

DESIGN AND IMPLEMENTATION OF A SIMPLE
TABLE DRIVEN COMPILER

by

16
349 5839

ROBERT DOUGLAS CRANK

B.S. in Computer Science, Kansas State University, 1971

A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

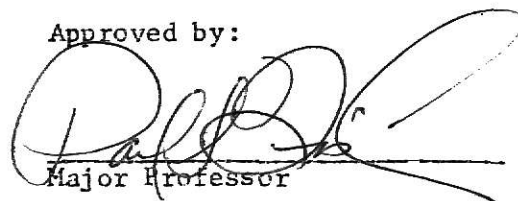
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1973

Approved by:



Major Professor

LD
2668
R4
1973
C73
C.2
Document

TABLE OF CONTENTS

	Page
LIST OF TABLES.....	iii
LIST OF FIGURES.....	iv
Chapter	
1. INTRODUCTION.....	1
2. REDUCTIONS ANALYSIS.....	4
3. REDUCTIONS TABLE GENERATOR.....	11
4. BASIC DATA STRUCTURES.....	18
5. COMPILER STRUCTURE.....	31
6. SUMMARY.....	40
BIBLIOGRAPHY.....	44
APPENDIXES.....	
I Language Definition.....	45
II Left Hand Sides.....	50
III Reductions Table.....	53
IV Generated Code.....	89
V Run Time Routines.....	96
VI Compiler Listing.....	98
VII Reductions Program.....	172

LIST OF TABLES

Table	Page
2.1 Sample Reductions Table.....	8
3.1 Reductions Table Segment.....	12

LIST OF FIGURES

Figure	Page
4.1 Simplified Symbol Table Structure.....	19
4.2 Left Hand Side Table Structure.....	24
4.3 Stack Structure Before Reduction.....	26
4.4 Stack Structure After Reduction.....	28

Chapter 1

INTRODUCTION

This report is concerned with the design and implementation of a small single pass compiler for a simple block structured language. This compiler is designed to accept as input a subset of PL/I, PLAN (Programming Language), with enough features to be able to compile itself, and generate as output assembler mnemonics for a Data General Nova mini-computer. The language necessarily includes string handling capabilities, but also includes array structures, elementary I/O commands, and single and double precision arithmetic. Design of the compiler allows simple changes in the syntax of the language to be made with relative ease.

The compiler has been divided into two program segments. The first, PGM1, is necessary to define the syntax of PLAN and generate the tables necessary to drive the second. PGM1 accepts as input a list of keywords along with a coded version of the rules which define the language. Syntactic values are associated with each keyword and these are reflected in the rewrite rules which define the structure of the language.

The second segment, PGM2, is driven by the data generated by PGM1. It reads in strings of text representing an algorithm written in PLAN and generates the appropriate Nova assembler code. PGM2 maintains all keywords and left hand sets along with user defined symbols in a stack symbol table to allow for the block structure allowed in PLAN.

The approach taken has been to first select a suitable subset of PL/I which was thought to be sufficient to write a program of this type. This subset has some minor additions not contained within PL/I. The necessary tables were then designed to hold the semantic information associated with the syntactic structures defined within the language. It was then possible to write PGM1 to process a coded form of the language definition and generate the information necessary to initialize the tables of PGM2. Knowing what semantic information had to be kept for a given construct and how input strings in the language were to be parsed, it was possible to design and write the semantic support routines, which are called from various semantic routines any time some table processing or manipulation has to be done, such as entering or leaving a block. Finally, the semantic routines themselves were designed and written. By delaying the actual writing of the semantic routines, any changes to the language which were needed in writing the rest of the compiler could be incorporated with a minimum amount of effort.

PLAN is defined by a context free grammar acceptable to a deterministic bottom-up parsing scheme. As a member of the class of $LR_{(1)}$ grammars, parsing consists of scanning input symbols left to right, shifting symbols onto a push-down stack until a handle appears on top of the stack. If no errors occur when this handle is reduced, the process continues until the input has been scanned and the sentential symbol remains. Each reduction of the handle may cause code to be generated.

The recognition algorithm is table driven and easily modified to reflect input language changes. However, code generation is very much embedded in the semantic routines and not so easily modified. Generated code consists simply of a string of system calls to run-time routines, each time passing the appropriate arguments and addresses. Not only is this a great deal easier than generating in-line code, but within any reasonably large program less core will be required in the end.

The recognition algorithm, a Floyd Reductions analysis scheme, is described and illustrated in Chapter II. Chapter III discusses the structured table used to drive the recognition algorithm. This is the table generated from the data supplied to PGML and structured in such a way as to allow the algorithm to function smoothly. Chapter IV describes the compiler data structures which hold the most important information, such as the symbol table and the reductions table. Finally, the compiler itself is described in Chapter V, with a simplified overall view of its logic.

Chapter 2

REDUCTIONS ANALYSIS

A number of syntactic analyzer routines are in common use today, each with its advantages and disadvantages. The syntax checker chosen for this compiler is referred to as a reductions analysis routine because its function is to methodically reduce, or shorten, input streams to some limiting value. The success or failure of these reductions provides information about what code to generate, or what errors were made in the input. The power of this method is derived from several points:

- 1.) It is easy to change the syntax of the language.
- 2.) It is generally faster than other syntactic analyzers.
- 3.) There is no backtracking necessary on input streams.
- 4.) It provides for modular program logic.

Reductions are determined by successfully matching patterns of input symbols to similar patterns of symbols in a reductions table, which can be constructed deterministically from the language specifications. This is a very straight forward idea, but with even a moderately sized language the number of reductions generated can be quite large. The purpose of PGML is to generate that table.

As an example of the method, let us define a simple language and derive the reductions from it. First we define the set of nonterminal symbols

$$S_n = \{S, E, T, F, P, V\}$$

and the set of terminal symbols (those which may appear in the input stream)

$$S_t = \{=, +, *, **, 1, 2, 3, A, B, C\}.$$

The language may now be defined in terms of a series of rules written in these symbols. Let a set of these rules be defined as follows:

- 1.) $\langle S \rangle :: = \langle V \rangle = \langle E \rangle$
- 2.) $\langle E \rangle :: = \langle T \rangle$
- 3.) $\langle E \rangle :: = +\langle T \rangle$
- 4.) $\langle E \rangle :: = \langle E \rangle + \langle T \rangle$
- 5.) $\langle T \rangle :: = \langle F \rangle$
- 6.) $\langle T \rangle :: = \langle T \rangle * \langle F \rangle$
- 7.) $\langle F \rangle :: = \langle P \rangle$
- 8.) $\langle F \rangle :: = \langle F \rangle ** \langle P \rangle$
- 9.) $\langle P \rangle :: = 1$
- 10.) $\langle P \rangle :: = 2$
- 11.) $\langle P \rangle :: = 3$
- 12.) $\langle V \rangle :: = A$
- 13.) $\langle V \rangle :: = B$
- 14.) $\langle V \rangle :: = C$

Within this notation, nonterminals appear with pointed braces '<' and '>' surrounding them. The symbol '::=' is read "is defined as". Also, recursive rewrite rules are permitted, allowing expressions to be generated of indefinite length.

These "rewrite rules" are called productions and can be used to determine if a given string of input symbols is a valid construct in the language as

follows. If, starting with the sentential symbol S , (defined by production #1), one can arrive at the proper sequence of terminal symbols which matches exactly that input string being tested, then the input is syntactically valid. If it cannot be reached, then it is invalid. For example, the string

$$A = 2 ** 3$$

is valid because of the following derivation.

$\langle S \rangle :: = \langle V \rangle = \langle E \rangle$	(Rule 1)
$\langle S \rangle :: = A = \langle E \rangle$	(Rule 12)
$\langle S \rangle :: = A = \langle T \rangle$	(Rule 2)
$\langle S \rangle :: = A = \langle F \rangle$	(Rule 5)
$\langle S \rangle :: = A = \langle F \rangle ** \langle P \rangle$	(Rule 8)
$\langle S \rangle :: = A = \langle P \rangle ** \langle P \rangle$	(Rule 7)
$\langle S \rangle :: = A = 2 ** 3$	(Rules 10, 11)

Other examples of valid strings are

$$B = 3*2**2$$

$$C = 1+2+3+3*2**1$$

The reductions method starts with the right hand sides of the productions and compares them to the input. If the comparison is successful, the part of the input stream which matches is reduced, or replaced, by the left hand side of the production. Successive replacements eventually reduce the string to the sentential symbol S . To determine which part of the input we must try to reduce, a pointer is defined on the input stream. Successive input symbols are included on the parse stack as the

pointer moves from left to right. For each symbol appearing immediately to the left of the pointer one or more matches may be attempted. If the necessary number of symbols left of the pointer match the part of the reductions table tested, then either a reduction is effected, or a new symbol is read by moving the pointer one place to the right. Otherwise, an error condition exists and the process stops. If more than one comparison may be made for a single left-of-pointer symbol, then the most restrictive patterns must be tested first. The unordered table for the previous example is given in table 2.1.

The reductions table 2.1 is interpreted as follows. The symbols along the left side are labels, or entry points. If the symbol at the top of the parse stack is a "+", for example, then that part of the table labelled with the "+" is considered. (Rules #14, 15). If this table were correctly ordered, rule #15 would precede rule #14 because, intuitively, rule #15 is a more restrictive rule than rule #14. That is, rule #14 is always satisfied if rule #15 is, but not vice versa. Therefore, we should attempt rule #15 first.

If we have on our parse stack a "+" preceded by the nonterminal metasympol "E" then we go to "ENTRY", or in effect go to read another symbol from the input stream. Otherwise if we have on the parse stack a "+" with no restrictions on what may precede it, then rule #14 tells us to go to "ENTRY", or again read another symbol. No other possibilities need to be tested because rule #14 will always be satisfied. The prerequisite to testing at this part of the table in the first place was only that a "+" was the immediate left-of-pointer symbol. Both rules 14 and 15 specify

<u>LEFT-OF-POINTER SYMBOL</u>	<u>RULE NUMBER</u>	<u>PARSE STACK</u>
E	1	$\langle V \rangle = \langle E \rangle_{\Delta} \rightarrow S$
	2	$\langle E \rangle_{\Delta} \rightarrow \text{ENTRY}$
I	3	$\langle T \rangle_{\Delta} \rightarrow E$
	4	$+ \langle T \rangle_{\Delta} \rightarrow E$
	5	$\langle E \rangle + \langle T \rangle_{\Delta} \rightarrow E$
	6	$\langle T \rangle_{\Delta} \rightarrow \text{ENTRY}$
F	7	$\langle F \rangle_{\Delta} \rightarrow T$
	8	$\langle T \rangle * \langle F \rangle_{\Delta} \rightarrow T$
	9	$\langle F \rangle_{\Delta} \rightarrow \text{ENTRY}$
P	10	$\langle P \rangle_{\Delta} \rightarrow F$
	11	$\langle F \rangle ** \langle P \rangle_{\Delta} \rightarrow F$
V	12	$\langle V \rangle_{\Delta} \rightarrow \text{ENTRY}$
=	13	$\langle V \rangle = \Delta \langle E \rangle \rightarrow \text{ENTRY}$
+	14	$+ \Delta \langle T \rangle \rightarrow \text{ENTRY}$
		$\langle E \rangle + \Delta \langle T \rangle \rightarrow \text{ENTRY}$
*	16	$\langle T \rangle * \Delta \langle F \rangle \rightarrow \text{ENTRY}$
**	17	$F ** \Delta \langle P \rangle \rightarrow \text{ENTRY}$
1	18	$1_{\Delta} \rightarrow P$
2	19	$2_{\Delta} \rightarrow P$
3	20	$3_{\Delta} \rightarrow P$
A	21	$A_{\Delta} \rightarrow V$
B	22	$B_{\Delta} \rightarrow V$
C	23	$C_{\Delta} \rightarrow V$

TABLE 2.1

a success action of reading a new input symbol. This new symbol in turn determines which part of the table (i.e., which rules) will be tested next.

Let us follow an example through the entire reductions process. Let the string to be reduced be ' $\Delta A = 2**3$ ' (where Δ is the pointer on the parse stack), and assume the table has been reordered to test patterns in the proper order.

<u>STEP</u>	<u>STRING</u>	<u>SUCCESSFUL RULE</u>	<u>NEXT RULE</u>
1	$\Delta A = 2**3$		ENTRY
2	$A_{\Delta} = 2**3$	21	12
3	$V_{\Delta} = 2**3$	12	ENTRY
4	$V =_{\Delta} 2**3$	ENTRY	ENTRY
5	$V - 2_{\Delta} **3$	ENTRY	19
6	$V = P_{\Delta} **3$	19	10
7	$V = F_{\Delta} **3$	10	ENTRY
8	$V = F**_{\Delta} 3$	ENTRY	ENTRY
9	$V = F**3_{\Delta}$	ENTRY	20
10	$V = F**P_{\Delta}$	20	11
11	$V = F_{\Delta}$	11	7
12	$V = T_{\Delta}$	7	3
13	$V = E_{\Delta}$	3	1
14	S_{Δ}	1	END

Since this string eventually reduced to (); the sentential symbol 'S' then it is a string in the language defined by the set of rules.

We may distinguish between reductions "programs" which do and do not check the character to the right of the delta pointer. Those that do are referred to as "reductions with context verification:", such as the previous example. Others are referred to as "reductions without context verification".

Chapter 3

REDUCTIONS TABLE GENERATOR

It is the responsibility of the first program, PGM1, to generate the data used to initialize certain tables in the compiler (PGM2) itself. It does this from input supplied by the user. Several things must be input, all of which contribute directly to defining either the syntax or the semantics of the language. First, the terminal symbols of the language which make up the set of keywords is supplied along with an arbitrarily assigned syntactic value. This is the value by which that keyword, or token, is numerically coded in the set of productions. This set of productions is input as a coded array and defines the syntax of all constructs which the user wishes the compiler to recognize. With each of these productions, or rewrite rules, is associated a unique, but not necessarily sequential, positive integer with which to identify that rule.

A character representation of nonterminal elements of the language is also input, but its only purpose is to improve the readability of printed output from this program.

