACTUAL);                  /*SIN,COS,EXP*/
3962              WITH IP1! DO                                       /*SQRT,LOG,TIME*/
3963                BEGIN NAME := NA$I; IDTYPE := FLTPTR; NEXT := IP;
3964                  FORWDECL := FALSE; EXTERN := TRUE; PFLEV := 0; PFADDR := I - 4
3965                END;
3966              ENTERID(IP1)
3967            END;
3968        END /*ENTERSTDNAMES*/ ;
3969
3970        PROCEDURE ENTERUNDECL;
3971        BEGIN
3972          NEW(UTYPPTR,TYPES,FALSE);
3973          WITH UTYPPTR! DO
3974            BEGIN NAME := '          '; IDTYPE := NIL END;
3975          NEW(UCSTPTR,KONST);
3976          WITH UCSTPTR! DO
3977            BEGIN NAME := '          '; IDTYPE := NIL; NEXT := NIL;
3978              VALUES.IVAL := 0
3979            END;
3980          NEW(UVARPTR,VARS);
3981          WITH UVARPTR! DO
3982            BEGIN NAME := '          '; IDTYPE := NIL; VKIND := ACTUAL;
3983              NEXT := NIL; VLEV := 0; VADDR := 0
3984            END;
3985          NEW(UFLDPTR,FIELD);
3986          WITH UFLDPTR! DO
3987            BEGIN NAME := '          '; IDTYPE := NIL; NEXT := NIL; FLDADDR := 0
3988            END;
3989          NEW(UPRCPTR,PROC,DECLARED,ACTUAL);
```

```
3990          WITH UPRCPTR! DO
3991            BEGIN NAME := '          '; IDTYPE := NIL; FORWDECL := FALSE;
3992              NEXT := NIL; EXTERN := FALSE; PFLEV := 0; PFADDR := 3
3993            END;
3994          NEW(UFIDPTR,FUNC,DECLARED,ACTUAL);
3995          WITH UFIDPTR! DO
3996            BEGIN NAME := '          '; IDTYPE := NIL; NEXT := NIL;
3997              FORWDECL := FALSE; EXTERN := FALSE; PFLEV := 0; PFADDR := 3
3998            END
3999        END /*ENTERUNDECL*/ ;
4000
4001        PROCEDURE INITSCALARS;
4002        BEGIN FWPTR := NIL; FSTLABP := NIL;
4003          ATTRLIST := FALSE; LIST := TRUE; PRSYMB := TRUE; PRCODE := TRUE;
4004          ERRORS := FALSE; TRACE := FALSE; LOCOPT := FALSE;
4005          PRTIME := FALSE; SUBSCRIPT := FALSE; XREF := FALSE;
4006          STATLAB := FALSE; NXTSTLAB := FALSE;
4007          BOELIM := FALSE; EOELIM := FALSE;
4008          DP := FALSE; BOD := FALSE; PRTERR := TRUE;
4009          FATALERR := FALSE; ERRTOT := 0; ERRINX := 0;
4010          JMPIX := 2; KK := ALFALENG; LASTSY := OTHERSY;
4011          LC := 4+BUFFSIZE+BUFFSIZE; GLC := LC;   /*INPUT, OUTPUT BUFFERS*/
4012          IC := JMPMAX + 1; GIC := IC; /*PROGRAM LOADED AT JMPMAX+1*/
4013          CH := ' '; CHCNT := 0; LINENO := 1; FILENO := 1C;
4014          BIX := 0; BLMAX := 0; ENDLEV := 0; MBLOCKTYPE := PROCBLK;
4015        END /*INITSCALARS*/ ;
4016
4017        PROCEDURE INITSETS;
4018        BEGIN
4019          DIGITS := ['0','1','2','3','4','5','6','7','8','9'];
4020          LETTERS := ['A','B','C','D','E','F','G','H','I','J','K','L','M',
4021                      'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'];
4022          SVATTRSYS :=[BINARYSY,DECIMALSY,FIXEDSY,FLOATSY,BASEDSY,INITIALSY,
4023                       LIKESY,EXTERNSY,AUTOSY,STATICSY];
4024          VATTRSYS := [POINTERSY,LABELSY,CHARSY,BITSY] OR SVATTRSYS;
4025          ATTRSYS := [FILESY,ENTRYSY] OR VATTRSYS;
4026          CONSTBEGSYS := [ADDOP,FIXCONST,FLTCONST,STRINGCONST,IDENT];
4027          SIMPTYPEBEGSYS := [LBRACK] OR CONSTBEGSYS OR SVATTRSYS;
4028          TYPEBEGSYS :=[RECORDSY,SETSY,FILESY,LPARENT] OR SIMPTYPEBEGSYS OR VATTRSYS;
4029          SELECTSYS := [ARROW,PERIOD,LPARENT];
4030          FACBEGSYS := [FIXCONST,FLTCONST,STRINGCONST,IDENT,LPARENT,LBRACK,NOTSY];
4031          STATBEGSYS := [BEGINSY,GOTOSY,IFSY,REPEATSY,LOOPSY,CALLSY,CASESY,
4032                         DOSY,FORSY,GETSY,PUTSY,READSY,WRITESY,CHECKSY,ENTRYSY,
4033                         RETURNSY,ALLOCATESY,ONSY,FREESY,FORMATSY,STOPSY];
4034          STATMENTSYS := [IDENT] OR STATBEGSYS;
4035          DECLBEGSYS := [CONSTSY,TYPESY,DECLARESY];
4036          BLOCKBEGSYS := DECLBEGSYS OR STATMENTSYS;
4037          STATDELIMS := [SEMICOLON,ENDSY];
4038          LISTDELIMS := [COMMA,RPARENT];
4039          SEMICOMMA := [SEMICOLON,COMMA];
4040        END /*INITSETS*/ ;
4041
4042        PROCEDURE INITTABLES;
4043          PROCEDURE RESWORDS;
4044          BEGIN
4045            RW[ 1] := 'BY        '; RW[ 2] := 'GO        '; RW[ 3] := 'DO        ';
4046            RW[ 4] := 'IF        '; RW[ 5] := 'IN        '; RW[ 6] := 'NO        ';
```

```
4047            RW¢ 7| := '0F        '; RW¢ 8| := 'ON        '; RW¢ 9| := 'OR        ';
4048            RW¢10| := 'TO        '; RW¢11| := 'AND       '; RW¢12| := 'BIT       ';
4049            RW¢13| := 'DCL       '; RW¢14| := 'DIV       '; RW¢15| := 'END       ';
4050            RW¢16| := 'BIN       '; RW¢17| := 'GET       '; RW¢18| := 'MOD       ';
4051            RW¢19| := 'NOT       '; RW¢20| := 'PUT       '; RW¢21| := 'SET       ';
4052            RW¢22| := 'CALL      '; RW¢23| := 'CASE      '; RW¢24| := 'CHAR      ';
4053            RW¢25| := 'EDIT      '; RW¢26| := 'ELSE      '; RW¢27| := 'EXIT      ';
4054            RW¢28| := 'FILE      '; RW¢29| := 'FREE      '; RW¢30| := 'FROM      ';
4055            RW¢31| := 'GOTO      '; RW¢32| := 'INTO      '; RW¢33| := 'LIST      ';
4056            RW¢34| := 'LOOP      '; RW¢35| := 'PAGE      '; RW¢36| := 'PROC      ';
4057            RW¢37| := 'READ      '; RW¢38| := 'LIKE      '; RW¢39| := 'SKIP      ';
4058            RW¢40| := 'STOP      '; RW¢41| := 'THEN      '; RW¢42| := 'TYPE      ';
4059            RW¢43| := 'INIT      '; RW¢44| := 'LINE      '; RW¢45| := 'BASED     ';
4060            RW¢46| := 'BEGIN     '; RW¢47| := 'CHECK     '; RW¢48| := 'CONST     ';
4061            RW¢49| := 'ENTRY     '; RW¢50| := 'FIXED     '; RW¢51| := 'FLOAT     ';
4062            RW¢52| := 'LABEL     '; RW¢53| := 'UNTIL     '; RW¢54| := 'WHILE     ';
4063            RW¢55| := 'WRITE     '; RW¢56| := 'BINARY    '; RW¢57| := 'FORMAT    ';
4064            RW¢58| := 'REPEAT    '; RW¢59| := 'RETURN    '; RW¢60| := 'STATIC    ';
4065            RW¢61| := 'OPTIONS   '; RW¢62| := 'POINTER   '; RW¢63| := 'DECIMAL   ';
4066            RW¢64| := 'DECLARE   '; RW¢65| := 'INITIAL   '; RW¢66| := 'RETURNS   ';
4067            RW¢67| := 'ALLOCATE  '; RW¢68| := 'EXTERNAL  '; RW¢69| := 'AUTOMATIC ';
4068            RW¢70| := 'CHARACTER '; RW¢71| := 'PROCEDURE ';
4069            PSSY¢1| := 'EQ        ';PSSY¢2| := 'NE        ';PSSY¢3| := 'GT        ';
4070            PSSY¢4| := 'GE        ';PSSY¢5| := 'LT        ';PSSY¢6| := 'LE        ';
4071            FRW¢1| :=  1; FRW¢2| :=  1; FRW¢3| :=  11; FRW¢4| := 22; FRW¢5| := 45;
4072            FRW¢6| := 56; FRW¢7| := 61; FRW¢8| := 67; FRW¢9| := 69; FRW¢10| := 72;
4073            FRW¢11| := 72;
4074          END /*RESWORDS*/ ;
4075
4076          PROCEDURE SYMBOLS;
4077          BEGIN
4078            RSY¢ 1| := BYSY; RSY¢ 2| := GOSY; RSY¢ 3| := DOSY;
4079            RSY¢ 4| := IFSY; RSY¢ 5| := RELOP ; RSY¢ 6| := NOSY  ;
4080            RSY¢ 7| := OFSY; RSY¢ 8| := ONSY; RSY¢ 9| := ADDOP;
4081            RSY¢10| := TOSY; RSY¢11| := MULOP; RSY¢12| := BITSY;
4082            RSY¢13| := DECLARESY; RSY¢14| := MULOP; RSY¢15| := ENDSY;
4083            RSY¢16| := BINARYSY; RSY¢17| := GETSY; RSY¢18| := MULOP;
4084            RSY¢19| := NOTSY; RSY¢20| := PUTSY; RSY¢21| := SETSY;
4085            RSY¢22| := CALLSY; RSY¢23| := CASESY; RSY¢24| := CHARSY;
4086            RSY¢25| := EDITSY; RSY¢26| := ELSESY; RSY¢27| := EXITSY;
4087            RSY¢28| := FILESY; RSY¢29| := FREESY; RSY¢30| := FROMSY;
4088            RSY¢31| := GOTOSY; RSY¢32| := INTOSY; RSY¢33| := LISTSY;
4089            RSY¢34| := LOOPSY; RSY¢35| := PAGESY; RSY¢36| := PROCEDURESY;
4090            RSY¢37| := READSY; RSY¢38| := LIKESY; RSY¢39| := SKIPSY;
4091            RSY¢40| := STOPSY; RSY¢41| := THENSY; RSY¢42| := TYPESY;
4092            RSY¢43| := INITIALSY; RSY¢44| := LINESY; RSY¢45| := BASEDSY;
4093            RSY¢46| := BEGINSY; RSY¢47| := CHECKSY; RSY¢48| := CONSTSY;
4094            RSY¢49| := ENTRYSY; RSY¢50| := FIXEDSY; RSY¢51| := FLOATSY;
4095            RSY¢52| := LABELSY; RSY¢53| := UNTILSY; RSY¢54| := WHILESY;
4096            RSY¢55| := WRITESY; RSY¢56| := BINARYSY; RSY¢57| := FORMATSY;
4097            RSY¢58| := REPEATSY; RSY¢59| := RETURNSY; RSY¢60| := STATICSY;
4098            RSY¢61| := OPTIONSY; RSY¢62| := POINTERSY; RSY¢63| := DECIMALSY;
4099            RSY¢64| := DECLARESY; RSY¢65| := INITIALSY; RSY¢66| := RETURNSSY;
4100            RSY¢67| := ALLOCATESY; RSY¢68| := EXTERNSY; RSY¢69| := AUTOSY;
4101            RSY¢70| := CHARSY; RSY¢71| := PROCEDURESY;
4102            SSY¢'+'| := ADDOP; SSY¢'-'| := ADDOP; SSY¢'*'| := MULOP;
4103            SSY¢'/'| := MULOP; SSY¢'('| := LPARENT; SSY¢')'| := RPARENT;
```

```
4104            SSY¢','¦ := COMMA; SSY¢'¢'¦ := LBRACK; SSY¢'.'¦ := PERIOD;
4105            SSY¢'¦'¦ := CATSY; SSY¢':'¦ := COLON;
4106            SSY¢'#'¦ := RELOP; SSY¢'='¦ := BECOMES;
4107            SSY¢'&'¦ := MULOP; SSY¢'!'¦ := ARROW;
4108            SSY¢'<'¦ := RELOP; SSY¢'>'¦ := RELOP;
4109            SSY¢'¬'¦ := NOTSY; SSY¢';'¦ := SEMICOLON;
4110          END /*SYMBOLS*/ ;
4111
4112          PROCEDURE OPERATORS;
4113            VAR I: INTEGER; CH: CHAR;
4114          BEGIN
4115            FOR I := 1 TO 71 /*NR OF RES WORDS*/ DO ROP¢I¦ := NOOP;
4116            ROP¢5¦ := INOP; ROP¢14¦ := ILIV; ROP¢18¦ := IMOD;
4117            ROP¢9¦ := OROP; ROP¢11¦ := ANDOP;
4118            FOR CH := '¢' TO ';' DO SOP¢CH¦ := NOOP; SOP¢'/'¦ := RDIV;
4119            SOP¢'+'¦ := PLUS; SOP¢'-'¦ := MINUS; SOP¢'*'¦ := MUL;
4120            SOP¢'='¦ := EQOP; SOP¢'#'¦ := NEOP;
4121            SOP¢'<'¦ := LTOP; SOP¢'>'¦ := GTOP; SOP¢'&'¦ := ANDOP;
4122            PSOP¢1¦ := EQOP; PSOP¢2¦ := NEOP; PSOP¢3¦ := GTOP;
4123            PSOP¢4¦ := GEOP; PSOP¢5¦ := LTOP; PSOP¢6¦ := LEOP;
4124          END /*OPERATORS*/ ;
4125
4126          PROCEDURE PROCMNEMONICS;
4127          BEGIN
4128            SNA¢ 1¦ := 'SUM'; SNA¢ 2¦ := 'SGN'; SNA¢ 3¦ := 'RDI'; SNA¢ 4¦ := 'RDR';
4129            SNA¢ 5¦ := 'RDC'; SNA¢ 6¦ := 'WRI'; SNA¢ 7¦ := 'WRC'; SNA¢ 8¦ := 'WRR';
4130            SNA¢ 9¦ := 'WRC'; SNA¢10¦ := 'WRS'; SNA¢11¦ := 'PAK'; SNA¢12¦ := 'NEW';
4131            SNA¢13¦ := 'SAV'; SNA¢14¦ := 'RST'; SNA¢15¦ := 'SIN'; SNA¢16¦ := 'COS';
4132            SNA¢17¦ := 'EXP'; SNA¢18¦ := 'SQT'; SNA¢19¦ := 'LOG'; SNA¢20¦ := 'TIM';
4133            SNA¢21¦ := 'MIN'; SNA¢22¦ := 'MAX';
4134          END /*PROCMNEMONICS*/ ;
4135
4136          PROCEDURE INSTRMNEMONICS;
4137          BEGIN
4138            MN¢ 0¦ := 'ABI'; MN¢ 1¦ := 'ABR'; MN¢ 2¦ := 'ADI'; MN¢ 3¦ := 'ADR';
4139            MN¢ 4¦ := 'AND'; MN¢ 5¦ := 'DIF'; MN¢ 6¦ := 'DVI'; MN¢ 7¦ := 'DVR';
4140            MN¢ 8¦ := 'EOF'; MN¢ 9¦ := 'FLO'; MN¢10¦ := 'FLT'; MN¢11¦ := 'INN';
4141            MN¢12¦ := 'INT'; MN¢13¦ := 'IOR'; MN¢14¦ := 'MOD'; MN¢15¦ := 'MPI';
4142            MN¢16¦ := 'MPR'; MN¢17¦ := 'NGI'; MN¢18¦ := 'NGR'; MN¢19¦ := 'NOT';
4143            MN¢20¦ := 'ODD'; MN¢21¦ := 'SBI'; MN¢22¦ := 'SBR'; MN¢23¦ := 'SGS';
4144            MN¢24¦ := 'SQI'; MN¢25¦ := 'SQR'; MN¢26¦ := 'STO'; MN¢27¦ := 'TRC';
4145            MN¢28¦ := 'UNI'; MN¢29¦ := 'STP'; MN¢30¦ := 'CSP'; MN¢31¦ := 'DEC';
4146            MN¢32¦ := 'ENT'; MN¢33¦ := 'FJP'; MN¢34¦ := 'INC'; MN¢35¦ := 'IND';
4147            MN¢36¦ := 'IXA'; MN¢37¦ := 'LAO'; MN¢38¦ := 'LCA'; MN¢39¦ := 'LDO';
4148            MN¢40¦ := 'MOV'; MN¢41¦ := 'MST'; MN¢42¦ := 'RET'; MN¢43¦ := 'SRO';
4149            MN¢44¦ := 'XJP'; MN¢45¦ := 'CHK'; MN¢46¦ := 'CUP'; MN¢47¦ := 'EQU';
4150            MN¢48¦ := 'GEQ'; MN¢49¦ := 'GRT'; MN¢50¦ := 'LDA'; MN¢51¦ := 'LDC';
4151            MN¢52¦ := 'LEQ'; MN¢53¦ := 'LES'; MN¢54¦ := 'LOD'; MN¢55¦ := 'NEQ';
4152            MN¢56¦ := 'STR'; MN¢57¦ := 'UJP';
4153          END /*INSTRMNEMONICS*/ ;
4154
4155      /*  PROCEDURE MESSAGES;                    (NOT YET IMPLEMENTED)
4156          BEGIN
4157            MS¢ 10¦ 'SYNTAX ERROR. ILLEGAL CHARACTERS SKIPPED'
4158            MS¢ 11¦ 'MISSING RIGHT PARENTHESIS'
4159            MS¢ 12¦ 'MISSING LEFT PARENTHESIS'
4160            MS¢ 13¦ 'UNPAIRED RIGHT PARENTHESIS'
```