Optionally, the user may also input a set of semantic requests which specify that when a particular sequence of symbols appears on the reductions stack during the scan, a particular user defined action is to occur, rather than the default action normally taken for successes and failures. Quite simply, this is the table which drives the compiler, PGM2. Any changes to the definition of the language are reflected in

ILLEGIBLE DOCUMENT

**THE FOLLOWING
DOCUMENT(S) IS OF
POOR LEGIBILITY IN
THE ORIGINAL**

**THIS IS THE BEST
COPY AVAILABLE**

this table, which causes the correspondingly different actions to occur in the compiler as input is processed.

This reductions table is generated as follows. For each symbol of the grammar, either terminal or nonterminal, collect all those productions which contain that symbol in their right hand side. Denote that symbol by S' . The right hand sides of these rules are then sorted first by the syntactic value of the single symbol to the right of S' (if one exists), and secondly by decreasing number of symbols appearing left of S' . This will cause more restrictive rules to be entered, and thus tested, before less restrictive rules. The double sort is necessary to allow right context verification. Finally, entries are made in the reductions table for each right hand side symbol, of those rules collected, left of, or one position to the right of, S' . Following is a segment from one such table showing how it is structured.

<u>LINE #</u>	<u>POSITION & MATCH</u>	<u>SUCCESS</u>	<u>FAILURE</u>
115	L1 CALL	0	116
116	L1 LABEL	0	117
117	L1 STOP	0	118
118	L2 < IF CLAUSE >	119	121
119	L1 < STATEMENT >	120	121
120	R1 ELSE	0	121
121	L2 < STATEMENT LIST >	122	124
122	L1 < STATEMENT >	-130	124
123	L0 < ANY >	2	6

REDUCTIONS TABLE SEGMENT

TABLE 3.1

The line number is not part of the table but serves only to identify the row of the table. This table, the reductions table, is a two dimensional table where each row represents a single test (and action) to be made, and the number of rows depends directly on the formal definition of the language and the rewrite rules used to define it. Reductions of the input stream take place as directed by the information in this table. A row may specify that a test, or comparison, is to be made at some position on the reductions stack for a particular symbol, and specify for both success and failure that one of two actions is to take place. Usually, the normal action is to jump to another row of the table and make another comparison. Alternatively, control may be transferred out of the table and into an indicated semantic routine.

The first element of the row specifies both the position on the stack which is to be considered, and the symbol which is to be compared against that position. For the purposes of illustration, in table 3.1, an L1 represents the first symbol to the left of the scan pointer, or equivalently, the top position on the stack; L2 represents the second, and so on. An R1 represents the next token yet to bring into the scan, or the right context character.

The second element of a row contains information pertaining to the action to take if the comparison is successful, and the third is similarly defined for a comparison which returns a failure indication. If the value of the required (second or third) element is zero, this simply indicates that the next element of the input stream is to be brought into the scan and the first row of the table is the next to be considered. If the

value of the required element is greater than zero, then no changes take place in the stack and that value defines the next row to be considered. Finally, if the indicated element is less than zero, then this value directly indicates a semantic routine which is to receive control, presumably to effect a reduction on the stack.

The first part of the table is referenced to determine where to jump into the middle of the table and proceed. For example, immediately after a reduction takes place, a new symbol has replaced one or more previous symbols on the stack. To continue processing, the block of comparisons relevant to that top symbol must be found, and rather than search the entire table, that section is determined by beginning at the first row and sequentially proceeding row by row until a success jump is taken to the correct entry point.

Examining table 3.1, we may determine the following. Line 115 says compare the first position on the stack (L1) for the keyword "CALL" and on a match bring in the next symbol of the input stream and on a failure go to line 116 and continue. Line 116 compares the first position for the keyword "LABEL" and proceeds to line 117 on a failure. In exactly the same way line 117 compares the top position on the stack for the keyword "STOP". The first 117 rows of the table are those which are searched sequentially to determine an entry point farther into the table for a new top-of-stack symbol, and presumably contain a successful match for every symbol which is expected to appear on the stack. Although lines 115 through 117 simply indicate that the next

symbol is to be entered upon a successful match, some of the previous rows will point to succeeding rows of the table.

Continuing in the table, line 118 is the first row indicating a test made for a construct on the stack corresponding to a reductions rule. If the second position (L2) on the stack contains the syntactic value for an "IF CLAUSE" then jump to line 119, otherwise jump to line 121. A failure at any position in a particular reduction causes the rest of the comparisons for that reduction to be ignored and a jump is made to the beginning of the next reduction. A successful comparison causes the next element of the reduction to be tested, until the whole rule has been successfully matched with the top elements of the stack, in which case a semantic routine is invoked. If the topmost element of the stack is not the rightmost element of the successfully compared reduction, the next symbol on the input stream is brought onto the stack and control returns to the head of the table to begin the whole process over again. The only way to reach line 119 is through a successful match at line 118. Line 119 tests the top position of the stack for "STATEMENT" (we already know that the second position on the stack contains an "IF CLAUSE") and transfers to line 120 upon success and line 121 upon failure. Line 120 makes the last test for the reduction being considered. If the next symbol in the input stream is the keyword "ELSE" then bring it into consideration by placing it on the top of the stack and go to the beginning of the table. If the next token is something else, then jump to row 121 and continue.

These last three rows point out several things. First, if comparisons fail at any point within a given reduction, control is transferred within the table to the beginning of the tests for the next reduction rule. These tests are not optimized in the sense that if comparisons fail at some point in the middle of a reduction, after having made one or more successful comparisons, and if the next reduction tests for the same symbols at the same positions on the stack, then redundant comparisons are made.

Second, rather than test in row 121 for the right context character, it would be possible to combine rows 120 and 121 and bring in the next symbol after a successful match on the top of the stack. This is a reduction without right context verification and would save room in the table. But, by retaining this information regarding what symbol is expected next in the input stream, more complete diagnostics are available if the reduction fails.

Lines 121, 122, and 123 define the tests and actions for an entire reduction. The reduction is:

$$\langle \text{STATEMENT LIST} \rangle :: = \langle \text{STATEMENT LIST} \rangle \quad \langle \text{STATEMENT} \rangle$$

Lines 121 and 122 indicate the two comparisons to be made to the first and second positions of the stack and the actions to be taken in all success/failure cases. If line 122 tests successfully, then the negative second element of that row (-130) indicates the semantic routine which is to do the semantic processing associated with the reduction. Row 123 is a special one used by the reductions routine to indicate the operations to be performed on the stack. The first element is blank (zero) since no comparison is made. The second element defines the number

of elements of the stack which are to be reduced (2) and is equal to the number of elements on the right side of a production rule. The third element specifies the syntactic value which is to be placed on the new top-of-stack and corresponds to the nonterminal symbol which those one or more symbols reduces to. In the case shown, a six (6) is the syntactic value of the symbol "STATEMENT LIST".

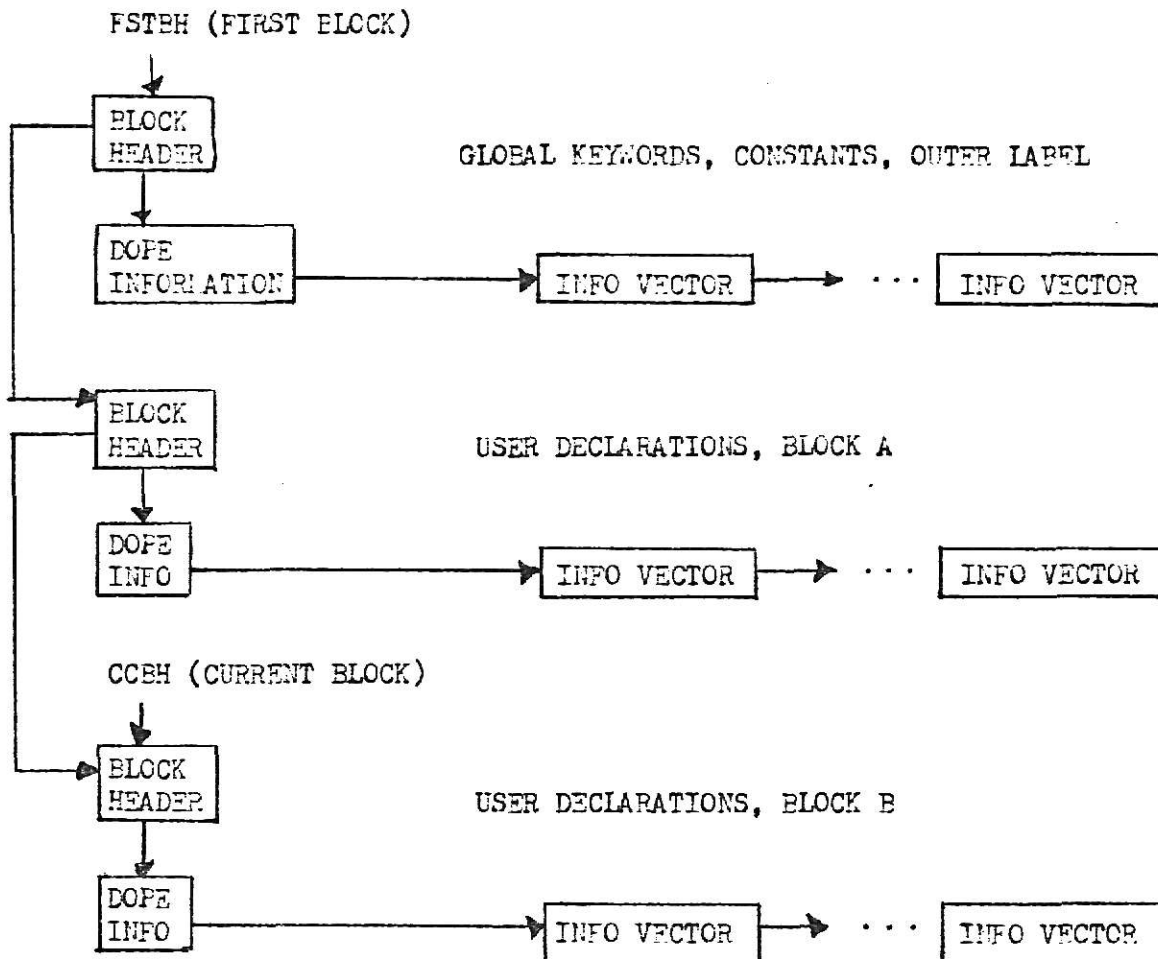
Chapter 4

BASIC DATA STRUCTURES

Basic to any program is the way in which information is stored and referenced. If the data structures are poorly designed then certainly the program as a whole cannot function efficiently. The information which is to be stored and the basic patterns of reference determine how data must be kept to optimize references and facilitate finding needed information. The structures in this compiler are simple enough to understand yet powerful enough to allow efficient reference.

The first to be considered is the symbol table. The symbol table, referred to as SYM, is effectively divided into three parts, of which all entries consist of a five element vector. The first part is the free list. This is a singly linked list of free vectors which are not currently in use by either of the other two substructures within SYM. This list is linked by the last (fifth) element of each free vector, the last of which has a zero link. A global pointer to the current first vector of the list may be referenced at any time for additions or deletions. During compiler initialization all vectors of SYM are linked together in the free list.

The second substructure of SYM is the symbol table containing the keywords and their definitions, and the user defined symbols and their definitions. Figure 4.1 contains an abbreviated symbol table as it might appear at the indicated point while processing the simple program A. Being a single pass compiler and at the same time accepting a grammar which defines blocks, the obvious symbol table structure is a stack in which each



```

A: | BEGIN
   | DECLARATIONS
   |
   | B: | BEGIN
   |   | DECLARATIONS
   |   |
   |   | ← COMPILATION HERE
   |   |
   |   | END
   | END
  
```

SIMPLIFIED SYMBOL TABLE STRUCTURE

FIGURE 4.1

level corresponds to a new block level within the program being scanned. As a matter of convenience, the outermost level contains all keywords, constants, and the label of the program. A single search through the symbol table starting with the innermost block is sufficient to determine if a character string or literal constant has been encountered within the current environment, and if so, what its definition is.

Each level is preceeded by a block header and a dope vector. The block header contains 1.) a link to the previous block header, 2.) a pointer to the dope vector, 3.) a link to the next inner block, 4.) the accumulated data storage area assigned within the block, and 5.) the largest data area total offset assigned within any contained (at a lower level) block. The last two must be clarified.

Storage assignment is always done relative to a single base location for the entire program. Since this is a single pass compiler, some means must be used to allow blocks at the same level to share the same storage areas. This is accomplished with the fourth and fifth elements of the block header. When a block is entered, static storage assignments begin at location zero; that is, with a zero displacement off the data pool base address. As storage is assigned, the fourth element, the accumulated data area, is incremented to reflect the increasing displacements. The fifth element is always initially zero. As data references are made throughout the block in which the item is defined, or any inner block in which the variable is not redefined with the same name, they are linked together in the intermediate code array.

All references to any storage declared within a block are linked to the block header of declaration. When a block exit is performed, if the fifth element of the block header is nonzero (the largest total data area assigned for any contained block), then all linked data reference offsets for the (about to be exited) block are incremented by that amount. If the accumulated data area for this same block is larger than the largest previous total data area assigned for any block at the same level and also in the same containing block, then the largest data area of the containing block is set to the assigned area of the block being exited. This scheme allows equivalence defining for storage assigned in parallel blocks.

A simple example will illustrate this. If the entire program consists of a single block A which declares ten words of storage, then any reference to one of these words will be of the form

.DATA+XXX

where XXX is less than ten. This assembles to a single word address XXX words beyond the data pool base '.DATA'.

As these references are made and code is generated for them, they are linked together in the intermediate output. When the block A is exited, because no inner block was encountered, these displacements suffice.

But suppose the program consists of a main block A with an inner block B. Suppose further that within block A ten words of storage are declared and referenced before block B is even encountered in the input stream. For the time being, these are assigned a displacement in the same fashion as the previous example, that is, with displacements less than ten.

Now if we wish to define storage local to block B, say one word, which does not overlap that of A, then either that one word of B must be assigned a displacement of ten off .DATA, or it could be assigned a displacement of zero and all previous references in block A could be shifted up one word. The former is easier but fails, or at least becomes somewhat complicated, if, upon exiting block B more storage is assigned within block A. For this reason the latter method is chosen.

When block B is exited, the storage assignment of one word is noted in the fifth element of the block header for block A. When block A is exited, all references to data declared in it have their displacement values shifted up one to allow for the data word in block B.