```
4161            MSG¢ 14! 'MISSING COMMA OR SEMICOLON'
4162            MSG¢ 15! 'SYNTAX ERROR, THEN EXPECTED'
4163            MSG¢ 16! 'SYNTAX ERROR, OF EXPECTED'
4164            MSG¢ 17! 'SYNTAX ERROR, UNTIL EXPECTED'
4165            MSG¢ 18! 'SYNTAX ERROR, DO EXPECTED'
4166            MSG¢ 19! 'SYNTAX ERROR, IF EXPECTED'
4167            MSG¢ 20! 'SYNTAX ERROR, EXIT EXPECTED'
4168            MSG¢ 21! 'SYNTAX ERROR, IDENTIFIER EXPECTED'
4169            MSG¢ 22! 'SYNTAX ERROR, [ EXPECTED'
4170            MSG¢ 23! 'SYNTAX ERROR, ¢ EXPECTED'
4171            MSG¢ 24! 'SYNTAX ERROR, ) EXPECTED'
4172            MSG¢ 25! 'SYNTAX ERROR, ( EXPECTED'
4173            MSG¢ 26! 'SYNTAX ERROR, COLON EXPECTED'
4174            MSG¢ 27! 'SYNTAX ERROR, SEMICOLON EXPECTED'
4175            MSG¢ 28! 'SYNTAX ERROR, ( OR SEMICOLON EXPECTED'
4176            MSG¢ 29! 'MISSING COMMA'
4177            MSG¢100! 'NUMBER EXPECTED IN CONSTANT DECLARATION'
4178            MSG¢101! 'SYNTAX ERROR IN DECLARATION PART'
4179            MSG¢102! 'IDENTIFIER EXPECTED IN DECLARATION PART'
4180            MSG¢103! 'DATA ATTRIBUTE EXPECTED'
4181            MSG¢104! 'KEYWORD SET, ARRAY, FILE EXPECTED'
4182            MSG¢105! 'MISSING = AFTER CONSTANT OR TYPE NAME'
4183            MSG¢106! 'UNSATISFIED FORWARD REFERENCE OF TYPE'
4184            MSG¢107! 'DIMENSIONS ILLEGAL FOR PROCEDURE DECLARATION'
4185            MSG¢108! 'INITIAL VALUES ILLEGAL FOR BASED VARIABLES'
4186            MSG¢120! 'NON-OCTAL DIGITS IN OCTAL CONSTANT'
4187            MSG¢121! 'INTEGER CONSTANT EXCEEDS RANGE'
4188            MSG¢122! 'ERROR IN REAL CONSTANT, DIGIT EXPECTED'
4189            MSG¢123! 'ERROR IN CONSTANT'
4190            MSG¢124! 'SIGN NOT ALLOWED FOR OTHER THAN REAL, INTEGER'
4191            MSG¢125! 'CHARACTER STRING MUST BE ON ONE LINE'
4192            MSG¢126! 'ERROR IN REPETITION COUNT'
4193            MSG¢130! 'IDENTIFIER IS BEING REDECLARED'
4194            MSG¢131! 'IDENTIFIER IS NOT OF APPROPRIATE CLASS'
4195            MSG¢132! 'IDENTIFIER HAS NOT BEEN DECLARED'
4196            MSG¢133! 'SYNTAX ERROR - IMPROPER USE OF VARIABLE'
4197            MSG¢134! 'CONTROL VARIABLE MUST NOT BE FORMAL'
4198            MSG¢150! 'IMPROPER TYPE IN BOUND PAIR LIST OF ARRAY'
4199            MSG¢151! 'UPPER, LOWER BOUNDS ARE OF INCOMPATIBLE TYPES'
4200            MSG¢152! 'UPPER BOUND MUST BE GREATER THAN LOWER'
4201            MSG¢153! 'INVALID SUBSCRIPT, MUST NOT BE REAL, INTEGER'
4202            MSG¢154! 'INVALID SUBSCRIPT, MUST BE SCALAR, SUBRANGE'
4203            MSG¢155! 'SUBSCRIPT VARIABLE MUST BE FORMAL'
4204            MSG¢156! 'SUBSCRIPT VARIABLE MUST BE INTEGER'
4205            MSG¢157! 'INVALID SUBSCRIPT, INCOMPATIBLE WITH DECLARATION'
4206            MSG¢158! 'VARIABLE IS NOT OF TYPE ARRAY'
4207            MSG¢159! 'ONLY ONE DIMENSION ALLOWED HERE'
4208            MSG¢170! 'BASE TYPE MUST BE SCALAR, SUBRANGE'
4209            MSG¢171! 'BASE TYPE MUST NOT BE REAL'
4210            MSG¢172! 'ILLEGAL SET ELEMENT TYPE'
4211            MSG¢173! 'SET ELEMENT TYPES INCOMPATIBLE'
4212            MSG¢174! 'ILLEGAL EXPRESSION TYPE - SHOULD BE SET'
4213            MSG¢180! 'POINTER TO FILE NOT ALLOWED'
4214            MSG¢181! 'FILENAME OR POINTER VARIABLE EXPECTED'
4215            MSG¢182! 'ASSIGNMENT TO FILES ILLEGAL'
4216            MSG¢183! 'MISSING POINTER NAME'
4217            MSG¢200! 'FIELD LEVEL MISSING OR INVALID'
```

```
4218          MSG¢2011 'FIRST FIELD SHOULD HAVE LEVEL = 1'
4219          MSG¢2021 'VARIABLE IS NOT OF TYPE RECORD'
4220          MSG¢2031 'NO SUCH FIELD IN THIS RECORD'
4221          MSG¢2041 'WITH STATEMENT VARIABLE NOT OF TYPE RECORD'
4222          MSG¢3001 'PROCEDURE IS BEING REDECLARED'
4223          MSG¢3011 'FUNCTION RESULT TYPE NOT SCALAR, SUBRANGE, POINTER'
4224          MSG¢3021 'MISSING DECLARATION OF PROCEDURE'
4225          MSG¢3031 'FUNCTION RESULT TYPE DISAGREES WITH DECLARATION'
4226          MSG¢3041 'NOT DECLARED AS A FUNCTION'
4227          MSG¢3051 'ASSIGNMENT TO STANDARD FUNCTION ILLEGAL'
4228          MSG¢3061 'ASSIGNMENT TO FORMAL FUNCTION ILLEGAL'
4229          MSG¢3071 'ARRAY, RECORD, FILE FUNCTIONS ILLEGAL'
4230          MSG¢3201 'ILLEGAL PARAMETER TYPE'
4231          MSG¢3211 'PARAMETER TYPE NOT SCALAR, SUBRANGE, POINTER'
4232          MSG¢3221 'ERROR IN TYPE LIST - SKIPPED'
4233          MSG¢3231 'ERROR IN PARAMETER LIST'
4234          MSG¢3241 'NO OF PARAMETERS DISAGREES WITH DECLARATION'
4235          MSG¢3251 'FORMAL PARS OF PAR. PROCS NOT IMPLEMENTED'
4236          MSG¢3261 'ERROR IN TYPE OF STANDARD PROCEDURE PARAMETER'
4237          MSG¢3271 'ERROR IN TYPE OF STANDARD FUNCTION PARAMETER'
4238          MSG¢3281 'RESULT TYPE OF PAR. FUNCTION DISAGREES WITH DECL.'
4239          MSG¢3291 'ACTUAL PARAMETER MUST BE A VARIABLE'
4240          MSG¢3301 'ILLEGAL PARAMETER SUBSTITUTION'
4241          MSG¢3311 'ERROR IN TYPE OF PARAMETER VARIABLE'
4242          MSG¢4001 'RETURN STATEMENT IN BEGIN-END ILLEGAL'
4243          MSG¢4011 'STATEMENT PROCESSOR NOT YET AVAILABLE'
4244          MSG¢4021 'FIRST LINE MISSING OR INVALID'
4245          MSG¢4031 'THIS FEATURE HAS NOT BEEN IMPLEMENTED'
4246          MSG¢4041 'TRANSFER TO INNER LOOP ILLEGAL'
4247          MSG¢4201 'DUPLICATE LABEL'
4248          MSG¢4211 'UNDEFINED LABEL'
4249          MSG¢4221 'CONTROL CAN NEVER REACH THIS STATEMENT'
4250          MSG¢4231 'VARIABLE IS NOT OF TYPE LABEL'
4251          MSG¢4401 'CASE LABEL DEFINED TWICE'
4252          MSG¢4411 'LABEL TYPE INCOMPATIBLE WITH SELECTING EXPRESSION'
4253          MSG¢4421 'CASE OVERFLOW'
4254          MSG¢4601 'ONLY I/O OF REALS, INTEGERS, STRINGS IMPLEMENTED'
4255          MSG¢4611 'F-FORMAT FOR REAL ONLY'
4256          MSG¢4701 'IDENTIFIER IS NOT BASED VARIABLE'
4257          MSG¢4711 'BASED VARIABLE EXPECTED'
4258          MSG¢4721 'BASED VARIABLE,POINTER TYPES DISAGREE'
4259          MSG¢4801 'MISSING := IN ASSIGNMENT STATEMENT'
4260          MSG¢5001 'EXPRESSION NOT OF TYPE BOOLEAN'
4261          MSG¢5011 'ERROR IN FACTOR'
4262          MSG¢5021 'OPERAND MUST BE TYPE BOOLEAN'
4263          MSG¢5031 'ILLEGAL OPERAND TYPE'
4264          MSG¢5041 'OPERAND TYPE CONFLICT'
4265          MSG¢5051 'ILLEGAL EXPRESSION TYPE'
4266          MSG¢5061 'EXPRESSION TYPE CONFLICT'
4267          MSG¢5071 'EXPRESSION MUST BE A CONSTANT'
4268          MSG¢5081 'ALLOCATED VARIABLE,POINTER TYPE CONFLICT'
4269          MSG¢5201 'EQUALITY TEST  ONLY ALLOWED'
4270          MSG¢5211 'STRICT INCLUSION ILLEGAL'
4271          MSG¢5221 'FILE COMPARISON ILLEGAL'
4272          MSG¢5231 'SYNTAX ERROR, = EXPECTED'
4273          MSG¢6001 'MISSING END - EXTRA ONE INSERTED'
4274          MSG¢7001 'MORE THAN 10 ERRORS ON THIS LINE'
```

```
4275            MSG¢7011 'STRING/SET/CONSTANT TABLE OVERFLOW'
4276            MSG¢7021 'CODE OVERFLOW IN CURRENT PROCEDURE'
4277            MSG¢7031 'TOO MANY FORWARD REFERENCES OF PROCEDURES'
4278            MSG¢7041 'TOO MANY NESTED BLOCKS OR PROCEDURES'
4279            MSG¢7051 'TOO MANY FORWARD JUMPS'
4280            MSG¢8001 'INDEXED ACCESS IN STORE-INTERNAL ERROR'
4281            MSG¢8011 'STRING NOT DEFINED AS CONSTANT-INTERNAL ERROR'
4282            MSG¢8021 'INDIRECT ACCESS IN STORE-INTERNAL ERROR'
4283            MSG¢8031 'LOAD ADDRESS OF EXPRESSION-INTERNAL ERROR'
4284          END; /*MESSAGES*/
4285
4286        BEGIN /*INITTABLES*/
4287          RESWORDS; SYMBOLS; OPERATORS;
4288          INSTRMNEMONICS; PROCMNEMONICS;
4289        END /*INITTABLES*/ ;
4290
4291        PROCEDURE PL1(VAR FIP: IDP);
4292              /*PROCESS THE FIRST LINE, AND READY THE
4293                COMPILER TO PROCESS THE OUTERMOST BLOCK*/
4294          VAR LIP: IDP;
4295        BEGIN INSYMBOL;
4296          NEW(LIP,PROC,DECLARED,ACTUAL);
4297          IF SY # IDENT THEN ERROR(402);
4298          LIP!.NAME := ID; LIP!.PFLEV := 0; LIP!.NEXT := NIL;
4299          LIP!.FORWDECL := TRUE; LIP!.EXTERN := TRUE;
4300          ENTERID(LIP); FIP := LIP;
4301        END /*PL1*/ ;
4302
4303      BEGIN /*MAIN PROGRAM*/
4304        LEVEL := 0; TOP := 0; IDTREE¢01 := NIL;
4305
4306        /*ENTER STANDARD NAMES AND TYPES*/
4307        ENTERSTDTYPES;   STANDARDNAMES; ENTERSTDNAMES;   ENTERUNDECL;
4308
4309        /*INITIALIZE:*/
4310        INITSCALARS; INITSETS; INITTABLES;
4311
4312        /*COMPILE:*/
4313        PL1(MAINP);
4314        BLOCK(BLOCKBEGSYS OR STATBEGSYS-[CASESY],MBLOCKTYPE,ENDPROG,MAINP);
4315
4316        /*AFTERMATH*/
4317        IF LIST THEN
4318          IF CH # EOL THEN WRITE(EOL);
4319        ENDOFLINE;
4320        IF ATTRLIST THEN ATTRIBUTELIST(TRUE);
4321        IF FATALERR THEN WRITE(EOL,'FATAL ERROR OCCURRED':20);
4322        WRITE(EOL,EOL,'COMPILATION COMPLETE.      ':26);
4323        IF ERRTOT = 1 THEN WRITE(ERRTOT:2,' ERROR WAS':10)
4324            ELSE WRITE(ERRTOT:3,' ERRORS WERE':12);
4325        WRITE(' FOUND':6,EOL,EOL)
4326      END .
```

APPENDIX D

SYMBOLIC INSTRUCTIONS OF PL/UCT STACK CODE

Each instruction is packed into a 36 bit word.
The op-code occupies a 6 bit field, parameter P
a 6 bit field, and address Q a 24 bit field.

The instructions are grouped into three classes:

CLASS A. DATA MANIPULATION AND COMPUTATION.

op-code	mnemonic	parameters	function
arithmetic	operations		
40	ABI		absolute value of integer
41	ABR		absolute value of floating point no.
28	ADI		integer addition
29	ADR		floating-point addition
53	DVI		integer division
54	DVR		floating-point division
34	FLO		float next top of stack
33	FLT		float top of stack
49	MOD		modulus
51	MPI		integer multiplication
52	MPR		floating-point multiplication
36	NGI		integer sign inversion
37	NGR		floating-point sign inversion
30	SBI		integer subtraction
31	SBR		floating-point subtraction
38	SQI		square integer
39	SQR		square floating-point number
35	TRC		truncation

op-code	mnemonic	parameters	function
logical	operations		
43	AND		logical AND
26	CHK		check against and lower bounds
27	EOF	P Q	test on end of file
17	EQU	P	compare on equal
19	GEQ	P	greater or equal
20	GRT	P	greater than
44	IOR		inclusive OR
21	LEQ	P	less than or equal
22	LES	P	less than
18	NEQ	P	not equal
42	NOT		logical NOT
50	ODD		test on odd
set	operations		
48	INN		test set membership
46	INT		set intersection
32	SGS		generate singleton set
23	UNI		set union
data	manipulation		
57	DEC	Q	decrement address
10	INC	Q	increment address
9	IND	Q	indexed fetch
16	IXA	Q	compute indexed address
5	LAO	Q	load base-level address
56	LCA	Q	load address of constant
4	LDA	P Q	load address
7	LDC	P	load immediate
1	LDO	Q	load contents of base-level address
0	LOD	P Q	load contents of address
55	MOV	Q	move
3	SRO	Q	store at base-level address
6	STO		store (indirect
2	STR	P Q	store at address

CLASS B. FLOW OF CONTROL AND BLOCK MANAGEMENT.

op-code	mnemonic	parameters	function
12	CUP	P Q	call user procedure
13	ENT	P Q	enter block
24	FJP	Q	false jump
11	MST	P Q	mark stack
14	RET	P	return from block
58	STP		stop
23	UJP	Q	unconditional jump
25	XJP	Q	indexed jump

CLASS C. BUILTIN PROCEDURES AND FUNCTIONS

These are invoked by a single instruction CSP (call standard procedure, opcode 15), and the routine is indicated by parameter P.

p - value builtin routines	mnemonic	function
22	MAX	max value of two arguments
21	MIN	min value of two arguments
12	NEW	allocate space in the heap
11	PAK	pack one area into another
14	RST	restore heap
13	SAV	mark heap for later restoration
2	SGN	return -1,0,1 for -ve, zero, -ve value
1	SUM	return sum of an array
20	TIM	return the time as a floating point number

p - value	mnemonic	function
mathematical functions		
17	EXP	e^{arg}
16	COS	cosine
19	LOG	natural logarithm
15	SIN	sine
18	SQT	square root
input/output routines		
5	RDC	read a character
3	RDI	read integer
4	RDR	read floating-point
9	WRC	write character
6	WRI	write integer
7	WRO	write octal
8	WRR	write floating-point
10	WRS	write string

APPENDIX E

LIST OF PROCEDURES IN GENERATED SEQUENCE.

APPENDIX E

LIST OF PROCEDURES IN GENERATED SEQUENCE

procedure	no. of instructions	function
DIAGNOSTIC	12	print an error number (and message)
ENDOFLINE	168	print each line
ERROR	33	store each error for printing
NEXTCH	18	get next character
OPTIONS	70	retrieve options
INSYMBOL	793	lexical scanner
ENTERID	88	enter new identifier into symbol table
SEARCHSECTION	38	search current level for procedure name
SEARCHID	131	search symbol table for identifier
GETBOUNDS	43	retrieve the bounds of an array
MARKSTP	51	produce an attribute listing
MARKIDP	21	
MARK	20	
FOLLOWSTP	217	
FOLLOWIDP	296	
ATTRIBUTELIST	64	
BEGBLOCK	44	block housekeeping routines
ENDBLOCK	119	
SKIP	10	skip to the next relevant symbol
ERRSKIP	8	skip and store an error
CONSTANT	197	set up cells for constants
COMPTYPES	115	compare two structures for equality
STRING	21	test for character string
REPEATCOUNT	52	return a repeat count for INITIAL attribute
SIMPLETYPE	501	analyze attributes and simple types
FIELDLIST	299	scan a record declaration
TYP	522	set up STRUCTURE cells
CONSTANTDECLARATION	89	set up cells for constants
TYPEDECLARATION	158	set up cells for user-defined types
PARAMTYPELIST	299	scan type list in entry declaration
PROCEDUREDECLARATION	241	scan an entry declaration
IDLIST	68	set up IDENTIFIER cells
DECLARATION	454	handle all DECLARE statements
PARAMETERLIST	111	scan a formal parameter list
PROCEDUREDEFINE	280	handle new definitions of procedures
GEN0	35	code-generating primitives
GEN1	33	
GEN2	37	
LOAD	143	generate code for loads
STORE	69	generate code for stores
LOADADDRESS	111	generate a load-address
INSERT	8	insert an address into an instruction
GENFJP	19	generate a false-jump instruction
WRITEOUT	494	output code (for a procedure) to a file
SELECTOR	450	deal with variables in statements
LABELCONSTANT	104	analyze labels in GOTO, assignment statements

procedure	no. of instructions	function
VARIABLE	21	deal with variables, calls SELECTOR
FILENAME	60	scans a filename
PACK	110	standard procedures
UNPACK	119	
MARK	22	
RELEASE	24	
ABS	27	builtin functions
TRUNC	18	
ORD	15	
CHR	15	
PREDSUCC	18	
SGN	27	
MINMAX	46	
SUM	32	
CALLNONSTANDARD	248	handle procedure calls (internal)
CALL	140	generate internal and external calls
FACTOR	361	expression analyzer
TERM	210	
SIMPLEEXPRESSION	252	
EXPRESSION	283	
ASSIGNMENT	258	statement processors
RETURNSTATEMENT	133	
GETSTATEMENT	151	
PUTOPTIONS	163	
PUTSTATEMENT	286	
GOTOSTATEMENT	65	
COMPOUNDSTATEMENT	178	
IFSTATEMENT	66	
CASESTATEMENT	285	
REPEATSTATEMENT	38	
DOSTATEMENT	784	
LOOPSTATEMENT	97	
ALLOCATESTATEMENT	133	
ONSTATEMENT	40	
STOPSTATEMENT	9	
STATEMENT	322	analyze each statement
BODY	427	handle body part of procedures
BLOCK	125	handles each block
STANDARDNAMES	171	standard names and types
ENTERSTDTYPES	86	
ENTERSTDNAMES	412	
ENTERUNDECL	174	
INITSCALARS	82	initialised data
INITSETS	52	
RESWORDS	530	
SYMBOLS	552	
OPERATORS	158	
PROCMNEMONICS	134	
INSTRMNEMONICS	288	
INITTABLES	12	
PL1	46	processes first line
MAIN PROGRAM	105	outermost-procedure control
MASTER PROGRAM	10	calls main program

APPENDIX F

ERROR NUMBER SUMMARY.

APPENDIX F.

ERROR NUMBER SUMMARY.

GENERAL SYNTAX

(10) 'SYNTAX ERROR. ILLEGAL CHARACTERS SKIPPED'
(11) 'MISSING RIGHT PARENTHESIS'
(12) 'MISSING LEFT PARENTHESIS'
(13) 'UNPAIRED RIGHT PARENTHESIS'
(14) 'MISSING COMMA OR SEMICOLON'
(15) 'SYNTAX ERROR' THEN EXPECTED'
(16) 'SYNTAX ERROR' OF EXPECTED'
(17) 'SYNTAX ERROR' UNTIL EXPECTED'
(18) 'SYNTAX ERROR' DO EXPECTED'
(19) 'SYNTAX ERROR' IF EXPECTED'
(20) 'SYNTAX ERROR' EXIT EXPECTED'
(21) 'SYNTAX ERROR' IDENTIFIER EXPECTED'
(22) 'SYNTAX ERROR'[EXPECTED'
(23) 'SYNTAX ERROR']EXPECTED'
(24) 'SYNTAX ERROR') EXPECTED'
(25) 'SYNTAX ERROR' (EXPECTED'
(26) 'SYNTAX ERROR' COLON EXPECTED'
(27) 'SYNTAX ERROR' SEMICOLON EXPECTED'
(28) 'SYNTAX ERROR' (OR SEMICOLON EXPECTED'
(29) 'MISSING COMMA'

DECLARATION PART

(100) 'NUMBER EXPECTED IN CONSTANT DECLARATION'
(101) 'SYNTAX ERROR IN DECLARATION PART'
(102) 'IDENTIFIER EXPECTED IN DECLARATION PART'
(103) 'DATA ATTRIBUTE EXPECTED'
(104) 'KEYWORD SET, ARRAY, FILE EXPECTED'
(105) 'MISSING = AFTER CONSTANT OR TYPE NAME'
(106) 'UNSATISFIED FORWARD REFERENCE OF TYPE'
(107) 'DIMENSIONS ILLEGAL FOR PROCEDURE DECLARATION'
(108) 'INITIAL VALUES ILLEGAL FOR BASED VARIABLES'

STRINGS/CONSTANTS/NUMBERS

(120) 'NON-OCTAL DIGITS IN OCTAL CONSTANT'
(121) 'INTEGER CONSTANT EXCEEDS RANGE'
(122) 'ERROR IN REAL CONSTANT, DIGIT EXPECTED'
(123) 'ERROR IN CONSTANT'
(124) 'SIGN NOT ALLOWED FOR OTHER THAN REAL, INTEGER'
(125) 'CHARACTER STRING MUST BE ON ONE LINE'
(126) 'ERROR IN REPETITION COUNT'

SIMPLE VARIABLES

(130) 'IDENTIFIER IS BEING REDECLARED'
(131) 'IDENTIFIER IS NOT OF APPROPRIATE CLASS'
(132) 'IDENTIFIER HAS NOT BEEN DECLARED'
(133) 'SYNTAX ERROR - IMPROPER USE OF VARIABLE'
(134) 'CONTROL VARIABLE MUST NOT BE FORMAL'