The dope vector contains the pointer to the linked references to data declared within the immediate block. It also contains a pointer to the vector descriptions of symbols declared within the immediate block. Each user defined symbol is entered into the symbol table with enough information to do minimal error checking. Each is assigned at least one vector of five words, and a second if the variable is subscripted. The first word is always a 16 bit indicator which identifies storage class, precision, and so on. The bit assignments are as follows:

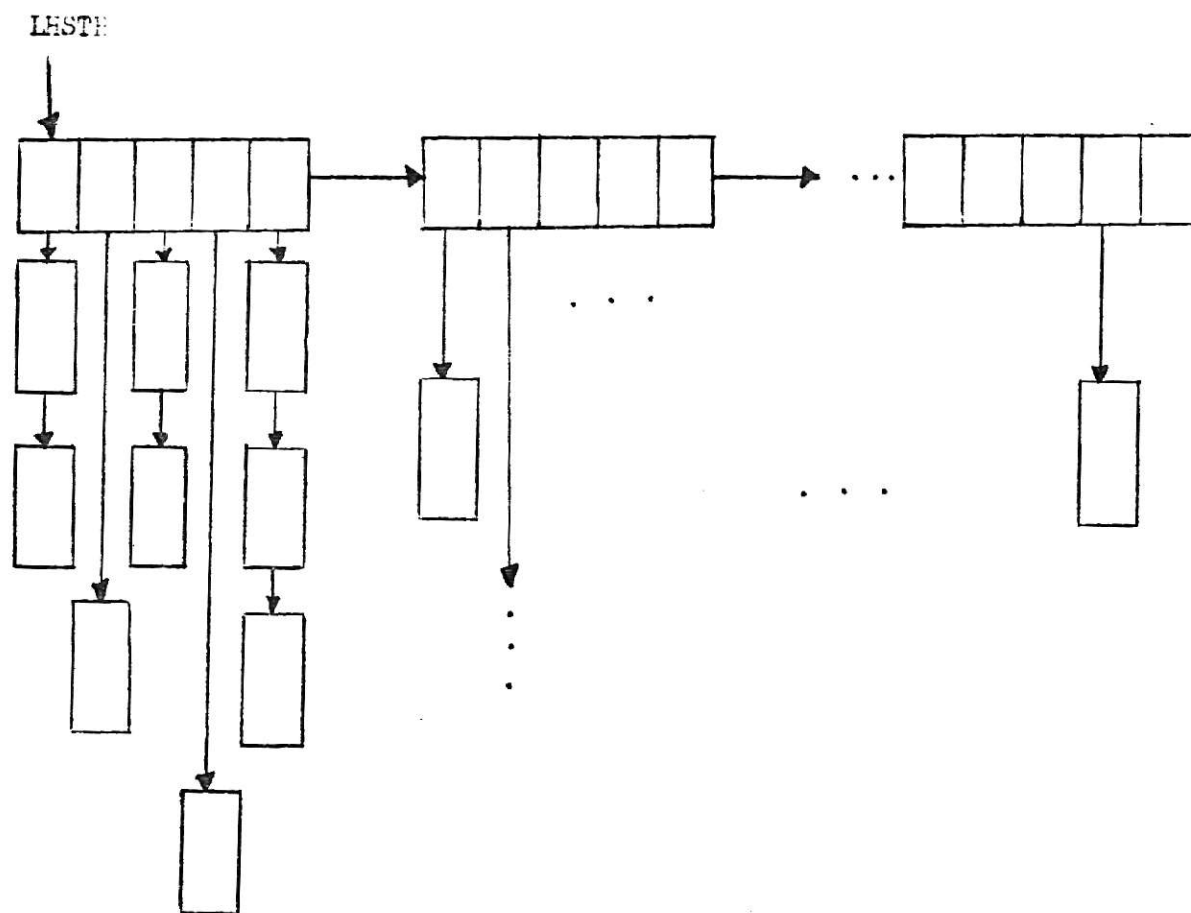
<u>BIT #</u>	<u>MEANING</u>
0	permanence (symbol table or semantic stack)
1	integer
2	character
3	literal
4	label
5	constant
6	precision (single or double)
7	dimensionality
8	keyword
9	assigned temporary storage
10	subscripted variable
11-15	not used

The second word of the vector is usually a pointer into a character array for those symbols and string constants which have a string representation.

The third and fourth words vary with the item described, and contain information such as addresses, unique values for labels, are pointers to more information. The fifth is a link to the next symbol defined within the same block.

The third substructure within SYM contains sets of left hand sides for those nonterminal symbols which appear to the right of a terminal symbol in some production. This allows a finer degree of error checking during syntactic analysis at the expense of the storage necessary to hold this information. For example, if at some point during scanning it is known that the next symbol on the input stream begins some nonterminal construct, then three choices present themselves. It would be possible to do a look ahead more than a single symbol and construct that nonterminal from the next N terminal symbols, but this requires a great deal of added effort. It also is possible to enter into the reductions table an extra reductions rule for every terminal symbol which may legally be the beginning of the expected incoming construct. This is easy to do and is currently a user option but requires a tremendous amount of additional table capacity.

The third alternative is simply to do a table look-up. Given that a particular nonterminal equivalent is expected next in the incoming stream, one may find the left hand set of that symbol and compare all its elements to the single next token in the stream. This is what is done. Figure 4.2



LEFT HAND SIDE TABLE STRUCTURE

FIGURE 4.2

illustrates the structure of lists used to store the left hand sides for all nonterminals.

Syntactic values of nonterminals are used to locate the correct sublist containing the left hand set.

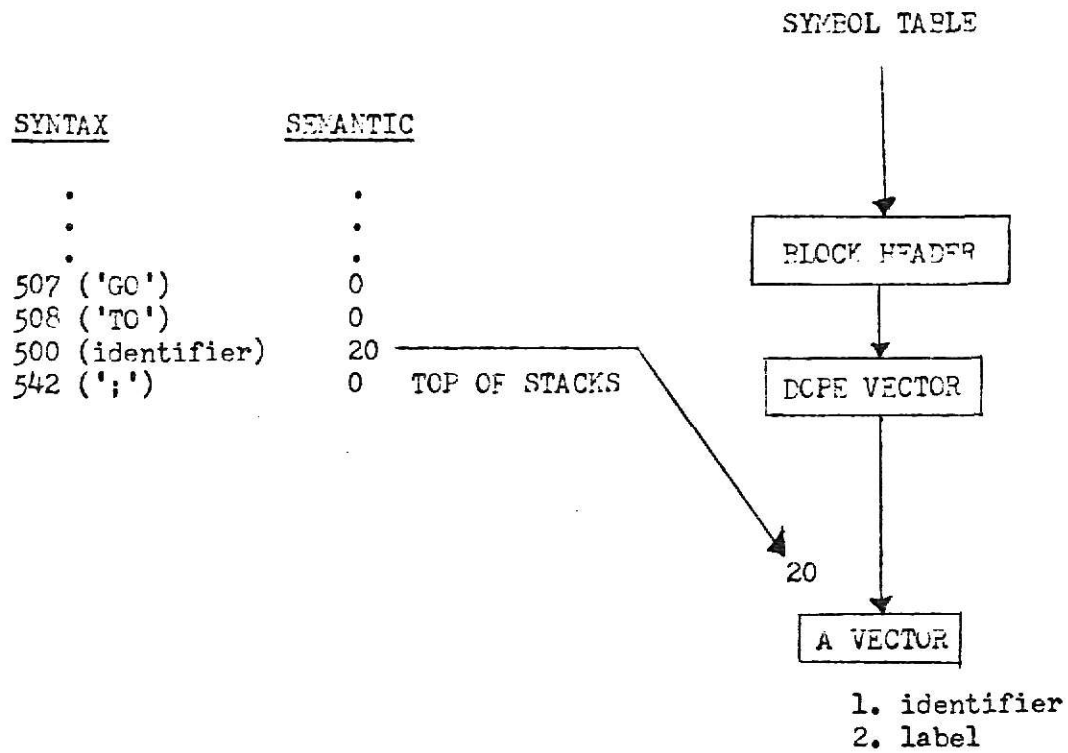
The next important structure to be considered is the stack. What has until now been referred to as simply the stack is actually two parallel stacks, the syntax stack and the semantic stack. As illustrated in Fig. 4.3, the syntax stack contains only the syntactic values of those items described semantically by the corresponding elements of the semantic stack. The semantic stack contains a zero in a position which needs no semantic information, and a pointer to an information vector of five words if there is some semantic information associated with that position on the stack.

All additions and deletions take place in parallel on the tops of the stacks. As shown in Fig. 4.3, the top four elements of the stacks contain the syntactic construct

GO TO IDENTIFIER ;

whose numeric representation on the syntactic stack consists of the values 507, 508, 500, and 542. The identifier, which is user defined, is the only item which need contain any associated semantic information and this is found by a pointer into the symbol table. Twenty (20) is the index of the symbol table vector which describes the identifier referenced in the example construct (transfer statement).

Changes to the stack take place in two ways. Either the scanner introduces a new symbol from the input stream, in which case syntactic and



STACK STRUCTURE BEFORE REDUCTION

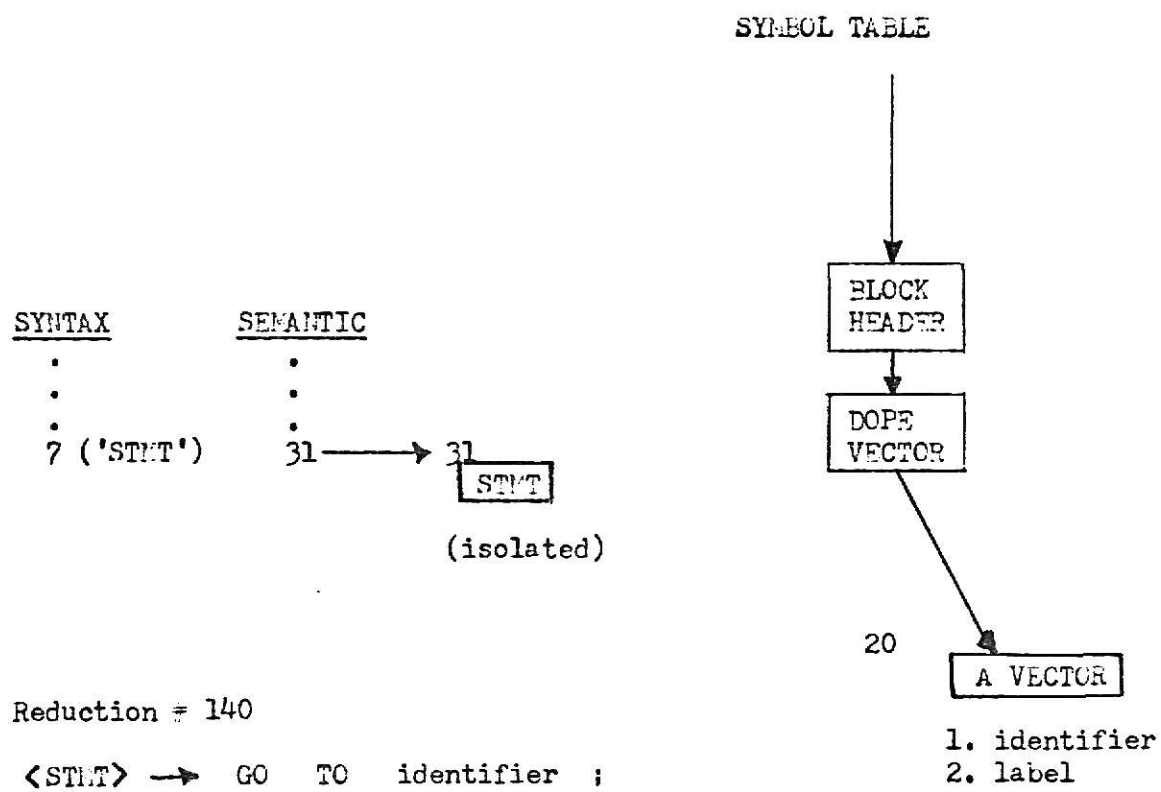
FIGURE 4.3

possibly semantic information is added to the top of the stack, or a complete construct is recognized on the top of the stack and a reduction of one or more elements off the top of the stacks takes place, causing a new symbol to be placed on the new top of the stack.

It is the job of the lexical scanner to place new information on the top of the stack in the case of keywords or user variables encountered in the input stream. It is sufficient to retain only the syntactic value for keywords (and enter a zero semantic pointer), whereas user defined symbols must be associated with a vector of the symbol table for further semantic processing.

Referring to figure 4.4, suppose some reduction rule #140 is applied, which effects a reduction of the top four elements of the stack to a single element and that element is to be a nonterminal symbol "STATEMENT". The semantic routine responsible for this particular reduction first gathers the information it will need from all those elements which are to take part in the reduction. The appropriate code is generated; the top four elements of both stacks are deallocated; and a new element is placed on the top of both stacks. The top semantic pointer references a new semantic unit called "STATEMENT" which now contains all the information which may be needed in the future to perform a further reduction over that stack position.

It is interesting to note that no change takes place within the symbol table itself. This is logical because the environment has in no way changed. The symbol table undergoes structural changes only when a declaration statement is scanned, or a block is entered or exited.



STACK STRUCTURE AFTER REDUCTION

FIGURE 4.4

As indicated in Fig. 4.4, the semantic information associated with the newly created "STATEMENT" is isolated in the sense that the only link to it is through the semantic stack itself. When the part of the stack which references this item is later involved in a reduction, the five word vector containing its semantic information is returned to the free pool. The routine which effects a reduction therefore must be able to distinguish between those units of semantic information vectors which are linked within the symbol table, and those which are "free". This is possible by the first bit of the first word of the vector, previously referred to as the "permanence" bit. If this bit is on at reduction time, the information is an integral part of the symbol table and is left untouched. If this bit is off at reduction time, the information is "floating" and may be released and returned to the free list.

The last major data structure to be considered is an array which holds the intermediate code as it is generated. This array, CODE, consists of any number of rows of three elements each. As code is generated, entries are placed into CODE in consecutive rows.

The first element of a row is a bit indicator specifying the type of information contained within that particular row. It may specify a class of operators (mnemonics), a label to be generated, a data reference, etc.

The second element optionally contains a numeric value necessary, for example, when a label is to be output. All user defined and internally generated labels are mapped into a common format and given a unique value. A typical label expression might be:

Lxxx:

where "L" is indicated by the bit configuration of the first word of the row and the number "xxx" is the value of the second word of the same row.