```
ARRAYS

  (150)   'IMPROPER TYPE IN BOUND PAIR LIST OF ARRAY'
  (151)   'UPPER, LOWER BOUNDS ARE OF INCOMPATIBLE TYPES'
  (152)   'UPPER BOUND MUST BE GREATER THAN LOWER'
  (153)   'INVALID SUBSCRIPT. MUST NOT BE REAL, INTEGER'
  (154)   'INVALID SUBSCRIPT. MUST BE SCALAR, SUBRANGE'
  (155)   'SUBSCRIPT VARIABLE MUST BE FORMAL'
  (156)   'SUBSCRIPT VARIABLE MUST BE INTEGER'
  (157)   'INVALID SUBSCRIPT, INCOMPATIBLE WITH DECLARATION'
  (158)   'VARIABLE IS NOT OF TYPE ARRAY'
  (159)   'ONLY ONE DIMENSION ALLOWED HERE'

SETS

  (170)   'BASE TYPE MUST BE SCALAR, SUBRANGE'
  (171)   'BASE TYPE MUST NOT BE REAL'
  (172)   'ILLEGAL SET ELEMENT TYPE'
  (173)   'SET ELEMENT TYPES INCOMPATIBLE'
  (174)   'ILLEGAL EXPRESSION TYPE - SHOULD BE SET'

FILES/POINTER

  (180)   'POINTER TO FILE NOT ALLOWED'
  (181)   'POINTER VARIABLE EXPECTED'
  (182)   'ASSIGNMENT TO FILES ILLEGAL'
  (183)   'FILENAME EXPECTED'

RECORDS/FIELDLISTS

  (200)   'FIELD LEVEL MISSING OR INVALID'
  (201)   'FIRST FIELD SHOULD HAVE LEVEL = 1'
  (202)   'VARIABLE IS NOT OF TYPE RECORD'
  (203)   'NO SUCH FIELD IN THIS RECORD'

PROCEDURES/FUNCTIONS

  (300)   'PROCEDURE IS BEING REDECLARED'
  (301)   'FUNCTION RESULT TYPE NOT SCALAR, SUBRANGE, POINTER'
  (302)   'MISSING DECLARATION OF PROCEDURE'
  (303)   'FUNCTION RESULT TYPE DISAGREES WITH DECLARATION'
  (304)   'NOT DECLARED AS A FUNCTION'
  (305)   'ASSIGNMENT TO STANDARD FUNCTION ILLEGAL'
  (306)   'ASSIGNMENT TO FORMAL FUNCTION ILLEGAL'
  (307)   'ARRAY, RECORD, FILE FUNCTIONS ILLEGAL'

PARAMETERS

  (320)   'ILLEGAL PARAMETER TYPE'
  (321)   'PARAMETER TYPE NOT SCALAR, SUBRANGE, POINTER'
  (322)   'ERROR IN TYPE LIST - SKIPPED'
  (323)   'ERROR IN PARAMETER LIST'
  (324)   'NO OF PARAMETERS DISAGREES WITH DECLARATION'
  (325)   'FORMAL PARS OF PAR, PROCS NOT IMPLEMENTED'
  (326)   'ERROR IN TYPE OF STANDARD PROCEDURE PARAMETER'
```

```
PARAMETERS

  (327)   'ERROR IN TYPE OF STANDARD FUNCTION PARAMETER'
  (328)   'RESULT TYPE OF PAR. FUNCTION DISAGREES WITH DECL.'
  (329)   'ACTUAL PARAMETER MUST BE A VARIABLE'
  (330)   'ILLEGAL PARAMETER SUBSTITUTION'
  (331)   'ERROR IN TYPE OF PARAMETER VARIABLE'

STATEMENTS

  (400)   'RETURN STATEMENT IN BEGIN-END ILLEGAL'
  (401)   'STATEMENT PROCESSOR NOT YET AVAILABLE'
  (402)   'FIRST LINE MISSING OR INVALID'
  (403)   'THIS FEATURE HAS NOT BEEN IMPLEMENTED'
  (404)   'TRANSFER TO INNER LOOP ILLEGAL'

LABELS

  (420)   'DUPLICATE LABEL'
  (421)   'UNDEFINED LABEL'
  (422)   'CONTROL CAN NEVER REACH THIS STATEMENT'
  (423)   'VARIABLE IS NOT OF TYPE LABEL'

CASE

  (440)   'CASE LABEL DEFINED TWICE'
  (441)   'LABEL TYPE INCOMPATIBLE WITH SELECTING EXPRESSION'
  (442)   'CASE OVERFLOW'

INPUT/OUTPUT

  (460)   'ONLY I/O OF REALS, INTEGERS, STRINGS IMPLEMENTED'
  (461)   "F-FORMAT FOR REAL ONLY'

BASED VARIABLES

  (470)   'IDENTIFIER IS NOT BASED VARIABLE'
  (471)   'BASED VARIABLE EXPECTED'
  (472)   'BASED VARIABLE, POINTER TYPES DISAGREE'

ASSIGNMENT

  (480)   'MISSING = IN ASSIGNMENT STATEMENT'

EXPRESSIONS

  (500)   'EXPRESSION NOT OF TYPE BOOLEAN'
  (501)   'ERROR IN FACTOR'
  (502)   'OPERAND MUST BE TYPE BOOLEAN'
  (503)   'ILLEGAL OPERAND TYPE'
  (504)   'OPERAND TYPE CONFLICT'
  (505)   'ILLEGAL EXPRESSION TYPE'
  (506)   'EXPRESSION TYPE CONFLICT'
  (507)   'EXPRESSION MUST BE A CONSTANT'
  (508)   'BASED VARIABLE, POINTER TYPE CONFLICT'
```

```
RELATIONS

  (520)   'EQUALITY TEST ONLY ALLOWED'
  (521)   'STRICT INCLUSION ILLEGAL'
  (522)   'FILE COMPARISON ILLEGAL'
  (523)   'SYNTAX ERROR, = EXPECTED'

BLOCK STRUCTURE
  (600)   'MISSING END - EXTRA ONE INSERTED'

OVERFLOW

  (700)   'MORE THAN 10 ERRORS ON THIS LINE'
  (701)   'STRING/SET/CONSTANT TABLE OVERFLOW'
  (702)   'CODE OVERFLOW IN CURRENT PROCEDURE'
  (703)   'TOO MANY FORWARD REFERENCES OF PROCEDURES'
  (704)   'TOO MANY NESTED BLOCKS OR PROCEDURES'
  (705)   'TOO MANY FORWARD JUMPS'

COMPILER ERROR

  (800)   'INDEXED ACCESS IN STORE-INTERNAL ERROR'
  (801)   'STRING NOT DEFINED AS CONSTANT-INTERNAL ERROR'
  (802)   'INDIRECT ACCESS IN STORE-INTERNAL ERROR'
  (803)   'LOAD ADDRESS OF EXPRESSION-INTERNAL ERROR'
```

APPENDIX G

LIST OF RESERVED IDENTIFIERS

APPENDIX H

THE PL/UCT INTERPRETER

```
CODE*PL1.PL/UCT
      C*****************************************************************
      C                                                                *
      C                                                                *
      C        THE PL/UCT INTERPRETER                                  *
      C        ----------------------                                  *
      C                                                                *
      C                  WRITTEN BY T. S. MCDERMOTT AND C. GOLDBERG    *
      C                             DEPARTMENT OF COMPUTER SCIENCE     *
      C                             UNIVERSITY OF CAPE TOWN            *
      C                                                                *
      C                                                                *
      C*****************************************************************
      C                                                                *
      C                                                                *
      C        THE INTERPRETER COMPRISES A MAIN SECTION AND TWO MAJOR  *
      C        SUBROUTINES, ASEMBL & INTERP.                           *
      C                                                                *
      C        SUBROUTINE ASEMBL ASSEMBLES THE SYMBOLIC CODE           *
      C        (ON DISC) INTO THE INTERPRETER ARRAY KODE,              *
      C        THEN WRITES THE ARRAY TO A FILE FOR LATER USE.          *
      C                                                                *
      C        SUBROUTINE INTERP ACTS AS THE INTERPRETER, AND          *
      C        ASSUMES THE GENERATED CODE IS IN BINARY FORM            *
      C        IN THE ARRAY KODE.                                      *
      C                                                                *
      C        INTERNAL SUBROUTINES XMARK, PMD, AND ERROR, AND         *
      C        FUNCTION MARK SERVE AS ANCILLARY ROUTINES.              *
      C                                                                *
      C        EXTERNAL SUBROUTINES CSF, OPTION, TIME, AND PREAD       *
      C        ARE PART OF THE EXECUTION TIME SUPPORT SYSTEM.          *
      C        THEIR FUNCTIONS ARE:-                                   *
      C                                                                *
      C        CSF       COMMUNICATE WITH OPERATING SYSTEM FILE HANDLER.*
      C                                                                *
      C        OPTION    RETRIEVE OPTIONS FROM PROCESSOR CALL CARD.    *
      C                                                                *
      C        TIME      READ RUN ELAPSED TIME IN SECONDS.             *
      C                                                                *
      C        PREAD     USED TO READ THE USER PROGRAM FROM CARDS      *
      C                  OR FROM A DISC FILE IF THE FILE (ELEMENT) NAME*
      C                  IS MENTIONED ON THE PROCESSOR CALL CARD.      *
      C                                                                *
      C*****************************************************************
      C
      CHAPTER ONE:   MAIN SECTION
      C
      CLAUSE 1 :  DECLARATIONS
            PARAMETER KODEX = 16000,MAXSTK = 17000,
           +          INTX=12,REALX=12,SETX=168,STRGX=2004,BNDX=12,
           +          CONSTX=INTX+REALX+SETX+STRGX+BNDX+1,
           +          SX=MAXSTK+CONSTX,KODEQ=CONSTX+1,MARKQ=KODEQ+KODEX,
           +          CSTQ=MAXSTK+1,MX=SX/12+1,KX=CONSTX+KODEX+MX
            INTEGER OUTBIT,TOP,BOTTOM,BASE,OP,P,Q,BLANK,PLUS,POINT,AFIELD(6),
           +        EQUAL,PROCS,HEAP,FILE
            LOGICAL EXPT,XQT,EOF,ALFBET(26)
            DIMENSION ISTORE(SX),STORE(SX),MARKER(MX),KODE(KODEX),
```

```
57          +       KONST(7),MUTATE(58),IN(14),LAYOUT(134),LINE(132),LREAL(2)
58          +       ,KSTORE(KX),AREA(2)
59           EQUIVALENCE (ISTORE(1),STORE(1)),(KSTORE(1),ISTORE(CSTQ)),
60          +         (AREA(2),STORE(1)),(KSTORE(KODEQ),KODE(1)),
61          +         (KSTORE(MARKQ),MARKER(1)),(LETTER,ISTORE(4))
62           DATA MUTATE /
63          +      26,27, 5, 6,15, 1,12,13,58,35,36, 4, 2,16,14, 9,10,28,29,34,
64          +      25, 7, 8,38,30,31,23,37, 3,44,47,32,49,39,33,41,11,53,52,56,
65          +      24,59,50,42,40,46,48,18,19,20,54,51,21,22,57,17,43,45/
66      CONSULT USE OF THE ARRAY MUTATE AT STATEMENT 712
67          +      BLANK,PLUS,MINUS,POINT,NOUGHT,NINE/1R ,1R+,1R-,1R.,1R0,1R9/
68          +      LAYOUT(1),LREAL(1)/3H(1X,2H(E/
69          +      AFIELD/3H,A1,3H,A2,3H,A3,3H,A4,3H,A5,3H,A6/
70           DATA K,KTR,IC,ITEM,LW,LOCAL/0,1,0,2,0,4/
71           PARAMETER CHAR=0,INT=1,RE=2,SET=3,BBOL=4,AD=5,PROL=6,UNDEF=7
72          +,   LARGE=524288,PMDKNT=5
73      CLAUSE 2 :  INITIALISATION OF STORE POINTERS
74           IXI = MAXSTK + 1
75           IXR = IXI + INTX
76           IXS = IXR + REALX
77           IXST = IXS + SETX
78           IXBD = IXST + STRGX
79           IMAX = IXBD + BNDX
80      CLAUSE 3 :  MAINSTREAM MODULE
81           CALL CSF('@ASG,AX2 CODE,PL1BINARY . ')
82           CALL CSF('@USE 4,CODE,PL1BINARY . ')
83           CALL CSF('@ASG,T 3 . ')
84           CALL OPTION(ALFBET)
85           LUN = 8
86           IF(.NOT.ALFBET(1)) GO TO 10
87           CALL ASEMBL
88           WRITE(4) KSTORE
89           WRITE(5,5)
90       5   FORMAT(' COMPILER LOADED')
91           IF(ALFBET(24)) STOP
92           GO TO 20
93       10  READ(4) KSTORE
94       20  WRITE(5,30)
95       30  FORMAT(' PL/UCT 1.00-07/74')
96           CALL CSF('@FREE CODE,PL1BINARY . ')
97           CALL TIME(START)
98           CALL INTERP($200)
99           IF(ISTORE(131).EQ.1) ALFBET(20) = .TRUE.
100          CALL TIME(T)
101          T = T - START
102          WRITE(5,40) T
103      40  FORMAT(' COMPILATION TIME ',F7.2,' SECONDS')
104          ENDFILE 3
105          REWIND 3
106          IF(ISTORE(427).EQ.0) GO TO 52
107          WRITE(5,50)
108      50  FORMAT(' ERRORS ENCOUNTERED')
109          GO TO 55
110      52  IF(.NOT.ALFBET(24)) GO TO 70
111      55  WRITE(5,60)
112      60  FORMAT(' EXECUTION BYPASSED')
113          GO TO 100
```

```
      70    XQT = .TRUE.
            EOF = .FALSE.
            LUN = 3
            WRITE(5,80)
      80    FORMAT(1H1)
            CALL ASEMBL
            CALL TIME(START)
            CALL INTERP(%500)
            CALL TIME(T)
            T = T - START
            WRITE(5,90) T
      90    FORMAT(/' EXECUTION TIME ',F7.2,' SECONDS'/1H1)
      100   CALL CSF('@ADD PL1.AFTER . ')
            STOP
      200   WRITE(5,210)
      210   FORMAT(/' COMPILER ERROR. PLEASE REPORT TO ADVISOR')
            STOP ABORT
      300   WRITE(5,310)
      310   FORMAT(/' EXECUTION TERMINATED')
            STOP ABORT
      C END OF MAIN SECTION
            START EDIT PAGE
            SUBROUTINE ASEMBL
      CHAPTER TWO :   ASSEMBLY OF STACKCODE
      C
      CLAUSE 1 : INITIALISATION
                       II = IXI - 1
                       IR = IXR - 1
                       IS = IXS - 2
                       IST = IXST - 1
                       IBC = IXBC
                       CALL XMARK(IXI,UNDEF,CONSTX)
            KTR = 41
            METER = 0
            CALL TIME(START)
      CLAUSE 2 :  READ LINE OF CODE AND SWITCH ON OPCODE
         70 READ(LUN,71,END=80,ERR=79) IOP,OP,P,Q,KONST
         71 FORMAT(15X,A3,T16,2I3,I12,7A6)
            IF(XQT) GO TO 711
              METER = METER + 1
              IF (METER.NE.1000) GOTO 711
                METER = 0
                  CALL TIME(T)
                  T = T - START
                  START  = T + START
                WRITE(5,710)  T
        710     FORMAT(' 1000 INSTRUCTIONS ASSEMBLED IN',F7.3,' SECS')
        711 CONTINUE
                       IF(IOP.EQ.'   ') GOTO 70
      CATALOGUE OF OP-CODES IN COMPILER ORDER <0-57>
      C     ABI,ABR,ADI,ADR,AND,DIF,DVI,DVR,EOF,FLO,
      C     FLT,INN,INT,IOR,MOD,MPI,MPR,NGI,NGR,NOT,
      C     ODD,SBI,SBR,SGS,SQI,SQR,STO,TRC,UNI,STP/
      C     CSP,DEC,ENT,FJP,INC,IND,IXA,LAO,LCA,LDO,
      C     MOV,MST,RET,SRO,XJP/CHK,CUP,EGU,GEQ,GRT,
      C     LDA,LDC,LEQ,LES,LOD,NEQ,STR,UJP
        712 OP = MOTATE(OP+1)
```

```
228                 GOTO 7424
229       C                                                 SET CONSTANTS
230       C 744 DECODE(35,7442,KONST(1)) M,N
231         744 READ(LUN,7442) M,N
232        7442 FORMAT(2X,2I15)
233               IF (IS.LT.IXS) GOTO 7445
234               DO 7443 I = IXS,IS,2
235               IF(ISTORE(I).NE.N) GOTO 7443
236               IF(ISTORE(I+1).EQ.M) GOTO 7423
237        7443   CONTINUE
238                 IF(IS.LT.IST-3) GOTO 7445
239                   WRITE(5,7444) KTR
240        7444       FORMAT(' INSTRUCTION',I6,': SET STORE OVERFLOW')
241                   GOTO 70
242        7445     IS = IS + 2
243                 ISTORE(IS) = N
244                 ISTORE(IS + 1) = M
245                 Q = IS
246                   CALL XMARK(IS,SET,2)
247                 GOTO 7424
248       C LCA   <LOADING A STRINGCONSTANT> OP=52  <P=STRINGLENGTH>
249          75 Q = IST + 1
250             IST = IST + P
251             IF (IST.LT.IXBD) GOTO 751
252                   WRITE(5,750) KTR
253          750      FORMAT(' STRING STORE OVERFLOW ASSEMBLING INSTRUCTION',I6)
254                   GOTO 77
255       C 751 DECODE(1,752,KONST(2))  (ISTORE(I),I=Q,IST)
256         751 READ(LUN,752) (ISTORE(I),I=Q,IST)
257         752  FORMAT(1X,79R1)
258               CALL XMARK(Q,CHAR,P)
259                 GOTO 77
260       CLAUSE 3 :  GENERAL ASSEMBLY PROCEDURES
261          77 IF(KTR.LE.KODEX) GOTO 771
262                   WRITE(5,770)
263          770      FORMAT(' CODESTORE OVERFLOW')
264                   STOP
265         771 FLD(0,6,KODE(KTR)) = OP
266             FLD(6,6,KODE(KTR)) = P
267             FLD(12,24,KODE(KTR)) = Q
268               KTR = KTR + 1
269                 GOTO 70
270          79 ISTAT = INSTAT(IDUMMY)
271             WRITE(5,790) ISTAT,KTR,OP,P,Q,KONST
272         790 FORMAT(' ERROR (STATUSWORD = ',O12,') IN INPUT TO ASSEMBLER'/
273            + ' INSTRUCTION ',I6,':',2I3,I12,9A6)
274             STOP
275          80 CONTINUE
276             RETURN
277             START EDIT PAGE
278             SUBROUTINE INTERP($)
279       CHAPTER THREE  :  INTERPRETER
280       C
281       CLAUSE 1 : INITIALISATIONS
282             KTR = 0
283             TOP = -1
284             BASE = 0
```

```
285           BOTTOM = IXI
286           MBOTTM = IXI
287           LOCAL = 4
288           MLOCAL = 4
289           NUBASE = 0
290           LETTER = 0
291           EOF = .FALSE.
292           FILE = 5
293           PROCS = 2
294           MAIN = MAXSTK
295           HEAP = MAXSTK
296           METER = 0
297           LW = 0
298           LP = 0
299           INWORD = 0
300           ITEM = 2
301       CLAUSE 2:  READ CURRENT INSTRUCTION AND SWITCH ON OP
302          60 K = KTR
303             KTR = KTR + 1
304       C NOTE THAT JUMPS IN STACKCODE ARE TO KTR-1 SINCE ARRAY KODE STARTS AT 1 NOT 0
305           OP = FLD(0,6,KODE(KTR))
306           P = FLD(6,6,KODE(KTR))
307           Q = FLD(12,24,KODE(KTR))
308           IF(ALFBET(20)) WRITE(5,6000) K,OP,P,Q
309       6000  FORMAT()
310           IF(ALFBET(5).AND.K.EQ.220) CALL PMD(TOP)
311            IF(OP.GT.0.AND.P.GE.0.AND.Q.GE.0) GOTO 601
312           WRITE(5,600) K,OP,P,Q
313         600 FORMAT(' ASSEMBLY ERROR INSTRUCTION',I6,2I3,I12)
314         601 CONTINUE
315           METER = METER + 1
316               IF (OP.GT.50) GOTO 64
317               IF (OP.GT.43) GOTO 69
318               IF (OP.GT.24) GOTO 67
319               IF (OP - 4) 61,62,63
320          61       TOP = TOP - 3
321                     GOTO 67
322          62       TOP = TOP -2
323                     GOTO 67
324          63       TOP = TOP - 1
325          67     IF(TOP.GT.LOCAL) GOTO 69
326           670           WRITE(5,6700) K,OP,P,Q
327          6700           FORMAT(' ACTIVE STACK UNDERFLOW:INSTRUCTION',I6,2I3,I12)
328                         CALL ERROR(LOCAL,2,MAXSTK,0)
329                         RETURN 1
330          64       TOP = TOP + 1
331                   IF (OP.EQ.59) TOP = TOP + 3
332          68     IF (TOP.LT.BOTTOM) GOTO 69
333           680           WRITE(5,6800) K,OP,P,Q
334          6800           FORMAT(' STACK OVERFLOW: INSTRUCTION',I6,2I3,I12)
335                         CALL ERROR(TOP,2,MLOCAL,IXI)
336                         RETURN 1
337          69 GOTO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,
338             +24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,
339             +46,47,48,49,50,51,52,53,54,55,56,57,58,59) , OP
340       C INSTRUCTIONS 1-3 INITIATED BY TESTING ACTIVE STATUS OF TOP (=TOP-3)
341       C DIF <TOP=TOP-3>                        *** SET OPERATIONS ***
```

```
342          1 ISTORE(TOP) = AND(ISTORE(TOP),COMPL(ISTORE(TOP+2)))
343              TOP = TOP + 1
344            ISTORE(TOP) = AND(ISTORE(TOP),COMPL(ISTORE(TOP+2)))
345                GOTO 60
346        C INT <TOP=TOP-3>
347          2 ISTORE(TOP) = AND(ISTORE(TOP),ISTORE(TOP+2))
348              TOP = TOP + 1
349            ISTORE(TOP) = AND(ISTORE(TOP),ISTORE(TOP+2))
350                GOTO 60
351        C UNI <TOP=TOP-3>
352          3 ISTORE(TOP) = OR(ISTORE(TOP),ISTORE(TOP+2))
353              TOP = TOP + 1
354            ISTORE(TOP) = OR(ISTORE(TOP),ISTORE(TOP+2))
355                GOTO 60
356        C INSTRUCTION 4 INITIATED BY TESTING ACTIVE STATUS OF TOP (=TOP-2)
357        C INN <TOP=TOP-2>                          *** SET RELATION ***
358          4 M = ISTORE(TOP)
359              N = M/36
360              M = 35 - M + N*36
361            ISTORE(TOP) = FLD(M,1,ISTORE(TOP+N+1))
362              CALL XMARK(TOP,BBOL,1)
363                GOTO 60
364        C INSTRUCTIONS 5-24 INITIATED BY TESTING ACTIVE STATUS OF TOP (=TOP-1)
365        C ADI,ADR <TOP=TOP-1>                      *** ARITHMETIC OPERATIONS ***
366          5 ISTORE(TOP) = ISTORE(TOP) + ISTORE(TOP+1)
367                GOTO60
368          6 STORE(TOP) = STORE(TOP) + STORE(TOP+1)
369                GOTO 60
370        C SBI,SBR <
371          7 ISTORE(TOP) = ISTORE(TOP) - ISTORE(TOP+1)
372                GOTO 60
373          8 STORE(TOP) = STORE(TOP) - STORE (TOP+1)
374                GOTO 60
375        C MPI,MPR
376          9 ISTORE(TOP) = ISTORE(TOP) * ISTORE(TOP+1)
377                GOTO 60
378         10 STORE(TOP) = STORE(TOP) * STORE(TOP+1)
379                  GOTO 60
380        C IXA       <INDEXED ADDRESSING>
381         11 ISTORE(TOP) = G * ISTORE(TOP+1) + ISTORE(TOP)
382                GOTO 60
383        C DVI,DVR
384         12 ISTORE(TOP) = ISTORE(TOP) / ISTORE(TOP+1)
385                GOTO 60
386         13 STORE(TOP) = STORE(TOP) / STORE(TOP+1)
387                GOTO 60
388        C MOD
389         14 ISTORE(TOP) = MOD(ISTORE(TOP),ISTORE(TOP+1))
390                  GOTO 60
391        C AND                                      *** LOGICAL OPERATIONS ***
392         15 ISTORE(TOP) = AND(ISTORE(TOP),ISTORE(TOP+1))
393                GOTO 60
394        C IOR
395         16 ISTORE(TOP) = OR(ISTORE(TOP),ISTORE(TOP+1))
396                GOTO 60
397        C NEG                                      *** RELATIONS ***
398         17 ASSIGN 184 TO EQUAL
```

```
399           ASSIGN 183 TO NOTEQL
400               GOTO 180
401       C EQU
402            18 ASSIGN 183 TO EQUAL
403               ASSIGN 184 TO NOTEQL
404           180     IF (P.EQ.1RM) GOTO 182
405                   IF(MARK(TOP).NE.SET) GOTO 181
406                     TOP = TOP - 2
407                       IF(TOP.LE.LOCAL) GOTO 670
408                     IF(ISTORE(TOP).NE.ISTORE(TOP+2)) GOTO 1822
409                     IF(ISTORE(TOP+1).EQ.ISTORE(TOP+3)) GOTO 1821
410                           GOTO 1822
411           181       IF(ISTORE(TOP).EQ.ISTORE(TOP+1)) GOTO 1821
412                           GOTO 1822
413           182     M = ISTORE(TOP) - 1
414                   N = ISTORE(TOP+1)
415                   IF(MARK(N).EQ.SET) Q = Q+Q
416                   N=N-1
417                     DO 1820 I=1,Q
418                     IF(ISTORE(I+M).NE.ISTORE(I+N)) GOTO 1822
419           1820      CONTINUE
420           1821          GOTO EQUAL, (183,184)
421           1822          GOTO NOTEQL, (183,184,194,212)
422            183 ISTORE(TOP) = 1
423                   GOTO 185
424            184 ISTORE(TOP) = 0
425            185   CALL XMARK(TOP,BBOL,1)
426                   GOTO 60
427       C GEQ
428             19 IF(P.EQ.1RM) GOTO 191
429                   IF(MARK(TOP).NE.SET) GOTO 190
430                     TOP = TOP - 2
431                       IF(TOP.LE.LOCAL) GOTO 670
432                 IF(AND(ISTORE(TOP),ISTORE(TOP+2)).NE.ISTORE(TOP+2)) GOTO 184
433                 IF(AND(ISTORE(TOP+1),ISTORE(TOP+3)).EQ.ISTORE(TOP+3)) GOTO 183
434                                                       GOTO 184
435           190       IF(ISTORE(TOP).GE.ISTORE(TOP+1)) GOTO 183
436                                                       GOTO 184
437             191 ASSIGN 183 TO EQUAL
438                 ASSIGN 194 TO NOTEQL
439           192     M = ISTORE(TOP) - 1
440                   N = ISTORE(TOP+1) - 1
441                     DO 193 I = 1,Q
442                     IF(ISTORE(I+M).NE. ISTORE(I+N)) GOTO 1822
443           193       CONTINUE
444                       GOTO 1821
445             194 IF(ISTORE(I+M).GT.ISTORE(I+N)) GOTO 183
446                                                       GOTO 184
447       C GRT
448             20 IF(P.EQ.1RM) GOTO 200
449                   IF(ISTORE(TOP).GT.ISTORE(TOP+1)) GOTO 183
450                                                       GOTO 184
451           200 ASSIGN 184 TO EQUAL
452               ASSIGN 194 TO NOTEQL
453                   GOTO 192
454       C LEQ
455             21 IF(P.EQ.1RM) GOTO 211
```

```
                IF(MARK(TOP).NE.SET) GOTO 210
                  TOP = TOP - 2
                    IF(TOP.LE.LOCAL) GOTO 670
          IF(AND(ISTORE(TOP),ISTORE(TOP+2)).NE.ISTORE(TOP)) GOTO 184
          IF(AND(ISTORE(TOP+1),ISTORE(TOP+3)).NE.ISTORE(TOP+1)) GOTO 183
                                            GOTO 184
  210       IF(ISTORE(TOP).LE.ISTORE(TOP+1)) GOTO 183
                                            GOTO 184
  211 ASSIGN 183 TO EQUAL
      ASSIGN 212 TO NOTEQL
          GOTO 192
  212 IF(ISTORE(I+M).LT.ISTORE(I+N)) GOTO 183
                                            GOTO 184
C LES
   22 IF (P.EQ.IRM) GOTO 220
                IF(ISTORE(TOP).LT.ISTORE(TOP+1)) GOTO 183
                                            GOTO 184
  220 ASSIGN 184 TO EQUAL
      ASSIGN 212 TO NOTEQL
          GOTO 192
C STO                                        *** LOADS AND STORES ****
   23 N = MARK(TOP +1)
        IF (N.EQ.UNDEF) GOTO 420
      IF(N.NE.SET) GOTO 230
        TOP = TOP - 1
          IF(TOP.LE.LOCAL) GOTO 670
        M = ISTORE(TOP)
        ISTORE(M+1) = ISTORE(TOP+2)
        J=2
        GOTO 231
  230 M = ISTORE(TOP)
      J = 1
      IF(N.NE.RE) GOTO 231
        STORE(M) = STORE(TOP +1)
        GOTO 232
  231 ISTORE(M) = ISTORE(TOP+1)
  232   CALL XMARK(M,N,J)
      TOP = TOP - 1
          GOTO 60
C MOV
   24 M = ISTORE(TOP) - 1
      N = ISTORE(TOP+1) - 1
      DO 240 I = 1,0
        J = I + N
        P = MARK(J)
      IF(P.EQ.RE) GOTO 241
        ISTORE(I+M) = ISTORE(J)
        GOTO 240
  241 STORE(I+M) = STORE(J)
  240   CALL XMARK(M+I,P,1)
      TOP = TOP - 1
          GOTO 60
C INSTRUCTIONS 25-43 INITIATED BY TESTING ACTIVE STATUS OF TOP
C ODD
   25 ISTORE(TOP) = MOD(ISTORE(TOP),2)
        CALL XMARK(TOP,BOOL,1)
          GOTO 60
```