The third element of the row is used to link together several rows when necessary. For example, all references to any data declared within a particular block are linked in the third element of the row describing each reference. If the need arises to shift, or increase, their respective offsets from the common data base address, then a pass through the chain of references can be made, where the second words contain offset values. Similarly, all forward references to a label in a block are linked together in their third row element in the event that label is not encountered inside the immediate block. This allows these same references to be effectively moved to the containing block at block exit time, and the numeric representation of that label to be changed to refer to a label in an outer block. As stated before, the chain of data references is pointed to by a word of the dope vector of the defining block. The chain of label references is pointed to by a word in the semantic vector of the symbol table descriptor. When the block exit routine scans the symbols in the block being exited and finds a label which has never been resolved, it has simply to move that information into a containing block (assuming no naming conflicts).

No code optimization is done at any time. After scanning the entire program and generating the corresponding code array, nothing further remains but to generate in a suitable format the assembly code from the intermediate code array.

Chapter 5

COMPILER STRUCTURE

The second program of this project, previously referred to as PGM2, constitutes the body of the compiler itself. This program is essentially written in the simple language it is designed to accept as input, and produces readable Nova assembler language as output. This chapter describes the general characteristics of the compiler as a whole, and the functions of the numerous modules as they relate to each other. A complete listing of the program appears in appendix VI.

Initial input to the compiler consists of the data necessary to initialize both the symbol table and the reductions table. This is produced in the correct format from a previous run of PGM1. Following initialization, an algorithm in the source language is included in punched card form. This program is scanned in a single pass of the compiler, and if no terminal errors are encountered, the equivalent Nova program is printed on the output.

The compiler can logically be divided into several sections. The lexical scanner accepts strings of text, divides it into tokens, and determines basically what type of symbol is being processed. For example, the scanner is structured to recognize the class of literals, the class of special symbols (\$,*,etc.), the class of numeric strings, or the class of keywords and user defined symbols. If necessary, a symbol is entered into the symbol table by the scanner. User symbols

are distinguished from keywords by a symbol table search. Constants are entered as global symbols if there is not a previous occurrence of the same constant already in the table. Nothing but the most elementary type checking takes place at this time.

After the scanner has processed the token and determined the general class (literal, special symbol, etc.), two units of information are returned to the syntactic analyzer. (The scanner is a subroutine, which is called as needed. As with all subroutines which make up the compiler, information is passed only through variables whose scope includes both the calling and the called routine. No argument lists are allowed in the language compiled, and therefore in the compiler itself.) The first variable passed is the syntactic value of the token just scanned. Variables, literals, and constants are assigned values as a class. Keywords and special operators are assigned unique syntactic values as defined by the initialization input data. The second variable passed back is a pointer into the symbol table to the semantic information associated with that same token just scanned. These two values then define the single token scanned, whatever class, each time the scanner is called.

The syntax analyzer is the control routine which calls the scanner whenever it determines that another token is to be brought in from the input stream. Initially, both stacks (the syntax and semantic stacks) are empty. The scanner is called once to introduce a single element to the top of each stack. The analyzer then begins at the first row of the reduction table and performs comparisons of elements on the stack

to symbols, or classes of symbols, as specified by the current row of the reductions table. Whenever a success "branch" is taken (as dictated by this table) whose action specifies that a new token be pushed onto the stack, the scanner is called, and matching begins again from the start of the table. Otherwise matches continue to be checked until either a success or failure "branch" specifies a particular semantic routine which is to receive control. This may happen as a result of all elements of a reduction rule yielding successful matches, in which case a reduction of the stack will occur, or if a syntactical error is encountered and a fix-up routine of some sort must be called. In any event, the syntactical analyzer is responsible for additions to the stack and transferring control to particular semantic routines.

The semantic routines correspond one-to-one with the rewrite rules which define the grammar. Each is called when it is determined that the top elements of the stack match the construct in the language for which that routine is written. These semantic routines are responsible for generating the necessary code, reducing the stacks, and keeping whatever information may be needed at a later time. With over one hundred semantic routines all performing many similar functions a set of support procedures was written which may be called individually whenever needed. This allows a great deal less code, and changes in the semantics can be made quite easily. The major routines are described below.

KEEP. The procedure titled KEEP places a symbol into the symbol table. The symbol is contained within a character string called NAME which is

built within the scanner. It may be kept at any level of the symbol table as determined by a pointer to the block header of that level.

BLKEX. This procedure is called by several semantic routines to effect a block exit. It deallocates the innermost, or lowest, level of the symbol table. Error checking is limited to undefined labels whose scope may not be enlarged to include the containing block due to naming conflicts.

BLKEN. BLKEN is invoked whenever a block is entered. It performs the simple linking chores necessary to create a new inner block. A global variable is maintained which always points to the current innermost block.

VFIND. This routine performs a search outward from a given level for the first occurrence of the symbol contained in the globally defined string "NAME". If the symbol is found, a pointer (the same as the semantic pointer returned from the scanner) is returned to that symbol table entry. If the symbol is not found, this pointer is set to zero.

SETUP. Because the compiler is organized around the stack concept, a number of routines perform simple operations on the two stacks. References to semantic information, for example, are made indirectly through positions on these stacks. When a semantic routine needs to reference information pertaining to some element of its construct, it does so by specifying the position on the semantic stack which contains a pointer to the symbol table information. SETUP loads information into five global variables from the vector in the symbol table located by a

specified stack position. SET# specifies that position. If the information associated with the top position of the stack is needed, SET# is set to 1, SETUP is called, and five globally known variables are loaded with that information. This enables simple changes in the syntax to be made relatively easily since the setting of SET# need only be changed in the appropriate semantic routines.

REDUCE. After a semantic routine has collected all the information it needs from the symbol table, REDUCE performs a simple reduction on both stacks as dictated by the reductions table. The new top element of the syntax stack will contain the syntactic value of the result. The top element of the semantic stack contains a pointer to a free five word vector to be used to store the semantic information associated with the result of the reduction.

GENER. Intermediate output is kept in a two dimensional array called "CODES". Each row of three words, as described earlier, represents some part of the output which is to be generated. As the semantic routines determine that some output is needed, three global variables corresponding to the three elements of the next row of CODES are set and GENER is called. These three elements are inserted into the CODES array, and if necessary, this row is linked by the third element to another row. All data references are linked together in this manner, the head of the chain being pointed to by the block header for which those data areas apply. Similarly, all references to the same label are linked together until that label is found in the scan.

YLDSP, YLDCH, YLDDP, YIELD. These four routines are all concerned with generating code for data conversions. YLDSP yields a single precision integer from a double precision integer. YLDCH yields an Ascii character string from a double precision integer. YLDDP yields a double precision integer from either a single precision integer or from an Ascii character string. Any of these three routines may be called directly from a semantic routine, or from YIELD, which is expected to yield a desired precision when the original precision may be any of the above.

A short semantic routine will illustrate some of the logic behind the above described routines. The following is the code used to implement the rule:

< PROCEDURE STATEMENT > :: = < LABEL > PROCEDURE

<u>SEMANTIC ROUTINE # 110</u>	<u>STATEMENT #</u>
S110: OPTYP=101;	1
CALL GENER;	2
CALL UNAME;	3
OPTYP=22;	4
OPVAL=UNIQE;	5
CALL GENER;	6
SET#=2;	7
CALL SETUP;	8
OPTYP=20;	9
OPVAL=SETC;	10
CALL GENER;	11
CALL REDUCE;	12
CALL BLKEN;	13
SETD=UNIQE;	14
SETB=16;	15
CALL SETIT;	16
GO TO MATCH;	17

A few points should be made first.

1. OPTYP, OPVAL, AND OPLNK are the three globally defined variables which correspond to the three elements of one row of the intermediate code array, CODES. When the procedure titled GENER is called, it expects these three variables to contain information to be entered and the manner in which to do so.
2. UNAME is a procedure which generates consecutive integers. The variable UNIQUE is returned with the current value.
3. SETA, SETB, SETC, SETD, SETE are five globally defined variables which correspond respectively to the five words of each vector in the symbol table. These are initialized by SETUP according to the stack position indicated by SET#.

Semantic routine S110 will receive control when the syntax analyzer recognizes the indicated construct on the top of the stack. Statement #1 sets OPTYP to the value assigned a JUMP pseudo-op, and statement #2 transfers to GENER, which places that value next in the intermediate code array. The code generated for a procedure is an unconditional jump around the procedure body to an internally generated label, followed by the unique label assigned to the label of the procedure. This particular rewrite rule is not concerned with the code of the procedure; it has not yet been scanned. Entry into a procedure can only happen by a procedure call.

Statement #3 calls upon UNAME to generate a unique constant to be used as part of the internal label of the statement following the procedure

body itself. Given the JUMP pseudo-op, its argument must be that label. Statement #4 defines the type of code to be generated (a label), statement #5 sets the numeric value of that label, and statement #6 calls upon GENER to enter that information into the intermediate code array, CODES. The generated code to this point is essentially the following:

JUMP

LXXX:

where LXXX: will appear after the procedure body.

Now the procedure label must be generated. This label has been assigned a unique numeric value because it has been previously encountered by the scanner, and entered into at least one reduction leading up to the < LABEL > in the current stack. Statement #7 indicates that this label information is located through the second position of the semantics stack, and statement #8 loads all that information into the five variables, SETA, SETB, SETC, SETD, and SETE.

The value assigned to that label is located in the third word, which was loaded into variable SETC. Statement #10 therefore sets up that value for a call to GENER, which takes place at statement #11. That is all the code that is needed.

Statement #12 performs a reduction on the stacks. The syntactic value for < PROCEDURE STATEMENT > becomes the top element of the syntax stack and the top element of the semantic stack points to an empty five word vector to be used to store whatever information must be associated with the nonterminal < PROCEDURE STATEMENT >.

Due to the nature of this particular rule, statement #13 calls the routine BLKEN to create a new inner level in the symbol table. At this point no further information can be entered into the symbol table other than the block header and dope vector information.

It turns out that the only information to keep with any < STATEMENT > is the statement type. This is only done so as not to generate multiple RETURNS at the end of a procedure. A < PROCEDURE STATEMENT > however is special because the generated label of the statement following the procedure must be kept and output at block exit. That label value is therefore saved in statement # 15. Statement # 16 stores SETA, SETB, SETC, SETD, SETE directly into the information vector corresponding to the top element of the semantics stack, and in statement # 17 control is returned to the syntax analyzer for further processing.

Chapter 6

SUMMARY

The compiler presented in this paper was debugged and tested with the examples found in Appendix IV. The code generated consists of system calls. This is not as inefficient as it may appear because only the return address status information need be saved for each call. The scope of the project is not such as to warrant extensive code generation facilities, and this type of code does allow reasonably good run-time diagnostics. For example, subscript bounds may be checked by including the limiting values as arguments to the address computation routine. Furthermore, generation of in-line code would in most cases expand into considerably more code than that required for a system call. This becomes quite significant on machines with limited core available, but does require extensive run-time packages.

It was discovered while debugging that although a particular set of productions may truly define the language, some amount of tailoring must take place to match the rules with the semantic processing which takes place during recognition. For example, to allow right context verification, the scanner must maintain a look ahead of one token which has not been entered onto the stacks. Since the scanner also does the table lookup for symbols encountered in the stream, if that symbol is to be entered into the symbol table, this will take place before any matches are attempted for that symbol. If two conditions hold, then an error condition arises. If the right context symbol is one that is

entered into the symbol table at scan time, and the next reduction on the stacks causes a block to be entered (a new inner level), then that symbol just entered appears at the wrong level within the symbol table. The solution is trivial, but was overlooked in the original language definition.

The small amount of information actually necessary to generate code is kept in the symbol table. Although little effort would be required to add to this basis, interesting side effects appear by doing minimal error checking. For example, the label on a procedure block is indistinguishable (within the symbol table) from one appearing on an ordinary statement. A transfer statement may therefore reference a procedure block, or a subroutine call may not actually refer to a subroutine. If the runtime routines implement calls and returns by saving status information on a stack, then the user has actually a more powerful program structure than that possible had the compiler made certain additional compile time checks.

Thus knowing the logic of the compiler, allows one to take advantage of peculiarities inherent in its design. PLAN has no means for defining storage assignment equivalences, but since parallel blocks allocate storage contiguously, and from the same point, this common area may be considered overlaid by the programmer.

Certain constructs of PL/I-F level which were not suited for the Nova were used in the writing of the compiler but not included in the definition on PLAN. Similar statements were designed to perform in a like manner. To resolve this within a self-compiling compiler a means

was devised where statements could be selectively included or excluded from the scan. To exclude a string from the code being scanned, it may appear as follows.

```
/* % */ statement /* % */
```

That portion of the text appearing between, and including, the two percent (%) signs is not in any way recognized by the scanner. Note that since PL/I-F is not so restricted, it will include the text contained between the two comment strings (a comment is delimited by a '/' and '*' pair).

The corresponding textual inclusion may be effected by the following coding which is specially recognized by the scanner.

```
/* @ statement @ */
```

'Statement' in this case will not be compiled by the PL/I-F compiler since it is found embedded within a valid comment. The PLAN scanner however recognizes a '@' as a special switch requesting that the comment currently being scanned be temporarily exited and the following code compiled. The example below shows how this sequence may be used.

```
/* % */
```

```
DECLARE I BINARY FIXED (15);
```

```
/* % */
```

```
/* @ DECLARE I SINGLE; @ */
```

The PL/I-F compiler will recognize the first declaration, but not the second because it is in a comment. The PLAN compiler will not recognize

the first but will scan the second. This is a useful tool for bootstrapping.

Several examples of acceptable PLAN code appear in appendix IV. It should be noted that all keywords are reserved and no attempt at fixup is made should an illegal occurrence of one be found. Also, any user defined variable which must be declared must not be encountered in the text before such a declaration has been processed. Declarations may appear anyplace in the source code.

This compiler has not currently been implemented on a Nova mini-computer. It does generate a great deal of legible Nova assembler code, and a number of the run-time routines exist. The major problems of half-bootstrapping have not been solved, nor has any means been designed to segment the code generated for such a relatively large program on a small mini-computer. This might not be too difficult if done on the intermediate code, or even if major segmentation decisions are made during compilation. The major data structures, excepting the symbol table, would likely be kept as disk resident files, with some small changes to the compiler needed.