```
513       C ABI,ABR <TOP = TOP>                    *** ARITHMETIC OPERATIONS ***
514          26 ISTORE(TOP) = IABS(ISTORE(TOP))
515                GOTO 60
516          27 STORE(TOP) = ABS(STORE(TOP))
517                GOTO 60
518       C NGI,NGR
519          28 ISTORE(TOP) = -ISTORE(TOP)
520                GOTO 60
521          29 STORE(TOP) = -STORE(TOP)
522                GOTO 60
523       C SGI,SGR
524          30 ISTORE(TOP) = ISTORE(TOP)*ISTORE(TOP)
525                GOTO 60
526          31 STORE(TOP) = STORE(TOP) * STORE(TOP)
527                GOTO 60
528       C DEC
529          32 ISTORE(TOP) = ISTORE(TOP) - Q
530                GOTO 60
531       C INC
532          33 ISTORE(TOP) = ISTORE(TOP) + Q
533                GOTO 60
534       C NOT                                    *** LOGICAL OPERATION ***
535          34 IF (ISTORE(TOP).EQ.1) GOTO 340
536                ISTORE(TOP) = 1
537                GOTO 60
538         340 ISTORE(TOP) = 0
539                GOTO 60
540       C FLO,FLT                                *** TYPE CONVERSIONS ***
541          35     IF(TOP.LE.LOCAL+1) GO TO 670
542                 CALL XMARK(TOP-1,RE,1)
543                STORE(TOP-1) = FLOAT(ISTORE(TOP-1))
544                   GOTO 60
545          36    CALL XMARK(TOP,RE,1)
546                STORE(TOP) = FLOAT(ISTORE(TOP))
547                   GOTO 60
548       C TRC
549          37    CALL XMARK(TOP,INT,1)
550                ISTORE(TOP) = IFIX(STORE(TOP))
551                   GOTO 60
552       C SGS
553          38      IF(TOP+1.GE.BOTTOM) GOTO 680
554                  N = ISTORE(TOP)
555                  M = N/36
556                  N = 35 - N + M*36
557                ISTORE(TOP) = 0
558                ISTORE(TOP+1) = 0
559                FLD(N,1,ISTORE(TOP+M)) = 1
560                  CALL XMARK(TOP,SET,2)
561                TOP = TOP + 1
562                   GOTO 60
563       C FJP
564          39 IF(ISTORE(TOP).EQ.0) KTR = Q
565                TOP = TOP - 1
566                   GOTO 60
567       C XJP <INDEXED JUMP>                     *** CONTROL OPERATION ***
568          40 KTR = ISTORE(TOP) + Q
569                TOP = TOP - 1
```

```
570                 GOTO 60
571       C IND                                        *** LOADS AND STORES ***
572          41 Q = ISTORE(TOP) + Q
573       C GENERAL ROUTINE FOR FETCHING FROM ADDRESS Q
574       C USED BY IND, LDO, LOD AND MODIFIED-LDC
575          415 N = MARK(Q)
576              IF(N.NE.UNDEF) GOTO 411
577                   WRITE(5,410) Q,K
578          410      FORMAT(' VALUE UNDEFINED AT ADDRESS',I6,'; CODE LINE =',I6)
579                   CALL PMD(Q)
580                   RETURN 1
581          411 IF(N.NE.RE) GOTO 412
582                STORE(TOP) = STORE(Q)
583                GOTO 413
584          412 ISTORE(TOP) = ISTORE(Q)
585          413   CALL XMARK(TOP,N,1)
586              IF(N.NE.SET) GOTO 60
587                TOP = TOP + 1
588                  IF(TOP.GE.BOTTOM) GOTO 680
589              ISTORE(TOP) = ISTORE(Q+1)
590                CALL XMARK(TOP,SET,1)
591                  GOTO 60
592       C SRO  <STORE IN ADDRESS AT BASELEVEL>
593          42 N = MARK(TOP)
594                IF (N.NE.UNDEF) GOTO 4202
595           4200   WRITE(5,4201) K
596           4201   FORMAT(' ATTEMPT TO STORE UNDEFINED VALUE. CODELINE ',I6)
597                  CALL PMD(TOP+1)
598                RETURN 1
599           4202 CONTINUE
600              IF(N.NE.SET) GOTO 420
601                TOP = TOP - 1
602                  IF(TOP.LE.LOCAL) GO TO 670
603                ISTORE(Q+1) = ISTORE(TOP+1)
604                J = 2
605                GOTO 421
606           420 J = 1
607              IF(N.NE.RE) GOTO 421
608                STORE(Q) = STORE(TOP)
609                GOTO 422
610           421 ISTORE(Q) = ISTORE(TOP)
611           422   CALL XMARK(Q,N,J)
612                 TOP = TOP - 1
613                   GOTO 60
614       C STR
615          43 ASSIGN 42 TO FETCH
616             IF(Q.EQ.0) Q = -1
617       C GENERAL ROUTINE FOR CALCULATING ADDRESS
618       C USED BY STR(ON WAY TO SRO), LDA(TO LAO), LOD(TO LDO)
619            432   M = BASE
620            430 IF(P.EQ.0) GO TO 431
621                  P = P-1
622                  M = ISTORE(M+1)
623                    GOTO 430
624            431 Q = M + Q
625                    GOTO FETCH, (42,53,415)
626       C INSTRUCTIONS 44-50 INITIATED WITHOUT TESTING ACTIVE STATUS OF TOP
```

```
C STP                                   *** CONTROL OPERATIONS ***
   44 CONTINUE
      RETURN
C UJP
   45 IF(P.EQ.99) GO TO 40
      KTR = Q
          GOTO 60
C CHK      <NOT IMPLEMENTED>  *** RUNTIME SUBSCRIPT CHECKING ***
C STF = STORE FUNCTION RESULT <TEMPORARY EXPEDIENT>
   46   IF (TOP.LE.LOCAL) GOTO 670
      Q = BASE - 1
        GOTO 42
C CSP                                   *** PROCEDURE CALLS ***
   47     IF(Q.GT.20) GOTO 4797
          IF(Q.EQ.20) GOTO 4796
          IF(Q.GE.15) GOTO 479
          IF(Q.GE.11) GOTO 478
          IF(Q.GE.6) GOTO 473
          IF(Q.GE.3) GOTO 470
          IF(Q.EQ.2) GOTO 47300
C SUM
      TOP = TOP - 2
      P = ISTORE(TOP)
      Q = ISTORE(TOP+1)
      IE = Q + P - 1
      ISTORE(TOP) = 0
      DO 147 IA = Q,IE
147   ISTORE(TOP) = ISTORE(TOP) + ISTORE(IA)
      GOTO 60
C READ <Q=3:RDI; Q=4:RDN; Q=5:RDC; Q=1:GET>
  470   IF (TOP.LE.LOCAL) GOTO 670
      M = ISTORE(TOP)
        TOP = TOP - 2
      IF(INWORD.NE.0) GOTO 4702
 4700   IF(XQT) READ(8,4701,END=47295) IN
 4701   FORMAT(13A6,A2)
        IF(.NOT.XQT) CALL PREAD(IC,IN,ISTAT,$47290,$47295)
47002     FLD(12,6,IN(14)) = 0
47005 CONTINUE
            INWORD = 1
            INBIT = 0
        LETTER = FLD(0,6,IN(1))
      IF(LETTER.EQ.1R%) LETTER = 0
 4702 GOTO (60,60,471,471,4703), Q
C RDC
 4703 ISTORE(M) = LETTER
        CALL XMARK(M,CHAR,1)
      IF(LETTER.NE.0) GOTO 4704
          INWORD = 0
          GOTO 60
 4704 INBIT = INBIT + 6
      IF(INBIT.NE.36) GOTO 4705
        INWORD = INWORD + 1
        INBIT = 0
 4705 LETTER = FLD(INBIT,6,IN(INWORD))
      IF(LETTER.EQ.1R%) LETTER = 0
          GOTO 60
```

```
684       C RDI  <LOOP=0>  &  RDR  <LOOP=2>
685         471 IF(LETTER.EQ.0) GOTO 4700
686       C                               PRELIMINARY LOOP SKIPPING TO FIRST RELEVANT SYMBOL
687             DO 4711 IW = INWORD,14
688               DO 4710 IB = INBIT,30,6
689               LETTER = FLD(IB,6,IN(IW))
690                 IF(LETTER.EQ.BLANK) GOTO 4710
691                 IF(LETTER.NE.PLUS) GOTO 4712
692          4710   CONTINUE
693          4711 INBIT = 0
694          4712 IF(LETTER.EQ.1R%) LETTER = 0
695               IF(LETTER.EQ.0) GOTO 4700
696       C                               POSITION TO 1ST WORD,BLANK IRRELEVANT BITS
697             INPTR = IW
698             INBIT = 0
699               LGTH = 0
700          4713 IF(INBIT.EQ.IB) GOTO 4714
701               FLD(INBIT,6,IN(INWORD)) = BLANK
702                 INBIT = INBIT + 6
703                 LGTH = LGTH + 1
704               GOTO 4713
705          4714 IF(LETTER.NE.MINUS) GOTO 472
706                 IB = IB + 6
707                 LGTH = LGTH + 1
708               IF(IB.EQ.36) GOTO 472
709                 IW = IW + 1
710                 IB = 0
711       C RDI
712           472 LOOP = 0
713               EXPT = .FALSE.
714                 IF(Q.EQ.3) GOTO 4721
715       C RDR
716               LOOP = 2
717       C                               LOOP TO FIND LENGTH OF NUMBER TO BE READ
718          4721 DO 4727 INWORD = IW,14
719                 DO 4726 INBIT = IB,30,6
720                 LETTER = FLD(INBIT,6,IN(INWORD))
721                   IF(LETTER.LT.NOUGHT) GOTO 4722
722                   IF(LETTER.LE.NINE) GOTO 4724
723                     IF(LOOP.NE.2) GOTO 4728
724                       IF(LETTER.NE.POINT) GOTO 4728
725                         LOOP = 1
726                         GOTO 4726
727          4722         IF(LOOP.EQ.0) GOTO 4728
728                       IF (EXPT) GOTO 4723
729                         IF(LETTER.NE.1RE) GOTO 4728
730                           EXPT = .TRUE.
731                           GOTO 4726
732          4723           IF(LETTER.EQ.PLUS) GOTO 4725
733                         IF(LETTER.EQ.MINUS) GOTO 4725
734                             GOTO 4728
735          4724       IF(.NOT.EXPT) GOTO 4726
736          4725           EXPT = .FALSE.
737                         LOOP = 0
738          4726       LGTH = LGTH + 1
739          4727 IB = 0
740       C                               READ THE NUMBER
```

```
      4728 DECODE(LGTH,4729,IN(INPTR))  ISTORE(M)
      4729 FORMAT()
             CALL XMARK(M,Q-2,1)
               GOTO 60
     47290 WRITE(5,47291) K
     47291 FORMAT(' ERROR OCCURRED IN READ, INSTRUCTION: ',I6)
           CALL PMD(TOP)
           RETURN 1
     47295   EOF = .TRUE.
             LETTER = '$$$$$$'
             GO TO 60
     C WRITE <Q=6:WRI; Q=7:WRO; Q=8:WRR; Q=9:WRC; Q=10:WRS; Q=2:PUT>
     C SIGN
     47300 IF(MARK(TOP).NE.RE) GOTO 47307
           ISTORE(TOP) = STORE(TOP)
           CALL XMARK(TOP,INT,1)
     47307 IF(ISTORE(TOP)) ,47308,47309
           ISTORE(TOP) = -1
           GOTO 60
     47308 ISTORE(TOP) = 0
           GOTO 60
     47309 ISTORE(TOP) = 1
           GOTO 60
       473   TOP = TOP - 1
               IF(TOP.LE.LOCAL) GOTO 670
           LGTH = ISTORE(TOP+1)
     47301 LP = LP + LGTH
             IF(LP.LE.132) GOTO 4731
      4730     IF (LW.NE.0) GOTO 47304
               WRITE(5,47303)
      47303     FORMAT(' ')
                 GOTO 4731
      47304     LAYOUT(ITEM) = 1H)
      47302     FORMAT(1X,3I6/(1X,20A6/))
                IF(.NOT.EOF) WRITE(FILE,LAYOUT)(LINE(I),I=1,LW)
                EOF = .FALSE.
                  LW = 0
                  LP = 0
                  ITEM = 2
      4731 GOTO (60,47600,60,60,60,474,4741,475,476,477,60,60,47601), G
     C WRI & WRO
       474 ENCODE(6,4740,LAYOUT(ITEM)),LGTH
      4740 FORMAT('(I',I4)
                GOTO 4743
      4741 ENCODE(6,4742,LAYOUT(ITEM)), LGTH
      4742 FORMAT('(O',I4)
      4743   ITEM = ITEM + 1
             LW = LW + 1
           LINE(LW) = ISTORE(TOP)
             TOP = TOP - 1
             GOTO 60
     C WRR   <USER SETTING OF DECIMAL POINT NOT YET IMPLEMENTED>
       475 M = LGTH - 6
             IF(M.LT.0) GOTO 4751
               IF (M.GT.8) M = 8
               GOTO 4752
      4751   M = 0
```

```
798       4752 ENCODE(6,4753,LREAL(2)) LGTH,M
799       4753 FORMAT(I3,'.',I1,')')
800              LW = LW + 1
801            ENCODE(LGTH,LREAL,LINE(LW)) STORE(TOP)
802            M = 0
803            IF(LGTH.LE.6) GOTO 4755
804                M = LGTH/6
805                LGTH = LGTH - M*6
806              ENCODE(6,4754,LAYOUT(ITEM)) M
807       4754   FORMAT('',',I3,'A6')
808              ITEM = ITEM + 1
809       4755 LAYOUT(ITEM) = AFIELD(LGTH)
810              ITEM = ITEM + 1
811              LW = LW + M
812                GOTO 60
813       C WRC
814         476 N = TOP
815       47600 IF (ISTORE(N).NE.0) GOTO 4760
816              Q = 10 + ISTORE(N+1)
817                TOP = TOP - 1
818               GOTO 4730
819       47601 WRITE(5,47602)
820       47602 FORMAT(1H1)
821            GO TO 60
822        4760 IF (LGTH.LE.1) GOTO 4762
823                LGTH = LGTH - 1
824              ENCODE(6,4761,LAYOUT(ITEM)) LGTH
825        4761   FORMAT('',',I4,'X')
826                ITEM = ITEM + 1
827        4762 LAYOUT(ITEM) = 3H,R1
828              ITEM = ITEM + 1
829              LW = LW + 1
830            LINE(LW) = ISTORE(N)
831              TOP = TOP - 2
832                GOTO 50
833       C WRS
834         477 LGTHX = ISTORE(TOP)
835              TOP = TOP - 1
836                IF(TOP.LE.LOCAL) GOTO 670
837            M = ISTORE(TOP)
838              TOP = TOP - 1
839            IF (LGTHX.LT.LGTH) LGTH = LGTHX
840              INPTR = LW + 1
841            DO 4770 LW = INPTR,132
842            DO 4770 OUTBIT = 0,30,6
843            FLD(OUTBIT,6,LINE(LW)) = ISTORE(M)
844              LGTH = LGTH - 1
845              IF (LGTH.EQ.0) GOTO 47700
846              M = M + 1
847        4770 CONTINUE
848       47700 CONTINUE
849              INPTR = LW - INPTR
850               IF (INPTR.EQ.0) GOTO 4771
851              ENCODE(6,4754,LAYOUT(ITEM)) INPTR
852                ITEM = ITEM + 1
853        4771     IF (OUTBIT.EQ.0) GOTO 4772
854                   OUTBIT = OUTBIT/6 + 1
```

```
855                 LAYOUT(ITEM) = AFIELD(OUTBIT)
856          4772     ITEM = ITEM + 1
857                   GOTO 60
858          C PAK,NEW,SAV,RST
859            478 M = G - 10
860                GOTO (4781,4782,4783,4784),M
861          C PAK
862           4781 TOP = TOP - 2
863                 IF (TOP.LE.LOCAL) GOTO 670
864                M1 = ISTORE(TOP+2)
865                M2 = ISTORE(TOP+1)
866                N = ISTORE(TOP)
867                DO 4780 I = 1,N
868                  J = I-1
869           4780 ISTORE(M2+J) = ISTORE(M1+J)
870                 CALL XMARK(M2,MARK(M1),N)
871                 TOP = TOP - 1
872                   GOTO 60
873          C NEW
874           4782    IF (TOP.LE.LOCAL) GOTO 670
875                N = ISTORE(TOP)
876                M = BOTTOM - N
877                IF (M.LT.MBOTTM) MBOTTM = M
878                  IF (M.GT.TOP) GOTO 47821
879                        WRITE(5,47820)
880          47820         FORMAT(' HEAP OVERFLOW')
881                          CALL PMD(M)
882                        RETURN 1
883          47821   CALL XMARK(M,UNDEF,N)
884                BOTTOM = M
885                TOP = TOP - 1
886                  IF (TOP.LE.LOCAL) GOTO 670
887                N = ISTORE(TOP )
888                  IF (N.EQ.0) GOTO 47823
889                DO 47822 I = 1,N
890                  TOP = TOP - 2
891                    IF (TOP.LE.LOCAL) GOTO 670
892                  J = ISTORE(TOP+1)
893                ISTORE(BOTTOM+J) = ISTORE(TOP)
894          47822   CALL XMARK(BOTTOM+J,MARK(TOP),1)
895          47823   TOP = TOP - 1
896                    IF (TOP.LE.LOCAL) GOTO 670
897                N = ISTORE(TOP)
898                ISTORE(N) = BOTTOM
899                  CALL XMARK(N,AD,1)
900                  TOP = TOP - 1
901                    GOTO 60
902          C SAV
903           4783   IF (TOP.LE.LOCAL) GOTO 670
904                N = ISTORE(TOP)
905                  CALL XMARK(N,INT,1)
906                ISTORE(N) = BOTTOM
907                  TOP = TOP - 1
908                    GOTO 60
909          C RST
910           4784   IF (TOP.LE.LOCAL) GOTO 670
911                BOTTOM = ISTORE(TOP)
```