BIBLIOGRAPHY

1. H. R. Haynes and L. J. Schutte. "Compilation of Optimized Syntactic Recognizers from Floyd-Evans Productions", SIGPLAN Notices, V 7 (July 1970), 38-51
2. Gries, David. Compiler Construction for Digital Computers. John Wiley & Sons, Inc. New York 1971

APPENDIX I

Language Definition

PRODUCTIONS

```

/100/ <PROCEDURE BLOCK> ::= <PROCEDURE STATEMENT> ; <STATEMENT LIST> END
/105/ <BEGIN BLOCK> ::= <BEGIN STATEMENT> ; <STATEMENT LIST> END
/107/ <DO GROUP> ::= DO ; <STATEMENT LIST> END ;
/110/ <PROCEDURE STATEMENT> ::= <LABEL> PROCEDURE
/115/ <BEGIN STATEMENT> ::= <LABEL> BEGIN
/120/ <BEGIN STATEMENT> ::= BEGIN
/125/ <LABEL> ::= IDENTIFIER ;
/130/ <STATEMENT LIST> ::= <STATEMENT LIST> <STATEMENT>
/135/ <STATEMENT LIST> ::= <STATEMENT>
/140/ <STATEMENT> ::= GO TO IDENTIFIER ;
/145/ <STATEMENT> ::= RETURN ;
/150/ <STATEMENT> ::= <IF STATEMENT> ;
/155/ <STATEMENT> ::= <ASSIGNMENT STATEMENT> ;
/160/ <STATEMENT> ::= <DO STATEMENT> ;
/162/ <STATEMENT> ::= <DO GROUP>
/165/ <STATEMENT> ::= <I/O STREAM STATEMENT> ;
/170/ <STATEMENT> ::= <I/O RECORD STATEMENT> ;
/175/ <STATEMENT> ::= <OPEN CLOSE STATEMENT> ;
/180/ <STATEMENT> ::= <DECLARATION STMT> ;
/185/ <STATEMENT> ::= <PROCEDURE BLOCK> ;
/190/ <STATEMENT> ::= <BEGIN BLOCK> ;
/193/ <STATEMENT> ::= CALL IDENTIFIER ;
/196/ <STATEMENT> ::= <LABEL> ;
/197/ <STATEMENT> ::= LABEL ( IDENTIFIER ) ;
/198/ <STATEMENT> ::= STOP CONSTANT ;
/199/ <STATEMENT> ::= STOP ;

```

```

/200/ <IF STATEMENT> ::= <IF CLAUSE> <STATEMENT>
/205/ <IF STATEMENT> ::= <TRUE PART> <STATEMENT>
/210/ <IF CLAUSE> ::= IF <LOGICAL EXPRESSION> THEN
/215/ <TRUE PART> ::= <IF CLAUSE> <STATEMENT> ELSE
/230/ <DO STATEMENT> ::= <ITERATION LIMITER> <STATEMENT LIST> END :
/235/ <DO 1> ::= DO IDENTIFIER = <EXPRESSION>
/240/ <DO 2> ::= <DO 1> BY <EXPRESSION>
/245/ <ITERATION LIMITER> ::= <DO 2> TO <EXPRESSION>
/270/ <I/O STREAM STATEMENT> ::= <I/O S LIST> ) ;
/271/ <I/O STREAM STATEMENT> ::= PUT SKIP ;
/272/ <I/O STREAM STATEMENT> ::= PUT PAGE ;
/275/ <I/O STREAM STATEMENT> ::= <FMT S LIST> ) ;
/280/ <I/O S LIST> ::= <DEVICE PHRASE> ( <EXPRESSION>
/285/ <I/O S LIST> ::= <I/O S LIST> , <EXPRESSION>
/290/ <FMT S LIST> ::= <I/O S LIST> ) ( <FORMAT ITEM>
/295/ <FMT S LIST> ::= <FMT S LIST> , <FORMAT ITEM>
/300/ <DEVICE PHRASE> ::= GET FILE ( <DEVICE> )
/301/ <DEVICE PHRASE> ::= PUT FILE ( <DEVICE> )
/305/ <DEVICE> ::= TTI
/310/ <DEVICE> ::= TTO
/315/ <DEVICE> ::= PTP
/320/ <DEVICE> ::= PTR
/325/ <DEVICE> ::= SEL
/330/ <FORMAT ITEM> ::= PAGE
/335/ <FORMAT ITEM> ::= SKIP
/340/ <FORMAT ITEM> ::= TAB
/345/ <FORMAT ITEM> ::= F
/350/ <FORMAT ITEM> ::= A
/355/ <FORMAT ITEM> ::= F ( CONSTANT )
/360/ <FORMAT ITEM> ::= A ( CONSTANT )

```



```

/365/ <FORMAT ITEM> ::= X ( CONSTANT )
/370/ <FORMAT ITEM> ::= LITERAL
/400/ <CHARACTER SEGMENT> ::= SUBSTR ( <VARIABLE> , <EXPRESSION> , <EXPRESSION> )
/405/ <PSEUDO VARIABLE> ::= INDEX ( <VARIABLE> , <VARIABLE> )
/410/ <PSEUDO VARIABLE> ::= LENGTH ( <VARIABLE> )
/415/ <PSEUDO VARIABLE> ::= AND ( <VARIABLE> , <PRIMARY> )
/420/ <PSEUDO VARIABLE> ::= OR ( <VARIABLE> , <PRIMARY> )
/425/ <PSEUDO VARIABLE> ::= SWAP ( <VARIABLE> )
/445/ <VARIABLE> ::= IDENTIFIER
/450/ <VARIABLE> ::= <SUBSCRIPTED VARIABLE>
/451/ <VARIABLE> ::= <PSEUDO VARIABLE>
/455/ <SUBSCRIPTED VARIABLE> ::= IDENTIFIER ( <EXPRESSION> )
/460/ <SUBSCRIPTED VARIABLE> ::= IDENTIFIER ( <EXPRESSION> , <EXPRESSION> )
/500/ <ASSIGNMENT STATEMENT> ::= <PRIMARY> = <EXPRESSION>
/501/ <ASSIGNMENT STATEMENT> ::= <PRIMARY> = <CHARACTER SEGMENT>
/502/ <ASSIGNMENT STATEMENT> ::= <CHARACTER SEGMENT> = <CHARACTER SEGMENT>
/503/ <ASSIGNMENT STATEMENT> ::= <CHARACTER SEGMENT> = <VARIABLE>
/515/ <EXPRESSION> ::= <TERM>
/520/ <EXPRESSION> ::= <ADD OP> <TERM>
/525/ <EXPRESSION> ::= <EXPRESSION> <ADD OP> <TERM>
/530/ <TERM> ::= <FACTOR>
/535/ <TERM> ::= <FACTOR> <MULT OP> <TERM>
/540/ <FACTOR> ::= <PRIMARY>
/545/ <FACTOR> ::= <PRIMARY> <EXPON OP> <FACTOR>
/550/ <PRIMARY> ::= CONSTANT
/555/ <PRIMARY> ::= <VARIABLE>
/560/ <PRIMARY> ::= ( <EXPRESSION> )
/565/ <PRIMARY> ::= LITERAL
/580/ <OPEN CLOSE STATEMENT> ::= OPEN FILE ( LITERAL , <VARIABLE> , <VARIABLE> )
/590/ <OPEN CLOSE STATEMENT> ::= CLOSE FILE ( LITERAL , <VARIABLE> , <VARIABLE> )

```

```

/620/ <I/O RECORD STATEMENT> ::= READ FILE ( <VARIABLE> ) <REC KEY>
/625/ <I/O RECORD STATEMENT> ::= WRITE FILE ( <VARIABLE> ) <REC KEY>
/630/ <REC KEY> ::= KEY ( <VARIABLE> )
/660/ <DECLARATION STMT> ::= <DCL PART> <OPTION>
/665/ <DCL PART> ::= DECLARE IDENTIFIER
/670/ <DCL PART> ::= DECLARE IDENTIFIER ( CONSTANT )
/675/ <DCL PART> ::= DECLARE IDENTIFIER ( CONSTANT , CONSTANT )
/680/ <OPTION> ::= SINGLE
/685/ <OPTION> ::= DOUBLE
/690/ <OPTION> ::= CHARACTER ( CONSTANT )
/700/ <LOGICAL EXPRESSION> ::= <EXPRESSION> <LOGICAL OPERATOR> <EXPRESSION>
/705/ <LOGICAL OPERATOR> ::= <
/710/ <LOGICAL OPERATOR> ::= >
/715/ <LOGICAL OPERATOR> ::= <=
/720/ <LOGICAL OPERATOR> ::= >=
/725/ <LOGICAL OPERATOR> ::= =
/726/ <LOGICAL OPERATOR> ::= !=
/730/ <ADD OP> ::= +
/735/ <AND OP> ::= -
/740/ <MULT OP> ::= *
/745/ <MULT OP> ::= /
/750/ <EXPON OP> ::= **

```

APPENDIX II

Left Hand Sides


```

AREA
DECLARATION_SMT DECLARE
OPTION_LIST
OPTION SINGLE DOUBLE CHARACTER
PICTURE
LOGICAL_EXPRESSION
INDEX LENGTH AND OR SWAP IDENTIFIER CONSTANT ( LITERAL + -
LOGICAL_OPERATOR
< > <= >= = !=
ADD_OP + -
MULT_OP * /
EXPON_OP **
DECL_PARI DECLARE
CHARACTER_SEGMENT
SUBSTR
DO_GROUP 00

```

APPENDIX III

Formatted Reductions


```

(130)* <STATEMENT LIST> LITERAL -> ENTER
(130)* <STATEMENT LIST> CONSTANT -> ENTER
(130)* <STATEMENT LIST> IDENTIFIER -> ENTER

```

STATEMENT

```

(215) <IF CLAUSE> <STATEMENT> ELSE -> ENTER
(215) <STATEMENT LIST> <STATEMENT> -> <STATEMENT LIST>
(200) <IF CLAUSE> <STATEMENT> -> <IF STATEMENT>
(205) <TRUE PART> <STATEMENT> -> <IF STATEMENT>
(135) <STATEMENT> -> <STATEMENT LIST>

```

IF STATEMENT

```

THIS IS AN ABBREVIATED BLOCK
(150)

```

```

<IF STATEMENT> : -> ENTER

```

IF CLAUSE

```

THIS IS AN ABBREVIATED BLOCK

```

```

<IF CLAUSE> STOP -> ENTER
<IF CLAUSE> STOP -> ENTER
<IF CLAUSE> LABEL -> ENTER
<IF CLAUSE> LABEL -> ENTER
<IF CLAUSE> CALL -> ENTER
<IF CLAUSE> CALL -> ENTER
<IF CLAUSE> DECLARE -> ENTER
<IF CLAUSE> DECLARE -> ENTER
<IF CLAUSE> ( -> ENTER
<IF CLAUSE> { -> ENTER
<IF CLAUSE> CLOSE -> ENTER
<IF CLAUSE> CLOSE -> ENTER
<IF CLAUSE> OPEN -> ENTER
<IF CLAUSE> OPEN -> ENTER
<IF CLAUSE> SWAP -> ENTER
<IF CLAUSE> SWAP -> ENTER
<IF CLAUSE> OR -> ENTER
<IF CLAUSE> OR -> ENTER
<IF CLAUSE> AND -> ENTER
<IF CLAUSE> AND -> ENTER
<IF CLAUSE> LENGTH -> ENTER
<IF CLAUSE> LENGTH -> ENTER
<IF CLAUSE> INDEX -> ENTER
<IF CLAUSE> INDEX -> ENTER
<IF CLAUSE> SUBSTR -> ENTER
<IF CLAUSE> SUBSTR -> ENTER
<IF CLAUSE> IF -> ENTER
<IF CLAUSE> IF -> ENTER
<IF CLAUSE> WRITE -> ENTER
<IF CLAUSE> WRITE -> ENTER
<IF CLAUSE> READ -> ENTER
<IF CLAUSE> READ -> ENTER
<IF CLAUSE> PUT -> ENTER
<IF CLAUSE> PUT -> ENTER
<IF CLAUSE> GET -> ENTER
<IF CLAUSE> GET -> ENTER
<IF CLAUSE> DO -> ENTER
<IF CLAUSE> DO -> ENTER
<IF CLAUSE> GO -> ENTER
<IF CLAUSE> GO -> ENTER
<IF CLAUSE> RETURN -> ENTER
<IF CLAUSE> RETURN -> ENTER
<IF CLAUSE> BEGIN -> ENTER

```



```

(215)* <IF CLAUSE> BEGIN -> ENTER
(200)* <IF CLAUSE> LITERAL -> ENTER
(215)* <IF CLAUSE> LITERAL -> ENTER
(200)* <IF CLAUSE> CONSTANT -> ENTER
(215)* <IF CLAUSE> CONSTANT -> ENTER
(200)* <IF CLAUSE> IDENTIFIER -> ENTER
(215)* <IF CLAUSE> IDENTIFIER -> ENTER

```

```

TRUE_PART
THIS IS AN ABBREVIATED BLOCK
(205)* <TRUE PART> STOP -> ENTER
(205)* <TRUE PART> LABEL -> ENTER
(205)* <TRUE PART> CALL -> ENTER
(205)* <TRUE PART> DECLARE -> ENTER
(205)* <TRUE PART> I -> ENTER
(205)* <TRUE PART> CLOSE -> ENTER
(205)* <TRUE PART> OPEN -> ENTER
(205)* <TRUE PART> SWAP -> ENTER
(205)* <TRUE PART> OR -> ENTER
(205)* <TRUE PART> AND -> ENTER
(205)* <TRUE PART> LENGTH -> ENTER
(205)* <TRUE PART> INDEX -> ENTER
(205)* <TRUE PART> SUBSTR -> ENTER
(205)* <TRUE PART> IF -> ENTER
(205)* <TRUE PART> WRITE -> ENTER
(205)* <TRUE PART> READ -> ENTER
(205)* <TRUE PART> PUT -> ENTER
(205)* <TRUE PART> GET -> ENTER
(205)* <TRUE PART> DO -> ENTER
(205)* <TRUE PART> GO -> ENTER
(205)* <TRUE PART> RETURN -> ENTER
(205)* <TRUE PART> BEGIN -> ENTER
(205)* <TRUE PART> LITERAL -> ENTER
(205)* <TRUE PART> CONSTANT -> ENTER
(205)* <TRUE PART> IDENTIFIER -> ENTER

```

```

DO_STATEMENT
THIS IS AN ABBREVIATED BLOCK
(160) <DO STATEMENT> : -> ENTER

```

```

DO_1
THIS IS AN ABBREVIATED BLOCK
(240) <DO 1> BY -> ENTER

```

```

DO_2
THIS IS AN ABBREVIATED BLOCK
(245) <DO 2> TO -> ENTER

```

```

ITERATION_LIMITER
THIS IS AN ABBREVIATED BLOCK
(230)* <ITERATION LIMITER> STOP -> ENTER
(230)* <ITERATION LIMITER> LABEL -> ENTER
(230)* <ITERATION LIMITER> CALL -> ENTER
(230)* <ITERATION LIMITER> DECLARE -> ENTER
(230)* <ITERATION LIMITER> I -> ENTER
(230)* <ITERATION LIMITER> CLOSE -> ENTER
(230)* <ITERATION LIMITER> OPEN -> ENTER
(230)* <ITERATION LIMITER> SWAP -> ENTER
(230)* <ITERATION LIMITER> OR -> ENTER
(230)* <ITERATION LIMITER> AND -> ENTER

```



```

(580) OPEN FILE ( LITERAL , <VARIABLE> , -> ENTER
(590) CLOSE FILE ( LITERAL , <VARIABLE> , -> ENTER
(405) INDEX ( <VARIABLE> , <VARIABLE> ) -> ENTER
(620) READ FILE ( <VARIABLE> ) -> ENTER
(625) WRITE FILE ( <VARIABLE> ) -> ENTER
(410) LENGTH ( <VARIABLE> ) -> ENTER
(425) SWAP ( <VARIABLE> ) -> ENTER
(630) KEY ( <VARIABLE> ) -> ENTER
(400) SUBSTR ( <VARIABLE> , -> ENTER
(405) INDEX ( <VARIABLE> , -> ENTER
(415) AND ( <VARIABLE> , -> ENTER
(420) OR ( <VARIABLE> , -> ENTER
(503) <CHARACTER SEGMENT> = <VARIABLE> -> <ASSIGNMENT STATEMENT>
(555) <VARIABLE> -> <PRIMARY>

<SUBSCRIPTED VARIABLE> -> <VARIABLE>

SUBSCRIPTS
THIS IS AN ABBREVIATED BLOCK

ASSIGNMENT STATEMENT
THIS IS AN ABBREVIATED BLOCK
(155)

LEFT PART
THIS IS AN ABBREVIATED BLOCK

EXPRESSION
(400) SUBSTR ( <VARIABLE> , <EXPRESSION> , <EXPRESSION> ) -> ENTER
(460) IDENTIFIER ( <EXPRESSION> , <EXPRESSION> ) -> ENTER
(400) SUBSTR ( <VARIABLE> , <EXPRESSION> , -> ENTER
(455) IDENTIFIER ( <EXPRESSION> ) -> ENTER
(460) IDENTIFIER ( <EXPRESSION> , -> ENTER
(560) ( <EXPRESSION> ) -> ENTER
(700)* <EXPRESSION> = -> ENTER
(700)* <EXPRESSION> = -> ENTER
(700)* <EXPRESSION> > -> ENTER
(700)* <EXPRESSION> <= -> ENTER
(700)* <EXPRESSION> < -> ENTER
(700)* <EXPRESSION> > -> ENTER
(525)* <EXPRESSION> - -> ENTER
(525)* <EXPRESSION> + -> ENTER
(235) DO IDENTIFIER = <EXPRESSION> -> <DO 1>
(240) <DO 1> BY <EXPRESSION> -> <DO 2>
(245) <DO 2> TO <EXPRESSION> -> <ITERATION LIMITER>
(280) <DEVICE PHRASE> ( <EXPRESSION> -> <I/O S LIST>
(285) <I/O S LIST> , <EXPRESSION> -> <I/O S LIST>
(500) <PRIMARY> = <EXPRESSION> -> <ASSIGNMENT STATEMENT>
(700) <EXPRESSION> <LOGICAL OPERATOR> <EXPRESSION> -> <LOGICAL EXPRESSION>

TERM
(535) <EXPRESSION> <ADD OP> <TERM> -> <EXPRESSION>
(535) <FACTOR> <MULT OP> <TERM> -> <TERM>
(520) <ADD OP> <TERM> -> <EXPRESSION>
(515) <TERM> -> <EXPRESSION>

FACTOR
(535)* <FACTOR> / -> ENTER
(535)* <FACTOR> * -> ENTER

```

```

(545)
(530)

PRIMARY <EXPON OP> <FACTOR> -> <FACTOR>
          <FACTOR> -> <TERM>

AND ( <VARIABLE> , <PRIMARY> ) -> ENTER
OR ( <VARIABLE> , <PRIMARY> ) -> ENTER
    <PRIMARY> = -> ENTER
    <PRIMARY> = -> ENTER
    <PRIMARY> ** -> ENTER
    <PRIMARY> -> <FACTOR>

OPEN_CLOSE_STATEMENT
THIS IS AN ABBREVIATED BLOCK
(175)

OPEN_CLOSE_LIST
THIS IS AN ABBREVIATED BLOCK

OPEN_OR_CLOSE
THIS IS AN ABBREVIATED BLOCK

I/O_RECORD_COMMAND
THIS IS AN ABBREVIATED BLOCK

REC_KEY
(620)
(625)

AREA
THIS IS AN ABBREVIATED BLOCK

DECLARATION_STMT
THIS IS AN ABBREVIATED BLOCK
(180)

OPTION_LIST
THIS IS AN ABBREVIATED BLOCK

OPTION
(650)

PICTURE
THIS IS AN ABBREVIATED BLOCK

LOGICAL_EXPRESSION
THIS IS AN ABBREVIATED BLOCK
(210)

LOGICAL_OPERATOR
THIS IS AN ABBREVIATED BLOCK
(700)*
(700)*
(700)*
(700)*
(700)*
(700)*
(700)*
(700)*
(700)*
(700)*

READ FILE ( <VARIABLE> ) <REC KEY> -> <I/O RECORD STATEMENT>
WRITE FILE ( <VARIABLE> ) <REC KEY> -> <I/O RECORD STATEMENT>

<DECLARATION_STMT> ; -> ENTER

<DCL PART> <OPTION> -> <DECLARATION_STMT>

IF <LOGICAL_EXPRESSION> THEN -> ENTER

<EXPRESSION> <LOGICAL_OPERATOR> - -> ENTER
<EXPRESSION> <LOGICAL_OPERATOR> + -> ENTER
<EXPRESSION> <LOGICAL_OPERATOR> ( -> ENTER
<EXPRESSION> <LOGICAL_OPERATOR> SWAP -> ENTER
<EXPRESSION> <LOGICAL_OPERATOR> OR -> ENTER
<EXPRESSION> <LOGICAL_OPERATOR> AND -> ENTER
<EXPRESSION> <LOGICAL_OPERATOR> LENGTH -> ENTER
<EXPRESSION> <LOGICAL_OPERATOR> INDEX -> ENTER
<EXPRESSION> <LOGICAL_OPERATOR> LITERAL -> ENTER
<EXPRESSION> <LOGICAL_OPERATOR> CONSTANT -> ENTER

```

```

(700)*
ADD_OP
THIS IS AN ABBREVIATED BLOCK
(525)*
(525)*
(525)*
(525)*
(525)*
(525)*
(525)*
(525)*
(525)*
(520)*
(520)*
(520)*
(520)*
(520)*
(520)*
(520)*
(520)*
MULT_OP
THIS IS AN ABBREVIATED BLOCK
(535)*
(535)*
(535)*
(535)*
(535)*
(535)*
(535)*
(535)*
(535)*
(535)*
EXPON_OP
THIS IS AN ABBREVIATED BLOCK
(545)*
(545)*
(545)*
(545)*
(545)*
(545)*
(545)*
(545)*
DCL_PART
THIS IS AN ABBREVIATED BLOCK
(660)*
(660)*
(660)*
CHARACTER_SEGMENT
(502)
(503)
(501)
(502)
DO_GROUP

```

<EXPRESSION> <LOGICAL OPERATOR> IDENTIFIER -> ENTER
 <EXPRESSION> <ADD OP> (-> ENTER
 <EXPRESSION> <ADD OP> SWAP -> ENTER
 <EXPRESSION> <ADD OP> OR -> ENTER
 <EXPRESSION> <ADD OP> AND -> ENTER
 <EXPRESSION> <ADD OP> LENGTH -> ENTER
 <EXPRESSION> <ADD OP> INDEX -> ENTER
 <EXPRESSION> <ADD OP> LITERAL -> ENTER
 <EXPRESSION> <ADD OP> CONSTANT -> ENTER
 <EXPRESSION> <ADD OP> IDENTIFIER -> ENTER
 <ADD OP> (-> ENTER
 <ADD OP> SWAP -> ENTER
 <ADD OP> OR -> ENTER
 <ADD OP> AND -> ENTER
 <ADD OP> LENGTH -> ENTER
 <ADD OP> INDEX -> ENTER
 <ADD OP> LITERAL -> ENTER
 <ADD OP> CONSTANT -> ENTER
 <ADD OP> IDENTIFIER -> ENTER
 <FACTOR> <MULT OP> (-> ENTER
 <FACTOR> <MULT OP> SWAP -> ENTER
 <FACTOR> <MULT OP> OR -> ENTER
 <FACTOR> <MULT OP> AND -> ENTER
 <FACTOR> <MULT OP> LENGTH -> ENTER
 <FACTOR> <MULT OP> INDEX -> ENTER
 <FACTOR> <MULT OP> LITERAL -> ENTER
 <FACTOR> <MULT OP> CONSTANT -> ENTER
 <FACTOR> <MULT OP> IDENTIFIER -> ENTER
 <PRIMARY> <EXPON OP> (-> ENTER
 <PRIMARY> <EXPON OP> SWAP -> ENTER
 <PRIMARY> <EXPON OP> OR -> ENTER
 <PRIMARY> <EXPON OP> AND -> ENTER
 <PRIMARY> <EXPON OP> LENGTH -> ENTER
 <PRIMARY> <EXPON OP> INDEX -> ENTER
 <PRIMARY> <EXPON OP> LITERAL -> ENTER
 <PRIMARY> <EXPON OP> CONSTANT -> ENTER
 <PRIMARY> <EXPON OP> IDENTIFIER -> ENTER
 <DCL PART> CHARACTER -> ENTER
 <DCL PART> DOUBLE -> ENTER
 <DCL PART> SINGLE -> ENTER
 <CHARACTER SEGMENT> = -> ENTER
 <CHARACTER SEGMENT> -> ENTER
 <PRIMARY> = <CHARACTER SEGMENT> -> <ASSIGNMENT STATEMENT>
 <CHARACTER SEGMENT> = <CHARACTER SEGMENT> -> <ASSIGNMENT STATEMENT>

```

(162)
IDENTIFIER
(197)
(140)
(235)
(670)
(675)
(103)
(455)
(450)
(125)
(665)
(445)

CONSTANT
(675)
(670)
(675)
(355)
(360)
(365)
(690)
(198)
(550)

LITERAL
(580)
(570)
(370)
(565)

PROCEDURE
(110)

BEGIN
(115)
(120)

RETURN
THIS IS AN ABBREVIATED BLOCK
(145)

END
(107)
(230)
(100)
(105)

GO
THIS IS AN ABBREVIATED BLOCK
(140)

DO
THIS IS AN ABBREVIATED BLOCK
(245)*
(245)*
(245)*
(245)*

<DO GROUP> -> <STATEMENT>

LABEL ( IDENTIFIER ) -> ENTER
GO TO IDENTIFIER : -> ENTER
DO IDENTIFIER = -> ENTER
DECLARE IDENTIFIER ( -> ENTER
DECLARE IDENTIFIER ( -> ENTER
CALL IDENTIFIER : -> ENTER
IDENTIFIER ( -> ENTER
IDENTIFIER ( -> ENTER
IDENTIFIER : -> <DCL PART>
IDENTIFIER -> <VARIABLE>

DECLARE IDENTIFIER ( CONSTANT , CONSTANT ) -> ENTER
DECLARE IDENTIFIER ( CONSTANT ) -> ENTER
DECLARE IDENTIFIER ( CONSTANT , -> ENTER
F ( CONSTANT ) -> ENTER
A ( CONSTANT ) -> ENTER
X ( CONSTANT ) -> ENTER
CHARACTER ( CONSTANT ) -> ENTER
STOP CONSTANT : -> ENTER
CONSTANT -> <PRIMARY>

OPEN FILE ( LITERAL , -> ENTER
CLOSE FILE ( LITERAL , -> ENTER
LITERAL -> <FORMAT ITEM>
LITERAL -> <PRIMARY>

<LABEL> PROCEDURE -> <PROCEDURE STATEMENT>

<LABEL> BEGIN -> <BEGIN STATEMENT>
BEGIN -> <BEGIN STATEMENT>

RETURN : -> ENTER

DO : <STATEMENT LIST> END : -> ENTER
<ITERATION LIMITER> <STATEMENT LIST> END : -> ENTER
<PROCEDURE STATEMENT> : <STATEMENT LIST> END -> <PROCEDURE BLOCK>
<BEGIN STATEMENT> : <STATEMENT LIST> END -> <BEGIN BLOCK>

GO TO -> ENTER
<DO 2> TO -> ENTER
<DO 2> TO + -> ENTER
<DO 2> TO ( -> ENTER
<DO 2> TO SWAP -> ENTER
<DO 2> TO OR -> ENTER

```


THIS IS AN ABBREVIATED BLOCK

(300)
(301)
(590)
(620)
(625)

GET FILE (-> ENTER
PUT FILE (-> ENTER
OPEN FILE (-> ENTER
CLOSE FILE (-> ENTER
READ FILE (-> ENTER
WRITE FILE (-> ENTER

III
(305)

TTI -> <DEVICE>

IIQ
(310)

TTQ -> <DEVICE>

PIP
(315)

PTP -> <DEVICE>

PIR
(320)

PTR -> <DEVICE>

SEL
(325)

SEL -> <DEVICE>

PAGE
(272)
(330)

PUT PAGE : -> ENTER
PAGE -> <FORMAT ITEM>

SKIP
(271)
(335)