```
 4785   TOP = TOP - 1
          GOTO 60
C SIN,COS,EXP,SQT,LOG
  479   IF (TOP.LE.LOCAL)  GOTO 670
        IF (MARK(TOP).NE.INT) GOTO 4790
          STORE(TOP) = ISTORE(TOP)
          CALL XMARK (TOP,RE,1)
 4790 Q = G - 14
        GOTO (4791,4792,4793,4794,4795),Q
 4791 STORE(TOP) = SIN(STORE(TOP))
          GOTO 60
 4792 STORE(TOP) = COS(STORE(TOP))
          GOTO 60
 4793 STORE(TOP) = EXP(STORE(TOP))
          GOTO 60
 4794 STORE(TOP) = SQRT(STORE(TOP))
          GOTO 60
 4795 STORE(TOP) = ALOG(STORE(TOP))
          GOTO 60
 4796 TOP = TOP + 1
        CALL TIME(STORE(TOP))
        CALL XMARK(TOP,RE,1)
          GOTO 60
C MIN,MAX
 4797 TOP = TOP - 1
        Q = G - 20
        GOTO(4798,4799),Q
 4798 ISTORE(TOP) = MIN(ISTORE(TOP),ISTORE(TOP+1))
        GOTO 60
 4799 ISTORE(TOP) = MAX(ISTORE(TOP),ISTORE(TOP+1))
        GOTO 60
C CUP
   48 BASE = NUBASE
        NUBASE = ISTORE(BASE+3)
        ISTORE(BASE+3) = KTR
          CALL XMARK(BASE+3,PROL,1)
        KTR = Q
          GOTO 60
C ENT
   49 LOCAL = BASE+Q
        IF (LOCAL.GT.MLOCAL) MLOCAL = LOCAL
          IF (LOCAL.GE.BOTTOM) GOTO 680
          IF (LOCAL.EQ.TOP) GOTO 491
            IF (BASE.NE.0) GOTO 4901
              CALL XMARK(TOP+1,CHAR,2)
               TOP = TOP + 2
 4901   CALL XMARK(TOP+1,UNDEF,LOCAL-TOP)
        TOP = LOCAL
  491 ISTORE(BASE) = LOCAL
          CALL XMARK(BASE,PROL,1)
        IF(K.EQ.6289) FILE = 3
          GOTO 60
C RET
   50 TOP = BASE - 1
        KTR = ISTORE(BASE+3)
        BASE = ISTORE(BASE+2)
        LOCAL = ISTORE(BASE)
```

```
969               FILE = 5
970                  GOTO 60
971         C INSTRUCTIONS 51-58 INITIATED BY TESTING AVAILABILITY OF TOP (=TOP+1)
972         C  <LOAD IMMEDIATE> LDC
973            51 IF(P.LE.2) P = P - 1
974               * CALL XMARK(TOP,P,1)
975               IF(P.EQ.AD) GOTO 510
976               ISTORE(TOP) = G
977                  GOTO 60
978           510 ISTORE(TOP) = IMAX
979                  GOTO 60
980         C  <LOAD ADDRESS>  LCA,LAO,LDA
981            52 CONTINUE
982            53 ISTORE(TOP) = G
983                 CALL XMARK(TOP,AD,1)
984                  GOTO 60
985            54 ASSIGN 53 TO FETCH
986                  GOTO 432
987         C  <LOAD FROM ADDRESS>  MODIFIED-LDC,LDO,LOD
988            55 CONTINUE
989            56 IF (K.EQ.5820) FILE = 15
990               GOTO 415
991            57 ASSIGN 415 TO FETCH
992                  GOTO 432
993         C EOF                                    *** TESTS ***
994            58 ISTORE(TOP) = G
995               IF (EOF) ISTORE(TOP) = 1
996                 CALL XMARK(TOP,BBOL,1)
997                  GOTO 60
998         C INSTRUCTION 59 INITIATED BY TESTING AVAILABILITY OF TOP (=TOP+4)
999         C MST <TOP=TOP+4>
1000           59 IF (Q.EQ.0) GOTO 5900
1001              TOP = TOP + 1
1002              IF (TOP.GE.BOTTOM) GOTO 680
1003                CALL XMARK(TOP-4,UNDEF,1)
1004          5900 CONTINUE
1005                CALL XMARK(TOP-3,UNDEF,4)
1006              M = BASE
1007           590 IF (P.EQ.0) GOTO 591
1008              P = P - 1
1009              M = ISTORE(M+1)
1010              GOTO 590
1011           591 ISTORE(TOP-2) = M
1012              ISTORE(TOP-1) = BASE
1013                CALL XMARK(TOP-2,PROL,2)
1014              ISTORE(TOP) = NUBASE
1015              NUBASE = TOP - 3
1016                 GOTO 60
1017              START EDIT PAGE
1018              SUBROUTINE XMARK(I,J,K)
1019              INTEGER L,M,N
1020              IF (K.EQ.0) RETURN
1021              M = I/12
1022              N = I - M*12
1023              M = M + 1
1024              IF (M.LE.MX) GOTO 1
1025              WRITE(5,10) KTR,OP,P,Q,M
```

```
1026          10 FORMAT(' INSTRUCTION',I6,2I3,I12,' INDEX TO MARKER =',I12)
1027             PROCS = 10
1028             CALL PMD(TOP)
1029           1 CONTINUE
1030             N = N + 3
1031             IF (N.EQ.0) N = 0
1032             L = K
1033           2 FLD(N,3,MARKER(M)) = J
1034             IF(L.LE.1) RETURN
1035             L = L - 1
1036             N = N + 3
1037             IF (N.NE.36) GOTO 2
1038             M = M + 1
1039             N = 0
1040             GOTO 2
1041             START EDIT PAGE
1042             FUNCTION MARK(I)
1043             INTEGER M,N
1044             M = I/12
1045             N = I - M*12
1046             M = M + 1
1047             IF (M.LE.MX) GOTO 1
1048             WRITE(5,10) K,OP,P,Q,M
1049          10 FORMAT(' INSTRUCTION',I6,2I3,I12,' INDEX TO MARKER =',I12)
1050             PROCS = 10
1051             CALL PMD(TOP)
1052           1 CONTINUE
1053             N = N + 3
1054             IF (N.EQ.0) N = 0
1055             MARK = FLD(N,3,MARKER(M))
1056             RETURN
1057             START EDIT PAGE
1058             SUBROUTINE PMD(ADDR)
1059             LOGICAL MORE
1060              INTEGER I,J,K,L,M,N,ADDR,LIST(5,PMDKNT),LETTER(6)
1061             DATA LETTER/1HI,1HR,1HS,1HB,1H¬,1H*/
1062             WRITE(5,111)
1063         111 FORMAT(//' POST MORTEM DUMP'/1X,16('-'))
1064             WRITE(5,1100) IN
1065        1100 FORMAT(/' INPUT BUFFER:'/1X,13A6,A2/' OUTPUT BUFFER:')
1066             LAYOUT(ITEM) = 1M)
1067             WRITE(5,LAYOUT) (LINE(I),I=1,LW)
1068             M = LOCAL+10
1069             IF (TOP.GT.M) M = TOP
1070             IF (ADDR.GT.M.AND.ADDR.LT.BOTTOM) M = ADDR
1071             L = LOCAL + 1
1072             WRITE(5,110)  TOP
1073         110 FORMAT(//' ACTIVE STACK   TOP =',I6/1X,12('-'))
1074             IPROC = PROCS + 1
1075             GOTO 5
1076        1110 M = LOCAL
1077             L = BASE
1078             GOTO 2
1079           1 M = L - 1
1080             L = ISTORE(L+2)
1081           2 IF (L.EQ.0) GOTO 4
1082             K = ISTORE(L+3) - 1
```

```
1140            IF (ISTORE(J).EQ.0) GOTO 49
1141            LIST(4,KNT) = 6H
1142            LIST(5,KNT) = 4HTRUE
1143              GOTO 99
1144           49 LIST(4,KNT) = 6H     F
1145            LIST(5,KNT) = 4HALSE
1146            GOTO 99
1147           97 MK = 1
1148           98 MORE = .TRUE.
1149           99 KNT = KNT + 1
1150            IF (KNT.LE.PMDKNT) GOTO 991
1151            WRITE(5,990) LIST
1152          990 FORMAT(1X,5(I6,A1,2A6,A4))
1153            KNT = 1
1154          991 IF (.NOT.MORE) GOTO 100
1155            MORE = .FALSE.
1156            IF (K.EQ.UNDEF) GOTO 61
1157            IF(K.EQ.CHAR) GOTO 81
1158            GOTO 82
1159          100 CONTINUE
1160            IF (MK) 1000,1001,1002
1161         1002 IF (N.NE.0) GOTO 1004
1162                IF (KNT.EQ.1) GOTO 1003
1163                  KNT = KNT - 1
1164                    GOTO 1001
1165         1004       IBEG = M - N + 1
1166                    LIST(1,KNT) = IBEG
1167                    LIST(2,KNT) = 1HC
1168                    ENCODE(16,800,LIST(3,KNT)) (ISTORE(I),I=IBEG,M)
1169                    N = 0
1170                    GOTO 1001
1171         1000 LIST(2,KNT) = 1H+
1172         1001 WRITE(5,990) ((LIST(I,J),I=1,5),J=1,KNT)
1173            KNT = 1
1174         1003 CONTINUE
1175            IF (L.EQ.0) GOTO 101
1176            IF(L.EQ.BOTTOM)  RETURN
1177            IPROC = IPROC - 1
1178            IF (IPROC-PROCS) 1,1110,1110
1179          101 J = SX
1180            WRITE (5,1010) BOTTOM,J
1181         1010 FORMAT(//' HEAP & CONSTANTS',2I10/1X,16('-')/)
1182            IF (HEAP.EQ.0) RETURN
1183            L = BOTTOM
1184            M = L + HEAP
1185            IF (M.GT.SX) M = SX
1186            GOTO 5
1187            SUBROUTINE ERRCR(I,J,K,L)
1188            INTEGER ADDR,A,B,C
1189            ADDR = I
1190            A = PROCS
1191            B = MAIN
1192            C = HEAP
1193            PROCS = J
1194            MAIN = K
1195            HEAP = L
1196            CALL PMD(ADDR)
```

```
CODE*PL1.TIME
     1        . THIS ROUTINE GIVES TOTAL ACCUMULATED EXECUTION TIME
     2       .
     3       . THE CALLING FORMAT IS
     4       .
     5        .       CALL TIME(T)
     6       .
     7        . WHERE T IS A REAL VARIABLE. THE TOTAL ACCUMULATED
     8        . EXECUTION TIME IS STORED IN T IN SECONDS CORRECT TO THE NEAREST
     9        . MILLISECOND.
    10       .
    11                AXR$                              .
    12       $(0),BUF RES       23                    . PCT BUFFER
    13       REG      RES       2                     . REGISTER STORE
    14       $(1),TIME*                               .
    15                DS        A2,REG                . SAVE REGISTERS
    16                LA        A0,(23,BUF)           . PCT BUFFER ACCESS PACKET
    17                ER        PCT$                  .
    18                LA,XU     A2,155                . BIAS FOR CONVERSION TO F.P.
    19                LCF       A2,BUF+22             . CONVERT TO FLOATING POINT
    20                DSC       A2,36                 . SWOP A2 AND A3
    21                FM        A2,(0.0002)           . CONVERT TO SECONDS
    22                SA        A2,*0,X11             . STORE TIME IN T.
    23                DL        A2,REG                . RESTORE REGISTERS
    24                J         2,X11                 . RETURN
    25                END                             .
```

```
CODE*PL1.PREAD
     1        . THIS ROUTINE INPUTS TO OR READS FROM AN ELEMENT SPECIFIED ON A
     2        . PROCESSOR CALL AND TO UPDATE IT INTO ANOTHER IF REQUIRED.
     3        . OPTIONS ARE I & U AS USUAL FOR PROCESSORS.
     4       . THE CALLING FORMAT IS
     5       .
     6        .       CALL PREAD(ITIME,ARRAY,ISTAT,$X,$Y)
     7       .
     8        . WHERE ITIME MUST BE INITIALIZED TO ZERO AND GETS INCREMENTED
     9        .             AFTER EACH CALL ON PREAD.
    10        .             IF NEGATIVE IT IS AS IF AN END OF FILE WAS FOUND.
    11        .       ARRAY MUST BE SIZE 14 & RETURNS THE IMAGE FROM THE ELEMENT.
    12        .       ISTAT RETURNS THE ERROR STATUS ON RETURN TO $X.
    13        .             =1 IF A PREPRM ERROR OCCURRED.
    14        .             =2 IF SOURCE INPUT ELEMENT CANNOT BE OPENED.
    15        .             =3 IF SOURCE INPUT OR UPDATING WRONG.
    16        .             =4 IF SOURCE OUTPUT ELEMENT CANNOT BE CLOSED OR
    17        .                HAS AN ERROR.
    18        .             =5 IF A POSTPR$ ERROR OCCURRED.
    19        .       FIRST STATEMENT RETURN ($X) IS THE STATEMENT NO. TO BE
    20        .             RETURNED TO IN CASE OF ERROR.
    21        .       SECOND STATEMENT RETURN ($Y) IS THE STATEMENT NO. TO BE
    22        .             RETURNED TO ON AN INPUT END OF FILE.
    23        .
    24                 AXR$                                .
    25       $(0),PARTBL* RES     28                       .
    26       REG         +        0                       .
    27       TEMP        +        0                       .
    28       $(1),PREAD*                                  .
    29                  SX       X6,REG                  .
    30                  LX       X6,X11                  .
    31                  LA       A0,*0,X6                .
    32                  AA,U     A0,1                    .
    33                  SA       A0,*0,X6                .
    34                  ANA,U    A0,1                    .
    35                  JZ       A0,INIT                 .
    36                  JP       A0,USUAL                .
    37                  J        EOF                     .
    38       INIT       LMJ      X11,PREPRM              .
    39                  J        ERR1                    .
    40                  LMJ      X11,OPNSR$              .
    41                  J        ERR2                    .
    42                  LA,U     A0,TEMP                 .
    43                  AA       A0,(01000000)           .
    44                  LMJ      X11,GETSR$              .
    45                  J        ERR3                    .
    46                  J        EOF                     .
    47       USUAL      LA       A0,1,X6                 .
    48                  LSSL     A0,18                   .
    49                  SSL      A0,18                   .
    50                  AA       A0,(016000000)          .
    51                  LMJ      X11,GETSR$              .
    52                  J        ERR3                    .
    53                  J        EOF                     .
    54                  SX       X6,X11                  .
    55                  LX       X6,RE6                  .
    56                  J        6,X11                   .
```

```
57        EOF     LMJ     X11,CLOSR$      .
58                J       ERR4            .
59                LMJ     X11,CLOSR$      .
60                J       ERR4            .
61                LMJ     X11,POSTPR      .
62                J       ERR5            .
63                SX      X6,X11          .
64                LX      X6,REG          .
65                EX      4,X11           .
66                J       ERR6            .
67        ERR1    LA,U    A0,1            .
68                J       ERROUT          .
69        ERR2    LA,U    A0,2            .
70                J       ERROUT          .
71        ERR3    LA,U    A0,3            .
72                J       ERROUT          .
73        ERR4    LA,U    A0,4            .
74                J       ERROUT          .
75        ERR5    LA,U    A0,5            .
76        ERROUT  SX      X6,X11          .
77                LX      X6,REG          .
78                SA      A0,*2,X11       .
79                EX      3,X11           .
80        ERR6    ER      ABORT$          .
81                END                     .
```

```
@PL/UCT  CALC
 PL/UCT 1.00-07/74
   B1                    1* CALC:    PROC OPTIONS(MAIN);
                        42   2*          DCL       A FIXED,
                        43   3*                    B,C FIXED BINARY,
                        45   4*                    D,E,F FLOAT,
                        48   5*                    G,H,I FLOAT DECIMAL,
                        51   6*                    J,K,L,M,N INITIAL(0),
                        56   7*                    A1(1..10) FLOAT,
                        66   8*                    CH(5) CHARACTER(6),
                        96   9*                    S1,S2(10,20) BINARY FIXED;
                       496  10*          DECLARE   1 ITEM,
                       496  11*                      2 X FLOAT ,
                       497  12*                      2 Y(10) FIXED BIN,
                       506  13*                      2 Z CHAR ;
                       508  14*          DECLARE SAME LIKE ITEM,
                       520  15*                    TARGET LIKE A;
                  40        16*  LAB1:   A = 2; D = 95.0; A1(A) = 29.86;
                  65        17*          CH(A+2) = 'HELLO ';
                  73        18*          L,M,N = -A;
                  80        19*          B = 200 + N*(N-1);
                  88        20*          C = A*8-40/N;
                 100        21*          E = D + 20.5 - A;
                 107        22*          F = A1(2*A-2) - A;
                 120        23*          S1(4,J*K+12-2*N) = C + 1;
                 139        24*  LAB2:   ITEM.Y(2) = 8;
                 145        25*          X = 16.0;
                 147        26*          F = F + X;
                 151        27*          TARGET = 85.0 + 5.0*E/A1(1+1);
                 166        28*          PUT LIST('TARGET = D + 5*E/A1(1+1)');
                 171        29*          PUT SKIP;
                 174        30*          PUT SKIP LIST('       = ',TARGET);
                 186        31*          PUT SKIP;
                 189        32*          PUT SKIP LIST('FOR  D = ',D,'   E = ',E,'   A1(2) = ',A1(2));
                 223        33*          PUT SKIP;


      E1                   34*          END CALC;

COMPILATION COMPLETE.      0 ERRORS WERE FOUND

COMPILATION TIME  56.76 SECONDS


TARGET = D + 5*E/A1(1+1)

       =          D

FOR  D =         .84999990+20,   E =         .10499999+21,   A1(2) =         .28999994+20

EXECUTION TIME  .   .11 SECONDS
```

```
            39  0          51 LDO    51
             9  0           0 FLO
            10  0           0 FLT
             7  0           0 DVR
             9  0           0 FLO
            22  0           0 SBR
            27  0           0 TRC
            43  0          43 SRO    43
100         39  0          47 LDO    47
            51  2           4 LDCR          .19999998+02
      .19999998+02
             3  0           0 ADR
            39  0          42 LDO    42
            10  0           0 FLT
            22  0           0 SBR
            43  0          46 SRO    46
            37  0          58 LAO    58
            51  1           2 LDCI             2
            39  0          42 LDO    42
110         15  0           0 MPI
            51  1           2 LDCI             2
            21  0           0 SBI
            31  0           1 DEC      1
            36  0           1 IXA      1
            35  0           0 IND      0
            39  0          42 LDO    42
            10  0           0 FLT
            22  0           0 SBR
            43  0          45 SRO    45
120         37  0         296 LAO   296
            51  1           4 LDCI             4
            31  0           1 DEC      1
            36  0          20 IXA     20
            39  0          55 LDO    55
            39  0          54 LDO    54
            15  0           0 MPI
            51  1          12 LDCI            12
             2  0           0 ADI
            51  1           2 LDCI             2
130         39  0          51 LDO    51
            15  0           0 MPI
            21  0           0 SBI
            31  0           1 DEC      1
            36  0           1 IXA      1
            39  0          43 LDO    43
            51  1           1 LDCI             1
             2  0           0 ADI
            26  0           0 STO
            37  0         497 LAO   497
140         51  1           2 LDCI             2
            31  0           1 DEC      1
            36  0           1 IXA      1
            39  0          44 LDO    44
            26  0           0 STO
            51  2           5 LDCR          .15999998+02
      .15999998+02
            43  0         496 SRO   496
```