PUT SKIP : -> ENTER
SKIP -> <FORMAT ITEM>

IAR
(340)

TAB -> <FORMAT ITEM>

SUBSTR

THIS IS AN ABBREVIATED BLOCK

(400)

SUBSTR (-> ENTER

INDEX
THIS IS AN ABBREVIATED BLOCK
(405)

INDEX (-> ENTER

LENGTH

THIS IS AN ABBREVIATED BLOCK

(410)

LENGTH (-> ENTER

AND
THIS IS AN ABBREVIATED BLOCK
(415)

AND (-> ENTER

OR
THIS IS AN ABBREVIATED BLOCK
(420)

OR (-> ENTER

SWAP
THIS IS AN ABBREVIATED BLOCK
(425)

SWAP (-> ENTER

OPEN

THIS IS AN ABBREVIATED BLOCK


```

(271)
(272)
(275)
(145)
(150)
(155)
(160)
(165)
(170)
(175)
(180)
(185)
(190)
(196)
(199)

THIS IS AN ABBREVIATED BLOCK

THIS IS AN ABBREVIATED BLOCK

THIS IS AN ABBREVIATED BLOCK

(620)*
(625)*
(197)
(290)
(270)
(275)
(430)
(530)
(420)
(675)
(435)
(415)
(420)
(460)
(300)
(301)
(670)
(355)
(360)
(365)
(410)
(425)
(455)
(630)
(630)
(560)

THIS IS AN ABBREVIATED BLOCK

(730)
=

```

```

PUT SKIP :
PUT PAGE :
<FMT S LIST> :
RETURN :
<ASSIGNMENT STATEMENT> :
<DO STATEMENT> :
<I/O STREAM STATEMENT> :
<I/O RECORD STATEMENT> :
<OPEN CLOSE STATEMENT> :
<DECLARATION STATEMENT> :
<PROCEDURE BLOCK> :
<BEGIN BLOCK> :
<LABEL> :
STOP :

PUT FILE ( <VARIABLE> ) KEY -> ENTER
WRITE FILE ( <VARIABLE> ) KEY -> ENTER
LABEL ( IDENTIFIER ) : -> ENTER
<I/O S LIST> ( ) -> ENTER
<I/O S LIST> : -> ENTER
<FMT S LIST> : -> ENTER
OPEN FILE ( LITERAL , <VARIABLE> , <VARIABLE> ) -> <OPEN CLOSE STATEMENT>
CLOSE FILE ( LITERAL , <VARIABLE> , <VARIABLE> ) -> <OPEN CLOSE STATEMENT>
SUBSTR ( <VARIABLE> , <EXPRESSION> , <EXPRESSION> ) -> <CHARACTER SEGMENT>
DECLARE IDENTIFIER ( CONSTANT , <VARIABLE> ) -> <DECL PART>
INDEX ( <VARIABLE> , <VARIABLE> ) -> <PSEUDO VARIABLE>
OR ( <VARIABLE> , <PRIMARY> ) -> <PSEUDO VARIABLE>
IDENTIFIER ( <EXPRESSION> , <EXPRESSION> ) -> <PSEUDO VARIABLE>
GET FILE ( <DEVICE> ) -> <SUBSCRIPTED VARIABLE>
PUT FILE ( <DEVICE> ) -> <DEVICE PHRASE>
DECLARE IDENTIFIER ( CONSTANT ) -> <DECL PART>
F ( CONSTANT ) -> <FORMAT ITEM>
A ( CONSTANT ) -> <FORMAT ITEM>
X ( CONSTANT ) -> <FORMAT ITEM>
LENGTH ( <VARIABLE> ) -> <PSEUDO VARIABLE>
SWAP ( <VARIABLE> ) -> <PSEUDO VARIABLE>
IDENTIFIER ( <EXPRESSION> ) -> <SUBSCRIPTED VARIABLE>
KEY ( <VARIABLE> ) -> <RFC KEY>
CHARACTER ( CONSTANT ) -> <OPTION>
( <EXPRESSION> ) -> <PRIMARY>

```

+ -> <ADD OP>

```

(735)      - -> <ADD OP>
(740)      * -> <MULT OP>
(745)      / -> <MULT OP>
(750)      ** -> <EXPON OP>
(755)      < -> <LOGICAL OPERATOR>
(760)      > -> <LOGICAL OPERATOR>
(765)      <= -> <LOGICAL OPERATOR>
(770)      >= -> <LOGICAL OPERATOR>

```

```

(775)      DO IDENTIFIER = - -> ENTER
(780)      DO IDENTIFIER = + -> ENTER
(785)      DO IDENTIFIER = ( -> ENTER
(790)      DO IDENTIFIER = SWAP -> ENTER
(795)      DO IDENTIFIER = OR -> ENTER
(800)      DO IDENTIFIER = AND -> ENTER
(805)      DO IDENTIFIER = LENGTH -> ENTER
(810)      DO IDENTIFIER = INDEX -> ENTER
(815)      DO IDENTIFIER = LITERAL -> ENTER
(820)      DO IDENTIFIER = CONSTANT -> ENTER
(825)      DO IDENTIFIER = IDENTIFIER -> ENTER
(830)      <PRIMARY> = - -> ENTER
(835)      <PRIMARY> = + -> ENTER
(840)      <PRIMARY> = ( -> ENTER
(845)      <PRIMARY> = SWAP -> ENTER
(850)      <PRIMARY> = SWAP -> ENTER
(855)      <PRIMARY> = OR -> ENTER
(860)      <PRIMARY> = AND -> ENTER
(865)      <PRIMARY> = LENGTH -> ENTER
(870)      <PRIMARY> = LENGTH -> ENTER
(875)      <PRIMARY> = INDEX -> ENTER
(880)      <PRIMARY> = INDEX -> ENTER
(885)      <PRIMARY> = SUBSTR -> ENTER
(890)      <PRIMARY> = SUBSTR -> ENTER
(895)      <PRIMARY> = LITERAL -> ENTER
(900)      <PRIMARY> = CONSTANT -> ENTER
(905)      <PRIMARY> = IDENTIFIER -> ENTER
(910)      <PRIMARY> = IDENTIFIER -> ENTER
(915)      <PRIMARY> = -> <LOGICAL OPERATOR>
(920)      != -> <LOGICAL OPERATOR>
(925)      THEN

```

(210)	IF <LOGICAL EXPRESSION> THEN ->	<IF CLAUSE>
ELSE		
(215)	<IF CLAUSE> <STATEMENT> ELSE ->	<TRUE PART>
DECLARE		
(645)	THIS IS AN ABBREVIATED BLOCK	
(670)		DECLARE IDENTIFIER -> ENTER
(675)		DECLARE IDENTIFIER -> ENTER
X		
(355)	THIS IS AN ABBREVIATED BLOCK	X (-> ENTER
CALL		
(193)	THIS IS AN ABBREVIATED BLOCK	CALL IDENTIFIER -> ENTER
LABEL		
(197)	THIS IS AN ABBREVIATED BLOCK	LABEL (-> ENTER
STOP		
(199)	THIS IS AN ABBREVIATED BLOCK	STOP ; -> ENTER
(198)		STOP CONSTANT -> ENTER

A NUVA COMPILER FOR A SIMPLE BLOCK STRUCTURED GRAMMAR
DOUG CRANK 1973

		COMPLETIF REDUCTIONS			
		SUCCESS	FAILURE		
L 1	POSITION & MATCH	0	2	#	1
L 1	PROCEDURE BLOCK	0	3	#	2
L 1	BEGIN BLOCK	0	4	#	3
L 1	PROCEDURE STATEMENT	0	5	#	4
L 1	REGIN STATEMENT	0	6	#	5
L 1	LABEL	0	7	#	6
L 1	STATEMENT LIST	118	8	#	7
L 1	STATEMENT	0	9	#	8
L 1	IF STATEMENT	0	10	#	9
L 1	IF CLAUSE	0	11	#	10
L 1	TRUE PART	0	12	#	11
L 1	DO STATEMENT	0	13	#	12
L 1	DO 1	0	14	#	13
L 1	DO 2	0	15	#	14
L 1	ITERATION LIMITER	0	16	#	15
L 1	I/O STREAM STATEMENT	0	17	#	16
L 1	I/O RECORD STATEMENT	0	18	#	17
L 1	I/O S LIST	0	19	#	18
L 1	FMT S LIST	0	20	#	19
L 1	DEVICE PHRASE	0	21	#	20
L 1	DEVICE	0	22	#	21
L 1	FORMAT ITEM	133	23	#	22
L 1	PSEUDO VARIABLE	143	24	#	23
L 1	PSEUDO LIST	0	25	#	24
L 1	FUNCTION NAME	0	26	#	25
L 1	VARIABLE	146	27	#	26
L 1	SUBSCRIPTED VARIABLE	229	28	#	27
L 1	SUBSCRIPTS	0	29	#	28
L 1	ASSIGNMENT STATEMENT	0	30	#	29
L 1	LEFT PART	0	31	#	30
L 1	EXPRESSION	232	32	#	31
L 1	TEMP	309	33	#	32
L 1	FACTOR	334	34	#	33
L 1	PRIMARY	0	35	#	34
L 1	OPEN CLOSE STATEMENT	0	36	#	35
L 1	OPEN CLOSE LIST	0	37	#	36
L 1	OPEN OR CLOSE	0	38	#	37
L 1	I/O RECORD COMMAND	0	39	#	38
L 1	REC KEY	355	40	#	39
L 1	AREA	0	41	#	40
L 1	DECLARATION STMT	0	42	#	41
L 1	OPTION LIST	0	43	#	42
L 1	OPTION	370	44	#	43
L 1	PICTURE	0	45	#	44
L 1	LOGICAL EXPRESSION	0	46	#	45
L 1	LOGICAL OPERATOR	0	47	#	46
L 1	ADD OP	0	48	#	47
L 1	MULT OP	0	49	#	48
L 1	EXPON OP	0	50	#	49
L 1	DECL PART	0	51	#	50
L 1	CHARACTER SEGMENT	374	52	#	51
L 1	DO GROUP	387			

L	IDENTIFIER	390	53	# 52
L	CONSTANT	422	54	# 53
L	LITERAL	461	55	# 54
L	PROCEDURE	476	56	# 55
L	BEGIN	480	57	# 56
L	RETURN	0	58	# 57
L	END	486	59	# 58
L	GO	0	60	# 59
L	TO	0	61	# 60
L	BY	0	62	# 61
L	DO	0	63	# 62
L	GET	0	64	# 63
L	PUT	0	65	# 64
L	READ	0	66	# 65
L	WRITE	0	67	# 66
L	IF	0	68	# 67
L	FILE	0	69	# 68
L	TTI	506	70	# 69
L	ITO	509	71	# 70
L	PTP	512	72	# 71
L	PTR	515	73	# 72
L	SFL	518	74	# 73
L	PAGE	521	75	# 74
L	SKIP	527	76	# 75
L	TAB	533	77	# 76
L	SUBSTR	0	78	# 77
L	INDEX	0	79	# 78
L	LENGTH	0	80	# 79
L	AND	0	81	# 80
L	OR	0	82	# 81
L	SWAP	0	83	# 82
L	OPEN	0	84	# 83
L	CLOSE	0	85	# 84
L	KEY	0	86	# 85
L	AREA	0	87	# 86
L	SINGLE	536	88	# 87
L	DOUBLE	539	89	# 88
L	CHARACTER	0	90	# 89
L	F	542	91	# 90
L	A	547	92	# 91
L	X	0	93	# 92
L	:	552	94	# 93
L	:	556	95	# 94
L	:	0	96	# 95
L	:	0	97	# 96
L	:	0	98	# 97
L	:	0	99	# 98
L	:	0	100	# 99
L	:	1018	101	# 100
L	:	1021	102	# 101
L	:	1024	103	# 102
L	:	1027	104	# 103
L	:	1030	105	# 104
L	:	1033	106	# 105
L	:	1036	107	# 106
L	:	1039	108	# 107
L	:	1042	109	# 108
L	:	1045	110	# 109
L	:	1149	111	# 110
L	:	1152	112	# 111

L 1	FLSF	1157	113	# 112
L 1	DECLARE	0	114	# 113
L 1	X	0	115	# 114
L 1	CALL	0	116	# 115
L 1	LABEL	0	117	# 116
L 1	STOP	0	118	# 117
L 2	IF CLAUSE	119	121	# 118
L 1	STATEMENT	120	121	# 119
R 1	ELSE	0	121	# 120
L 2	STATEMENT LIST	122	124	# 121
L 1	STATEMENT	-130	124	# 122
L 1	<ANY SYMBOL>	2	6	
L 2	IF CLAUSE	125	127	# 124
L 1	STATEMENT	-200	127	# 125
L 1	<ANY SYMBOL>	2	8	
L 2	TRUE PART	128	130	# 127
L 1	STATEMENT	-205	130	# 128
L 1	<ANY SYMBOL>	2	8	
L 1	STATEMENT	-135	132	# 130
L 1	<ANY SYMBOL>	1	6	
L 1	<ANY SYMBOL>	-999	-999	
L 1	***** BLOCK END	*****		
L 4	I/O S LIST	134	138	# 133
L 3	)	135	138	# 134
L 2	(	136	138	# 135
L 1	FORMAT ITEM	-290	138	# 136
L 1	<ANY SYMBOL>	4	18	
L 3	FMT S LIST	139	142	# 138
L 2	)	140	142	# 139
L 1	FORMAT ITEM	-295	142	# 140
L 1	<ANY SYMBOL>	3	18	
L 1	<ANY SYMBOL>	-999	-999	
L 1	***** BLOCK END	*****		
L 1	PSEUDO VARIABLE	-451	145	# 143
L 1	<ANY SYMBOL>	1	25	
L 1	<ANY SYMBOL>	-999	-999	
L 1	***** BLOCK END	*****		
L 9	OPEN	147	155	# 146
L 7	FILE	148	155	# 147
L 6	(	149	155	# 148
L 5	LITERAL	150	155	# 149
L 4	)	151	155	# 150
L 3	VARIABLE	152	155	# 151
L 2	)	153	155	# 152
L 1	VARIABLE	154	155	# 153
R 1	)	0	155	# 154
L 8	CLOSE	156	164	# 155
L 7	FILE	157	164	# 156
L 6	(	158	164	# 157
L 5	LITERAL	159	164	# 158
L 4	)	160	164	# 159
L 3	VARIABLE	161	164	# 160
L 2	)	162	164	# 161
L 1	VARIABLE	163	164	# 162
R 1	)	0	164	# 163
L 6	OPEN	165	171	# 164
L 5	FILE	166	171	# 165
L 4	(	167	171	# 166
L 3	LITERAL	168	171	# 167
L 2	)	169	171	# 168