```
                39  0          45 LDO    49
                39  0         496 LDO   496
                 3  0           0 ADR
    150         43  0          45 SRO    45
                51  2           6 LDCR          .84999990+02
          .84999990+02
                51  2           7 LDCR          .49999994+01
          .49999994+01
                39  0          46 LDO    46
                16  0           0 MPR
                37  0          56 LAO    56
                51  1           1 LDCI             1
                51  1           1 LDCI             1
                 2  0           0 ADI
                31  0           1 DEC     1
    160         36  0           1 IXA     1
                35  0           0 IND     0
                 7  0           0 DVR
                 3  0           0 ADR
                27  0           0 TRC
                43  0         520 SRO   520
                50  1          23 LDA     1    23
                38 24           8 LCA     1
TARGET = D + 5*E/A1(1+1)
                51  1          24 LDCI            24
                51  1          24 LDCI            24
    170         30  0          10 CSP WRS     0
                51  1           0 LDCI             0
                51  1           2 LDCI             2
                30  0           9 CSP WRC     0
                51  1           0 LDCI             0
                51  1           2 LDCI             2
                30  0           9 CSP WRC     0
                50  1          23 LDA     1    23
                38  9           9 LCA     1
      =
                51  1           9 LDCI             9
    180         51  1           9 LDCI             9
                30  0          10 CSP WRS     0
                50  1          23 LDA     1    23
                39  0         520 LDO   520
                51  1          10 LDCI            10
                30  0           6 CSP WRI     0
                51  1           0 LDCI             0
                51  1           2 LDCI             2
                30  0           9 CSP WRC     0
                51  1           0 LDCI             0
    190         51  1           2 LDCI             2
                30  0           9 CSP WRC     0
                50  1          23 LDA     1    23
                38  9          10 LCA     1
FOR  D =
                51  1           9 LDCI             9
                51  1           9 LDCI             9
                30  0          10 CSP WRS     0
                50  1          23 LDA     1    23
                39  0          47 LDO    47
```

```
                51  1          20 LDCI            20
    200         30  0           8 CSP WRR      0
                50  1          23 LDA      1     23
                38  8          11 LCA      1
*   E =
                51  1           8 LDCI             8
                51  1           8 LDCI             8
                30  0          10 CSP WRS      0
                50  1          23 LDA      1     23
                39  0          46 LDO     46
                51  1          20 LDCI            20
                30  0           8 CSP WRR      0
    210         50  1          23 LDA      1     23
                38 12          12 LCA      1
*   A1(2) =
                51  1          12 LDCI            12
                51  1          12 LDCI            12
                30  0          10 CSP WRS      0
                50  1          23 LDA      1     23
                37  0          56 LAO     56
                51  1           2 LDCI             2
                31  0           1 DEC      1
                36  0           1 IXA      1
    220         35  0           0 IND      0
                51  1          20 LDCI            20
                30  0           8 CSP WRR      0
                51  1           0 LDCI             0
                51  1           2 LDCI             2
                30  0           9 CSP WRC      0
                42  0          21 RETP
                32  0           0 ENT      0                              CALC
      0         41  0           0 MST      0
                46  0          40 CUP      0     40
                29  0           0 STP
                42  0           5 RET
```

```
@PL/OCT  POLY
 PL/OCT 1.00-07/74
   B1                    1* POLY:    PROC OPTIONS(MAIN);
                    42   2*          DCL       X,Y FLOAT,
                    44   3*                     K,N FIXED,
                    46   4*                     B(0..10) FLOAT;
                    57   5*     /*READ IN DEGREE OF POLYNOMIAL*/
              40         6*          GET LIST(N);    PUT SKIP;
   B2         51         7*          BEGIN
   B3         51         8*            DO K = 0 TO N;
              69         9*              GET LIST(B(K))
        E3    74        10*            END;
              77        11*            GET LIST(X);
              80        12*            Y = B(N);
   B4         85        13*            DO K = N-1 TO 0 BY -1;
             107        14*              Y = Y*X+B(K);
        E4   116        15*            END;
             119        16*            PUT SKIP LIST('VALUE OF POLYNOMIAL OF ',
             127        17*                          'DEGREE N = ',N);
   B5        136        18*            DO K = 0 TO N;
             154        19*              PUT SKIP LIST('B(',K,') = ',B(K));
        E5   178        20*            END;
             181        21*            PUT SKIP LIST('X = ',X,' Y = ',Y);
             202        22*            PUT SKIP;
        E2   205        23*          END;


        E1              24*          END POLY;

 COMPILATION COMPLETE.       0 ERRORS WERE FOUND

 COMPILATION TIME    35.43 SECONDS




 VALUE OF POLYNOMIAL OF DEGREE N =          4
 B(          0) =          .36900000+02
 B(          1) =          .12900000+02
 B(          2) =          .31774000+04
 B(          3) =          .54120000+02
 B(          4) =          .86000000+02
 X =          .72740000+02 Y =          .24452827+10

 EXECUTION TIME      .21 SECONDS
```

```
L1869     10/07-13:40:52
          40        32  0          56 ENT     56                              POLY
                    51  1           0 LDCI                 0
                    43  0           4 SRO      4
                    51  1           0 LDCI                 0
                    43  0          23 SRO     23
                    50  1           4 LDA      1       4
                    37  0          44 LAO     44
                    30  0           3 CSP RDI       0
                    51  1           0 LDCI                 0
                    51  1           2 LDCI                 2
          50        30  0           9 CSP WRC       0
                    51  1           0 LDCI                 0
                    43  0          45 SRO     45
                    39  0          44 LDO     44
                    43  0          58 SRO     58
                    51  1           1 LDCI                 1
                    43  0          57 SRO     57
                    57  0          69 UJP     69
                    39  0          45 LDO     45
                    34  0           1 INC      1
          60        43  0          45 SRO     45
                    39  0          45 LDO     45
                    39  0          58 LDO     58
                    52 14           1 LEGI
                    19  0           0 NOT
                    33  0          69 FJP     69
                    51  1           2 LDCI                 2
                    43  0          57 SRO     57
                    57  0          74 UJP     74
                    50  1           4 LDA      1       4
          70        37  0          46 LAO     46
                    39  0          45 LDO     45
                    36  0           1 IXA      1
                    30  0           4 CSP RDR       0
                    54  0          57 LDO      0     57
                    44  0          75 XJP     75
                    57  0          58 UJP     58
                    50  1           4 LDA      1       4
                    37  0          43 LAO     43
                    30  0           4 CSP RDR       0
          80        37  0          46 LAO     46
                    39  0          44 LDO     44
                    36  0           1 IXA      1
                    35  0           0 IND      0
                    43  0          42 SRO     42
                    39  0          44 LDO     44
                    51  1           1 LDCI                 1
                    21  0           0 SBI
                    43  0          45 SRO     45
                    51  1           0 LDCI                 0
          90        43  0          58 SRO     58
                    51  1           1 LDCI                 1
                    17  0           0 NGI
                    51  1           1 LDCI                 1
                    43  0          57 SRO     57
```

```
                57   0          107 UJP    107
                59   0           45 LDO     45
                31   0            1 DEC      1
                43   0           45 SRO     45
                39   0           45 LDO     45
    100         39   0           58 LDC     58
                48  14            1 GEQI
                19   0            0 NOT
                33   0          107 FJP    107
                51   1            2 LDCI               2
                43   0           57 SRO     57
                57   0          116 UJP    116
                39   0           42 LDO     42
                39   0           43 LDC     43
                16   0            0 MPR
    110         37   0           46 LAO     46
                39   0           45 LDO     45
                36   0            1 IXA      1
                35   0            0 IND      0
                 3   0            0 ADR
                43   0           42 SRO     42
                54   0           57 LOD      0     57
                44   0          117 XJP    117
                57   0           96 UJP     96
                51   1            0 LDCI               0
    120         51   1            2 LDCI               2
                30   0            9 CSP WRC      0
                50   1           23 LDA      1     23
                38  23            1 LCA      1
VALUE OF POLYNOMIAL OF
                51   1           23 LDCI              23
                51   1           23 LDCI              23
                30   0           10 CSP WRS      0
                50   1           23 LDA      1     23
                38  11            2 LCA      1
DEGREE N =
                51   1           11 LDCI              11
    130         51   1           11 LDCI              11
                30   0           10 CSP WRS      0
                50   1           23 LDA      1     23
                39   0           44 LDO     44
                51   1           10 LDCI              10
                30   0            6 CSP WRI      0
                51   1            0 LDCI               0
                43   0           45 SRO     45
                39   0           44 LDC     44
                43   0           58 SRO     58
    140         51   1            1 LDCI               1
                43   0           57 SRO     57
                57   0          154 UJP    154
                39   0           45 LDO     45
                34   0            1 INC      1
                43   0           45 SRO     45
                39   0           45 LDO     45
                39   0           58 LDO     58
                52  14            1 LEQI
                19   0            0 NOT
```

```
                150         33  0          154 FJP   154
                            51  1            2 LDCI            2
                            43  0           57 SRO    57
                            57  0          178 UJP   178
                            51  1            0 LDCI            0
                            51  1            2 LDCI            2
                            30  0            9 CSP WRC     0
                            50  1           23 LDA     1    23
                            38  2            3 LCA     1
B(
                            51  1            2 LDCI            2
                160         51  1            2 LDCI            2
                            30  0           10 CSP WRS     0
                            50  1           23 LDA     1    23
                            39  0           45 LDO    45
                            51  1           10 LDCI           10
                            30  0            6 CSP WRI     0
                            50  1           23 LDA     1    23
                            38  4            4 LCA     1
) =
                            51  1            4 LDCI            4
                            51  1            4 LDCI            4
                170         30  0           10 CSP WRS     0
                            50  1           23 LDA     1    23
                            37  0           46 LAO    46
                            39  0           45 LDO    45
                            36  0            1 IXA     1
                            35  0            0 IND     0
                            51  1           20 LDCI           20
                            30  0            8 CSP WRR     0
                            54  0           57 LOD     0    57
                            44  0          179 XJP   179
                180         57  0          143 UJP   143
                            51  1            0 LDCI            0
                            51  1            2 LDCI            2
                            30  0            9 CSP WRC     0
                            50  1           23 LDA     1    23
                            38  4            5 LCA     1
X =
                            51  1            4 LDCI            4
                            51  1            4 LDCI            4
                            30  0           10 CSP WRS     0
                            50  1           23 LDA     1    23
                190         39  0           43 LDO    43
                            51  1           20 LDCI           20
                            30  0            8 CSP WRR     0
                            50  1           23 LDA     1    23
                            38  5            6 LCA     1
Y =
                            51  1            5 LDCI            5
                            51  1            5 LDCI            5
                            30  0           10 CSP WRS     0
                            50  1           23 LDA     1    23
                            39  0           42 LDO    42
                200         51  1           20 LDCI           20
                            30  0            8 CSP WRR     0
                            51  1            0 LDCI            0
```

```
                51   1            2 LDCI              2
                30   0            9 CSP WRC      0
                42   0           21 RETP
                32   0            0 ENT     0                          POLY.
0               41   0            0 MST     0
                46   0           40 CUP     0    40
                29   0            0 STP
                42   0            5 RET
```

```
&PL/UCT  MAXIMUM
 PL/UCT 1.00-07/74
  B1                    1* MAXIMUM: PROC OPTIONS(MAIN);
                    42  2*          DECLARE   I,N FIXED BIN,
                    44  3*                    X(10),BIG FLOAT DECIMAL;
                    55  4*     /*READ IN ARRAY X AND FIND ITS MAXIMUM*/
              40        5*          GET LIST(N);
  B2          48        6*          DO I = 1 TO N;
              66        7*            GET LIST(X(I));
              72        8*            PUT LIST(X(I));
        E2    80        9*          END;
              83       10*          BIG = X(1);
  B3          89       11*          DO I = 2 TO N;
             107       12*            BIG = MAX(BIG,X(I));
        E3   115       13*          END;
             118       14*          PUT SKIP LIST('MAX VALUE IS ',BIG);
             130       15*          PUT SKIP;


        E1             16*          END MAXIMUM;

 COMPILATION COMPLETE.       0 ERRORS WERE FOUND

 COMPILATION TIME    25.63 SECONDS



         .76000000+02        .99999000+03        .12900000+02        .15380000+04        .64100000+02        .70000000+04
         .58800000+02        .56000100+01
 MAX VALUE IS  .70000000+04

 EXECUTION TIME      .24 SECONDS
```

```
RL1869      10/07-16:41:28
        40          32  0           56 ENT     56                          MAXIMUM
                    51  1            0 LDCI                0
                    43  0            4 SRO      4
                    51  1            0 LDCI                0
                    43  0           23 SRO     23
                    50  1            4 LDA      1       4
                    57  0           42 UJP     42
                    30  0            3 CSP RDI        0
                    51  1            1 LDCI                1
                    43  0           43 SRO     43
        50          39  0           42 LDO     42
                    43  0           56 SRO     56
                    51  1            1 LDCI                1
                    43  0           55 SRO     55
                    57  0           66 UJP     66
                    39  0           43 LDO     43
                    34  0            1 INC      1
                    43  0           43 SRO     43
                    39  0           43 LDO     43
                    39  0           56 LDO     56
        60          52 14            1 LLQI     .
                    19  0            0 NOT
                    33  0           66 FJP     66
                    51  1            2 LDCI                2
                    43  0           55 SRO     55
                    57  0           80 UJP     80
                    50  1            4 LDA      1       4
                    37  0           45 LAO     45
                    39  0           43 LDO     43
                    31  0            1 DEC      1
        70          36  0            1 IXA      1
                    30  0            4 CSP RDR        0
                    50  1           23 LDA      1      23
                    37  0           45 LAO     45
                    39  0           43 LDO     43
                    31  0            1 DEC      1
                    36  0            1 IXA      1
                    35  0            0 IND      0
                    51  1           20 LDCI                20
                    30  0            8 CSP WRR        0
        80          54  0           55 LOD      0      55
                    44  0           81 XJP     81
                    57  0           55 UJP     55
                    37  0           45 LAO     45
                    51  1            1 LDCI                1
                    31  0            1 DEC      1
                    36  0            1 IXA      1
                    35  0            0 IND      0
                    43  0           44 SRO     44
                    51  1            2 LDCI                2
        90          43  0           43 SRO     43
                    39  0           42 LDO     42
                    43  0           56 SRO     56
                    51  1            1 LDCI                1
                    43  0           55 SRO     55
```

```
                57  0          107 UJP   107
                39  0           43 LDC    43
                34  0            1 INC     1
                43  0           43 SRO    43
                39  0           43 LDO    43
    100         39  0           56 LDO    56
                52 14            1 LLQI
                19  0            0 NOT
                33  0          107 FJP   107
                51  1            2 LDCI              2
                43  0           55 SRO    55
                57  0          115 UJP   115
                39  0           44 LDO    44
                37  0           45 LAO    45
                59  0           43 LDO    43
    110         31  0            1 DEC     1
                36  0            1 IXA     1
                35  0            0 IND     0
                30  0           22 CSP MAX      0
                43  0           44 SRO    44
                54  0           55 LOD     0    55
                44  0          116 XJP   116
                57  0           96 UJP    96
                51  1            0 LDCI              0
                51  1            2 LDCI              2
    120         30  0            9 CSP WRC      0
                50  1           23 LDA     1    23
                38 13            1 LCA     1
MAX VALUE IS
                51  1           13 LDCI             13
                51  1           13 LDCI             13
                30  0           10 CSP WRS      0
                50  1           23 LDA     1    23
                39  0           44 LDO    44
                51  1           20 LDCI             20
                30  0            8 CSP WRR      0
    130         51  1            0 LDCI              0
                51  1            2 LDCI              2
                30  0            9 CSP WRC      0
                42  0           21 RETP
                32  0            0 ENT     0                          MAXIMUM
      0         41  0            0 MST     0
                46  0           40 CUP     0    40
                29  0            0 STP
                42  0            5 RET
```

```
@PL/UCT  SORT
 PL/UCT 1.00-07/74
   B1                    1* SORT:    PROCEDURE OPTIONS(MAIN);
                      42   2*          DECLARE   I,J,N FIXED,
                      45   3*                    X(10),T FIXED BINARY;
                      56   4*   /*BUBBLE SORT*/
                40          5*          GET LIST(N);
                48          6*          PUT LIST('ORIGINAL INPUT');
                53          7*          PUT SKIP;
   B2           56          8* READIN:  DO I = 1 TO N;
                74          9*          GET FILE(SYSIN) LIST(X(I));
                80         10*          PUT  LIST(X(I));
        E2      88         11*          END READIN;
                91         12*          PUT SKIP;
   B3           94         13*  DO1:     DO I = N-1 TO 1 BY -1;
   B4          116         14*  DO2:     DO J = 1 TO I;
               134         15*          IF X(J) > X(J+1) THEN
   B5          148         16*            BEGIN
               148         17*              T = X(J);
               154         18*              X(J) = X(J+1);
               166         19*              X(J+1) = T
        E5     172         20*            END;
        E4     174         21*          END DO2;    /*INNER LOOP*/
        E3     177         22*          END DO1;   /*OUTER LOOP*/
               180         23*          PUT SKIP LIST('ARRAY X SORTED');
               188         24*          PUT SKIP;
   B6          191         25* PRT:     DO I = 1 TO N;
               209         26*          PUT LIST(X(I));
        E6     217         27*          END PRT;
               220         28*          PUT SKIP LIST(' END OF BUBBLE SORT  ') ;
               228         29*          PUT SKIP;


        E1                 30*          END SORT;
  E

COMPILATION COMPLETE.       0 ERRORS WERE FOUND

COMPILATION TIME    42.77 SECONDS



ORIGINAL INPUT
      7585        34         10         99        600          4

ARRAY X SORTED
         4        10         34         99        600       7585
 END OF BUBBLE SORT

EXECUTION TIME      .34 SECONDS
```

```
S.
RL1869     10/07-18:42:25
        40          32  0          59 ENT     59                          SORT
                    51  1           0 LDCI                 0
                    43  0           4 SRO      4
                    51  1           0 LDCI                 0
                    43  0          23 SRO     23
                    50  1           4 LDA      1      4
                    37  0          42 LAC     42
                    30  0           3 CSP RDI       0
                    50  1          23 LDA      1     23
                    38 14           1 LCA      1
    ORIGINAL INPUT
        50          51  1          14 LDCI                14
                    51  1          14 LDCI                14
                    30  0          10 CSP WRS      0
                    51  1           0 LDCI                 0
                    51  1           2 LDCI                 2
                    30  0           9 CSP WRC      0
                    51  1           1 LDCI                 1
                    43  0          44 SRO     44
                    39  0          42 LDO     42
                    43  0          57 SRO     57
        60          51  1           1 LDCI                 1
                    43  0          56 SRO     56
                    57  0          74 UJP     74
                    39  0          44 LDO     44
                    34  0           1 INC      1
                    43  0          44 SRO     44
                    39  0          44 LDO     44
                    39  0          57 LDO     57
                    52 14           1 LEQI
                    19  0           0 NOT
        70          33  0          74 FJP     74
                    51  1           2 LDCI                 2
                    43  0          56 SRO     56
                    57  0          88 UJP     88
                    50  1           4 LDA      1      4
                    37  0          46 LAC     46
                    39  0          44 LDO     44
                    31  0           1 DEC      1
                    36  0           1 IXA      1
                    30  0           3 CSP RDI       0
        80          50  1          23 LDA      1     23
                    37  0          46 LAC     46
                    39  0          44 LDO     44
                    31  0           1 DEC      1
                    36  0           1 IXA      1
                    35  0           0 IND      0
                    51  1          10 LDCI                10
                    30  0           6 CSP WRI       0
                    54  0          56 LOD      0     56
                    44  0          89 XJP     89
        90          57  0          63 UJP     63
                    51  1           0 LDCI                 0
                    51  1           2 LDCI                 2
                    30  0           9 CSP WRC      0
```

```
                39  0          42 LDC    42
                51  1           1 LDCI              1
                21  0           0 SEI
                43  0          44 SRO    44
                51  1           1 LDCI              1
                43  0          57 SRO    57
100             51  1           1 LDCI              1
                17  0           0 NEI
                51  1           1 LDCI              1
                43  0          56 SRO    56
                57  0         116 UJP   116
                39  0          44 LDO    44
                31  0           1 DEC     1
                43  0          44 SRO    44
                39  0          44 LDC    44
                39  0          57 LDO    57
110             48 14           1 GEQI
                19  0           0 NOT
                33  0         116 FJP   116
                51  1           2 LDCI              2
                43  0          58 SRO    58
                57  0         177 UJP   177
                51  1           1 LDCI              1
                43  0          43 SRO    43
                39  0          44 LDO    44
                43  0          59 SRO    59
120             51  1           1 LDCI              1
                43  0          58 SRO    58
                57  0         134 UJP   134
                39  0          43 LDO    43
                34  0           1 INC     1
                43  0          43 SRO    43
                39  0          43 LDC    43
                39  0          59 LDO    59
                52 14           1 LEQI
                19  0           0 NOT
130             33  0         134 FJP   134
                51  1           2 LDCI              2
                43  0          58 SRO    58
                57  0         174 UJP   174
                37  0          46 LAC    46
                39  0          43 LDO    43
                31  0           1 DEC     1
                36  0           1 IXA     1
                35  0           0 IND     0
                37  0          46 LAO    46
140             39  0          43 LDC    43
                51  1           1 LDCI              1
                 2  0           0 ADI
                31  0           1 DEC     1
                36  0           1 IXA     1
                35  0           0 IND     0
                49 14           1 GRTI
                33  0         174 FJP   174
                37  0          46 LAC    46
                39  0          43 LDO    43
150             31  0           1 DEC     1
```

```
                36  0          1 IXA     1
                35  0          0 IND     0
                43  0         45 SRC    45
                37  0         46 LAO    46
                39  0         43 LDO    43
                31  0          1 DEC     1
                36  0          1 IXA     1
                37  0         46 LAO    46
                39  0         43 LDO    43
    160         51  1          1 LDCI              1
                 2  0          0 ADI
                31  0          1 DEC     1
                36  0          1 IXA     1
                35  0          0 IND     0
                26  0          0 STO
                37  0         46 LAO    46
                39  0         43 LDO    43
                51  1          1 LDCI              1
                 2  0          0 ADI
    170         31  0          1 DEC     1
                36  0          1 IXA     1
                39  0         45 LDO    45
                26  0          0 STO
                54  0         58 LOD     0    58
                44  0        175 XJP   175
                57  0        123 UJP   123
                54  0         56 LOD     0    56
                44  0        178 XJP   178
                57  0        195 UJP   195
    180         51  1          0 LDCI              0
                51  1          2 LDCI              2
                30  0          9 CSP WRC     0
                50  1         23 LDA     1    23
                38 14          2 LCA     1
ARRAY X SORTED
                51  1         14 LDCI             14
                51  1         14 LDCI             14
                30  0         10 CSP WRS     0
                51  1          0 LDCI              0
                51  1          2 LDCI              2
    190         30  0          9 CSP WRC     0
                51  1          1 LDCI              1
                43  0         44 SRC    44
                39  0         42 LDO    42
                43  0         57 SRC    57
                51  1          1 LDCI              1
                43  0         56 SRC    56
                57  0        209 UJP   209
                39  0         44 LDO    44
                34  0          1 INC     1
    200         43  0         44 SRC    44
                39  0         44 LDO    44
                39  0         57 LDO    57
                52 14          1 LLQI
                19  0          0 NOT
                33  0        209 FJP   209
                51  1          2 LDCI              2
```

```
                43  0          58 SRO     58
                57  0         217 UJP    217
                50  1          23 LDA      1     23
    210         37  0          46 LAO     46
                39  0          44 LDO     44
                31  0           1 DLC      1
                36  0           1 IXA      1
                55  0           0 IND      0
                51  1          10 LDCI              10
                50  0           0 CSP WRI       0
                54  0          56 LOD      0     56
                44  0         218 XJP    218
                57  0         198 UJP    198
    220         51  1           0 LDCI               0
                51  1           2 LDCI               2
                30  0           9 CSP WRO       0
                50  1          23 LDA      1     23
                38 21           3 LOA      1
END OF BUBBLE SORT
                51  1          21 LDCI              21
                51  1          21 LDCI              21
                36  0          10 CSP WRS       0
                51  1           0 LDCI               0
                51  1           2 LDCI               2
    230         30  0           9 CSP WRO       0
                42  0          21 RLTP
                32  0           0 ENT      0                          SORT
      0         41  0           0 MST      0
                46  0          40 CUP      0     40
                29  0           0 STP
                42  0           5 RET
```

```
#PL/UCT  FACTORIAL
 PL/UCT 1.00-07/74
   B1                    1* FACTORIAL: PROCEDURE OPTIONS(MAIN);
                      42   2*           DCL       (N,F) FIXED BIN;
                      44   3*           DCL       FACT ENTRY(FIXED BIN) RETURNS(FIXED BIN);
   B2                      4* FACT:     PROC (M) RETURNS (FIXED BIN);
                        5   5*             DCL       (I,ANS) FIXED BINARY;
                 40         6*             IF M < 2 THEN ANS = 1;
                 48         7*               ELSE
   B3            49         8*                 BEGIN I = M - 1;
                 54         9*                   ANS = M*FACT(I)
        E3       58        10*                 END;
                 61        11*             RETURN (ANS);
        E2                 12*             END FACT;
                 65        13*           GET LIST (N);
                 73        14*           F = FACT(N);
                 77        15*           PUT SKIP LIST(F);
                 84        16*           PUT LIST(' IS THE FACTORIAL OF ',N);
                 93        17*           PUT PAGE;


        E1                 18*           END FACTORIAL;

 COMPILATION COMPLETE.       0 ERRORS WERE FOUND

 COMPILATION TIME   23.84 SECONDS



       120 IS THE FACTORIAL OF          5
```

3.

```
RL1869      10/07-16:43:43
          40          32  0            6 ENT      0                          FACT
                      54  0            4 LOD      0      4
                      35  0            0 IND      0
                      51  1            2 LDCI                2
                      53 14            1 LLSI
                      33  0           49 FJP     49
                      51  1            1 LDCI                1
                      56  0            5 STR      0      5
                      57  0           61 UJP     61
                      54  0            4 LOD      0      4
          50          35  0            0 IND      0
                      51  1            1 LDCI                1
                      21  0            0 SBI
                      56  0            6 STR      0      6
                      54  0            4 LOD      0      4
                      35  0            0 IND      0
                      41  0            1 MST      1
                      50  0            6 LDA      0      6
                      46  1           40 CUP      1     40
                      15  0            0 MPI
          60          56  0            5 STR      0      5
                      54  0            5 LOD      0      5
                      56  0            0 STR      0      0
                      57  0           64 UJP     64
                      42  0           11 RETF
                      32  0           43 ENT     43                          FACTORIAL
                      51  1            0 LDCI                0
                      43  0            4 SRO      4
                      51  1            0 LDCI                0
                      43  0           23 SRO     23
          70          50  1            4 LDA      1      4
                      37  0           43 LAO     43
                      30  0            3 CSP RDI       0
                      41 -1            1 MST      1
                      37  0           43 LAO     43
                      46  1           40 CUP      1     40
                      43  0           42 SRO     42
                      51  1            0 LDCI                0
                      51  1            2 LDCI                2
                      30  0            9 CSP WRC       0
          80          50  1           23 LDA      1     23
                      39  0           42 LDO     42
                      51  1           10 LDCI               10
                      30  0            6 CSP WRI       0
                      50  1           23 LDA      1     23
                      38 21            1 LCA      1
IS THE FACTORIAL OF
                      51  1           21 LDCI               21
                      51  1           21 LDCI               21
                      30  0           10 CSP WRS       0
                      50  1           23 LDA      1     23
          90          39  0           43 LDO     43
                      51  1           10 LDCI               10
                      30  0            6 CSP WRI       0
                      51  1            0 LDCI                0
```

```
                51   1            3 LUCI              3
                30   0            9 CSP WRC      0
                42   0           21 RETP
                32   0            0 ENT      0                        FACTORIAL
          0     41   0            0 MST      0
                46   0           65 CUP      0     65
                29   0            0 STP
                42   0            5 RET
```

```
@PL/UCT  PROCS
 PL/UCT 1.00-07/74
  B1                    1* PROCS: PROCEDURE OPTIONS(MAIN);
                   42   2*         DCL       A,B,C,D,E;
                   47   3*         DCL       PEG1,PEG2,PEG3 CHAR,
                   50   4*                   DISCS FIXED INIT(4);
                   51   5*         DCL       P1 ENTRY(FIXED,CHAR,CHAR,CHAR);
                   51   6*         DCL       P2 ENTRY(FLOAT,FLOAT);
                   51   7*         DCL       F1 ENTRY(FLOAT) RETURNS(FLOAT);
  B2                    8*  P1:    PROC(N,A,B,C);
                    8   9*           DCL     M FIXED;
  B3          40       10*           IF N > 0 THEN DO;
              46       11*               M = N - 1;
              51       12*               CALL P1(M,A,C,B);
              57       13*               PUT SKIP LIST('MOVE PIECE',N,' FROM ',A,' TO ',C);
              90       14*               CALL P1(M,B,A,C);
       E3     96       15*             END;
       E2              16*           END P1;
  B4                   17*  P2:    PROC(F,G);
                    6  18*           DCL       P3 ENTRY(FLOAT);
                    6  19*           DCL       FLOCAL,GLOCAL,D,E;
  B5                   20*      P3:    PROC(D);
                    5  21*             DCL DLOCAL,E;
  B6          97       22*             DO;
              98       23*              E := D;
             101       24*              FLOCAL := E;
             103       25*              E := A;
       E6    105       26*             END;
       E5              27*           END P3;
             106       28*           E := G;
             110       29*           CALL P3(E);
       E4              30*         END P2;
  B7                   31*  F1:    PROC(A) RETURNS(FLOAT);
             114       32*           RETURN(A+1.0);
       E7              33*         END F1;
             122       34*         PUT PAGE;
             132       35*         PEG1 = 'A'; PEG2 = 'B'; PEG3 = 'C';
             138       36*         CALL P1(DISCS,PEG1,PEG2,PEG3);
             144       37*         PUT SKIP;
             147       38*         A = 100.0; B = 20.0;
             151       39*         CALL P2(A,B);
             155       40*         C := F1(B);


       E1              41*         END PROCS;

 COMPILATION COMPLETE.       0 ERRORS WERE FOUND

 COMPILATION TIME    53.01 SECONDS
```

```
MOVE PIECE          1 FROM A TO B
MOVE PIECE          2 FROM A TO C
MOVE PIECE          1 FROM B TO C
MOVE PIECE          3 FROM A TO B
MOVE PIECE          1 FROM C TO A
MOVE PIECE          2 FROM C TO B
MOVE PIECE          1 FROM A TO B
MOVE PIECE          4 FROM A TO C
MOVE PIECE          1 FROM B TO C
MOVE PIECE          2 FROM B TO A
MOVE PIECE          1 FROM C TO A
MOVE PIECE          3 FROM B TO C
MOVE PIECE          1 FROM A TO B
MOVE PIECE          2 FROM A TO C
MOVE PIECE          1 FROM B TO C

EXECUTION TIME      .41 SECONDS
```

```
869     10/07-16:44:12
        40          32  0              8 ENT      8                          P1
                    54  0              4 LOD      0      4
                    35  0              0 IND      0
                    51  1              0 LDCI                 0
                    49 14              1 ORTI
                    53  0             96 FJP     96
                    54  0              4 LOD      0      4
                    35  0              0 IND      0
                    51  1              1 LDCI                 1
                    21  0              0 SBI
        50          56  0              8 STR      0      8
                    41  0              0 MST      0
                    50  0              8 LDA      0      8
                    54  0              5 LOD      0      5
                    54  0              7 LOD      0      7
                    54  0              6 LOD      0      6
                    46  4             40 CUP      4     40
                    51  1              0 LDCI                 0
                    51  1              2 LDCI                 2
                    30  0              9 CSP WRC         0
        60          50  2             23 LDA      2     23
                    38 10              1 LCA      2
MOVE PIECE
                    51  1             10 LDCI                10
                    51  1             10 LDCI                10
                    30  0             10 CSP WRS         0
                    50  2             23 LDA      2     23
                    54  0              4 LOD      0      4
                    35  0              0 IND      0
                    51  1             10 LDCI                10
                    30  0              6 CSP WRI         0
        70          50  2             23 LDA      2     23
                    38  8              2 LCA      2
FROM
                    51  1              6 LDCI                 6
                    51  1              6 LDCI                 6
                    30  0             10 CSP WRS         0
                    50  2             23 LDA      2     23
                    54  0              5 LOD      0      5
                    35  0              0 IND      0
                    51  1              1 LDCI                 1
                    30  0              9 CSP WRC         0
        80          50  2             23 LDA      2     23
                    38  4              3 LCA      2
TO
                    51  1              4 LDCI                 4
                    51  1              4 LDCI                 4
                    30  0             10 CSP WRS         0
                    50  2             23 LDA      2     23
                    54  0              7 LOD      0      7
                    35  0              0 IND      0
                    51  1              1 LDCI                 1
                    30  0              9 CSP WRC         0
        90          41  0              0 MST      0
                    50  0              8 LDA      0      8
```

```
                54   6          5 LOD      3       6
                54   0          5 LOD      0       5
                54   6          7 LOD      6       7
                46   4         40 CUP      4      40
                42   0         21 RETP
                32   0          6 ENT      6                          P3
                54   0          4 LOD      0       4
                35   0          0 IND      0
100             56   6          5 STR      3       5
                54   3          5 LOD      0       5
                56   1          9 STR      1       9
                39   0         46 LDO     46
                56   0          5 STR      0       5
                42   0         21 RETP
                32   0          9 ENT      9                          P2
                54   6          5 LOD      6       5
                35   0          0 IND      0
                56   0          6 STR      0       6
110             41  -1          0 MST      0
                50   0          6 LDA      6       6
                46   1         97 CUP      1      97
                42   0         21 RETP
                32   0          4 ENT      4                          F1
                54   0          4 LOD      0       4
                35   0          0 IND      0
                51   2          1 LDCR                .99999988+00
        .99999988+00
                 3   0          0 ADR
                56   0          0 STR      0       0
120             57   0        121 UJP    121
                42   0         11 RETF
                32   0         50 ENT     50                          PROCS
                51   1          0 LDCI                0
                43   0          4 SRO      4
                51   1          0 LDCI                0
                43   0         23 SRO     23
                51   1          4 LDCI                4
                43   0         50 SRO     50
                51   1          0 LDCI                0
130             51   1          3 LDCI                3
                30   0          9 CSP WRC     0
                51   1          6 LDCI                6
                43   0         49 SRO     49
                51   1          7 LDCI                7
                43   0         48 SRO     48
                51   1          8 LDCI                8
                43   0         47 SRO     47
                41  -1          0 MST      0
                37   0         50 LAC     50
140             37   0         49 LAO     48
                37   0         48 LAO     48
                37   0         47 LAO     47
                46   4         40 CUP      4      40
                51   1          0 LDCI                0
                51   1          2 LDCI                2
                30   0          9 CSP WRC     0
                51   2          1 LDCR                .99999988+02
```

```
      .99999998+02
            43  0          46 SRO    46
            51  2           2 LOCR          .19999998+02
      .19999998+02
154         43  0          45 SRO    45
            41 -1           0 MST     0
            37  0          46 LAC    46
            37  0          45 LAD    45
            46  2         106 CUP     2   106
            41 -1           1 MST     1
            37  0          45 LAC    45
            46  1         114 CUP     1   114
            43  0          44 SRO    44
            42  0          21 RETP
            32  0           0 ENT     0                          PROCS
            41  0           0 MST     0
            46  0         122 CUP     0   122
            29  0           0 STP
            42  0           5 RET
```