L 1	VARIABLE	170	171	# 169
R 1	, CLOSE	170	171	# 170
L 6	FILE	172	178	# 171
L 5	(	173	178	# 172
L 4	LITERAL	174	178	# 173
L 3	, VARIABLE	175	178	# 174
L 2	, VARIABLE	176	178	# 175
L 1	, INDEX	177	178	# 176
R 1	(	0	178	# 177
L 5	VARIABLE	179	184	# 178
L 4	, VARIABLE	180	184	# 179
L 3	, VARIABLE	181	184	# 180
L 2	, VARIABLE	182	184	# 181
L 1	, READ	183	184	# 182
R 1	FILE	0	184	# 183
L 4	, VARIABLE	185	189	# 184
L 3	, VARIABLE	186	189	# 185
L 2	, VARIABLE	187	189	# 186
L 1	, SWAP	188	189	# 187
R 1	FILE	0	189	# 188
L 4	, VARIABLE	190	194	# 189
L 3	, VARIABLE	191	194	# 190
L 2	, VARIABLE	192	194	# 191
L 1	, LENGTH	193	194	# 192
R 1	, VARIABLE	0	194	# 193
L 3	, VARIABLE	195	198	# 194
L 2	, VARIABLE	196	198	# 195
L 1	, SWAP	197	198	# 196
R 1	FILE	0	198	# 197
L 3	, VARIABLE	199	202	# 198
L 2	, VARIABLE	200	202	# 199
L 1	, KEY	201	202	# 200
R 1	, VARIABLE	0	202	# 201
L 3	, VARIABLE	203	206	# 202
L 2	, VARIABLE	204	206	# 203
L 1	, SURSTp	205	206	# 204
R 1	, VARIABLE	0	206	# 205
L 3	, VARIABLE	207	210	# 206
L 2	, VARIABLE	208	210	# 207
L 1	, INDEX	209	210	# 208
R 1	(	0	210	# 209
L 3	, VARIABLE	211	214	# 210
L 2	, VARIABLE	212	214	# 211
L 1	, AND	213	214	# 212
R 1	, VARIABLE	0	214	# 213
L 3	, VARIABLE	215	218	# 214
L 2	, VARIABLE	216	218	# 215
L 1	, OR	217	218	# 216
R 1	, CHARACTER SEGMENT	0	218	# 217
L 3	, VARIABLE	219	222	# 218
L 2	, VARIABLE	220	222	# 219
L 1	, CHARACTER SEGMENT	221	222	# 220
R 1	, VARIABLE	0	222	# 221
L 3	, VARIABLE	223	226	# 222
L 2	, VARIABLE	224	226	# 223
L 1	, ANY SYMBOL	-503	226	# 224
L 0	, ANY SYMBOL	3	28	
L 1	, ANY SYMBOL	-555	228	# 226
L 0	, ANY SYMBOL	1	33	
L 0	, ANY SYMBOL	-999	-999	

```

L 1 1 ***** BLOCK END *****
L 0 0 SUBSCRIPTED VARIABLE -450
L 0 0 <ANY SYMBOL> 1
L 0 0 <ANY SYMBOL> -999
L 7 SURSTR 233
L 6 ( 234
L 5 VARIABLE 235
L 4 , 236
L 3 EXPRESSION 237
L 2 , 238
L 1 EXPRESSION 239
L 1 ) 0
L 5 IDENTIFIER 241
L 4 ( 242
L 3 EXPRESSION 243
L 2 , 244
L 1 EXPRESSION 245
L 1 ) 0
L 5 SURSTR 247
L 4 ( 248
L 3 VARIABLE 249
L 2 , 250
L 1 EXPRESSION 251
L 1 ) 0
L 3 IDENTIFIER 253
L 2 ( 254
L 1 EXPRESSION 255
L 1 ) 0
L 3 IDENTIFIER 257
L 2 ( 258
L 1 EXPRESSION 259
L 1 , 0
L 2 ( 261
L 1 EXPRESSION 262
L 1 ) 0
L 1 EXPRESSION 264
L 1 = 0
L 1 EXPRESSION 266
L 1 = 0
L 1 EXPRESSION 268
L 1 >= 0
L 1 EXPRESSION 270
L 1 <= 0
L 1 EXPRESSION 272
L 1 > 0
L 1 EXPRESSION 274
L 1 < 0
L 1 EXPRESSION 276
L 1 - 0
L 1 EXPRESSION 278
L 1 + 0
L 4 DO 280
L 3 IDENTIFIER 281
L 2 = 282
L 1 EXPRESSION -235
L 0 <ANY SYMBOL> 4
L 3 DO 1 285
L 2 BY 286
L 1 EXPRESSION -240
L 229 # 232
L 233 # 233
L 234 # 234
L 235 # 235
L 236 # 236
L 237 # 237
L 238 # 238
L 239 # 239
L 240 # 240
L 241 # 241
L 242 # 242
L 243 # 243
L 244 # 244
L 245 # 245
L 246 # 246
L 247 # 247
L 248 # 248
L 249 # 249
L 250 # 250
L 251 # 251
L 252 # 252
L 253 # 253
L 254 # 254
L 255 # 255
L 256 # 256
L 257 # 257
L 258 # 258
L 259 # 259
L 260 # 260
L 261 # 261
L 262 # 262
L 263 # 263
L 264 # 264
L 265 # 265
L 266 # 266
L 267 # 267
L 268 # 268
L 269 # 269
L 270 # 270
L 271 # 271
L 272 # 272
L 273 # 273
L 274 # 274
L 275 # 275
L 276 # 276
L 277 # 277
L 278 # 278
L 279 # 279
L 280 # 280
L 281 # 281
L 282 # 282
L 283 12
L 284 288
L 285 288
L 286 288

```

```

L 1 1 PRIMARY 345 346 # 344
L 1 1 ) 0 346 # 345
L 1 1 PRIMARY 347 346 # 346
L 1 1 = 0 348 # 347
L 1 1 PRIMARY 349 350 # 348
L 1 1 = 0 350 # 349
L 1 1 PRIMARY 351 352 # 350
L 1 1 ** 0 352 # 351
L 1 1 PRIMARY -540 354 # 352
L 0 1 <ANY SYMBOL> 1 32
L 0 1 <ANY SYMBOL> -999
L 0 ***** BLOCK END *****

L 5 READ 362 # 355
L 5 FILE 362 # 356
L 4 ( 362 # 357
L 3 VARIABLE 362 # 358
L 2 ) 362 # 359
L 1 REC KEY 362 # 360
L 0 <ANY SYMBOL> 16
L 6 WRITE 369 # 362
L 5 FILE 369 # 363
L 4 ( 365 # 364
L 3 VARIABLE 366 # 365
L 2 ) 367 # 366
L 1 REC KEY -625 # 367
L 0 <ANY SYMBOL> 6
L 0 <ANY SYMBOL> -999
L 0 ***** BLOCK END *****

L 2 DCL PART 371 # 370
L 1 OPTION -660 373 # 371
L 0 <ANY SYMBOL> 2
L 0 <ANY SYMBOL> -999
L 0 ***** BLOCK END *****

L 1 CHARACTER SEGMENT 375 # 374
L 1 = 0 376 # 375
L 1 CHARACTER SEGMENT 377 # 376
L 1 = 0 378 # 377
L 3 PRIMARY 379 # 378
L 2 = 380 # 379
L 1 CHARACTER SEGMENT -501 # 380
L 3 3 28
L 3 CHARACTER SEGMENT 383 # 382
L 2 = 384 # 383
L 1 CHARACTER SEGMENT -502 # 384
L 0 <ANY SYMBOL> 3
L 0 <ANY SYMBOL> -999
L 0 ***** BLOCK END *****

L 1 DO GROUP -162 # 387
L 0 <ANY SYMBOL> 1 7
L 0 <ANY SYMBOL> -999
L 0 ***** BLOCK END *****

L 3 LABEL 391 # 390
L 2 ( 392 # 391
L 1 IDENTIFIER 393 # 392
L 1 ) 0 394 # 393
L 3 G 395 # 394
L 2 TO 396 # 395
L 1 IDENTIFIER 397 # 396
L 1 : 0 # 397
L 2 DO 399 # 398

```

```

L 1 IDENTIFIER 400 401 # 399
R 1 = 0 401 # 400
L 2 DECLARE 402 404 # 401
L 1 IDENTIFIER 403 404 # 402
L 1 ( 0 404 # 403
R 1 DECLARE 405 407 # 404
L 1 IDENTIFIER 406 407 # 405
L 1 ( 0 407 # 406
L 2 CALL 408 410 # 407
L 1 IDENTIFIER 409 410 # 408
R 1 ; 0 410 # 409
L 1 IDENTIFIER 411 412 # 410
L 1 ( 0 412 # 411
L 1 IDENTIFIER 413 414 # 412
L 1 ( 0 414 # 413
L 1 IDENTIFIER 415 416 # 414
L 1 ; 0 416 # 415
L 2 DECLARE 417 419 # 416
L 1 IDENTIFIER -665 419 # 417
L 0 <ANY SYMBOL> 2 49
L 1 IDENTIFIER -445 421 # 419
L 1 <ANY SYMBOL> 1 25
L 0 <ANY SYMBOL> -999 -999
L 0 ***** BLOCK END *****

L 6 DECLARE 423 429 # 422
L 5 IDENTIFIER 424 429 # 423
L 4 ( 425 429 # 424
L 3 CONSTANT 426 429 # 425
L 2 , 427 429 # 426
L 1 CONSTANT 428 429 # 427
L 1 ) 0 429 # 428
L 4 DECLARE 430 434 # 429
L 3 IDENTIFIER 431 434 # 430
L 2 ( 432 434 # 431
L 1 CONSTANT 433 434 # 432
L 1 ) 0 434 # 433
L 4 DECLARE 435 439 # 434
L 3 IDENTIFIER 436 439 # 435
L 2 ( 437 439 # 436
L 1 CONSTANT 438 439 # 437
L 1 , 0 439 # 438
L 3 F 440 443 # 439
L 2 ( 441 443 # 440
L 1 CONSTANT 442 443 # 441
L 1 ) 0 443 # 442
L 3 A 444 447 # 443
L 2 ( 445 447 # 444
L 1 CONSTANT 446 447 # 445
L 1 ) 0 447 # 446
L 3 X 448 451 # 447
L 2 ( 449 451 # 448
L 1 CONSTANT 450 451 # 449
L 1 ) 0 451 # 450
L 3 CHARACTER 452 455 # 451
L 2 ( 453 455 # 452
L 1 CONSTANT 454 455 # 453
L 1 ) 0 455 # 454
L 2 STOP 456 458 # 455
L 1 CONSTANT 457 458 # 456
L 1 ; 0 458 # 457

```

```

L 1 CONSTANT -550 # 458
L 0 <ANY SYMBOL> 460
L 0 <ANY SYMBOL> 1 33
      -999
      ***** BLOCK END *****
L 4 OPEN 462 466 # 461
L 3 FILE 463 466 # 462
L 2 I 464 466 # 463
L 1 LITERAL 465 466 # 464
R 1 I 0 466 # 465
L 4 CLOSE 467 471 # 466
L 3 FILE 468 471 # 467
L 2 I 469 471 # 468
L 1 LITERAL 470 471 # 469
R 1 I 0 471 # 470
L 1 LITERAL 473 471 # 471
L 0 <ANY SYMBOL> 21 475
L 1 LITERAL -565 475 # 473
L 0 <ANY SYMBOL> 1 33
L 0 <ANY SYMBOL> -999
      ***** BLOCK END *****
L 2 LABEL 477 479 # 476
L 1 PROCEDURE 477 479 # 477
L 0 <ANY SYMBOL> -110 479
L 0 <ANY SYMBOL> 2 3
L 0 <ANY SYMBOL> -999
      ***** BLOCK END *****
L 2 LABEL 481 483 # 480
L 1 BEGIN -115 483 # 481
L 0 <ANY SYMBOL> 2 4
L 1 BEGIN -120 485 # 483
L 0 <ANY SYMBOL> 1 4
L 0 <ANY SYMBOL> -999
      ***** BLOCK END *****
L 4 D/I 487 491 # 486
L 3 I 488 491 # 487
L 2 STATEMENT LIST 489 491 # 488
L 1 END 490 491 # 489
R 1 I 0 491 # 490
L 3 ITERATION LIMITER 492 495 # 491
L 2 STATEMENT LIST 493 495 # 492
L 1 END 494 495 # 493
R 1 I 0 495 # 494
L 4 PROCEDURE STATEMENT 496 500 # 495
L 3 I 497 500 # 496
L 2 STATEMENT LIST 498 500 # 497
L 1 END -100 500 # 498
L 0 <ANY SYMBOL> 4 1
L 4 BEGIN STATEMENT 501 505 # 500
L 3 I 502 505 # 501
L 2 STATEMENT LIST 503 505 # 502
L 1 END -105 505 # 503
L 0 <ANY SYMBOL> 4 2
L 0 <ANY SYMBOL> -999
      ***** BLOCK END *****
L 1 TTI -305 508 # 506
L 0 <ANY SYMBOL> 1 20
L 0 <ANY SYMBOL> -999
      ***** BLOCK END *****
L 1 TTO -310 511 # 509
L 0 <ANY SYMBOL> 1 20
L 0 <ANY SYMBOL> -999

```

```

L 1      PTP      ***** BLOCK END *****      514      # 512
L 0      <ANY SYMBOL>      -315      20
L 0      <ANY SYMBOL>      -999
L 1      PTR      ***** BLOCK END *****      517      # 515
L 0      <ANY SYMBOL>      -320      20
L 0      <ANY SYMBOL>      -999
L 1      SEL      ***** BLOCK END *****      520      # 518
L 0      <ANY SYMBOL>      -325      20
L 0      <ANY SYMBOL>      -999
L 2      PUT      ***** BLOCK END *****      524      # 521
L 1      PAGE      522      524
L 1      :      523      524
L 1      PAGE      -330      524
L 0      <ANY SYMBOL>      -999      21
L 0      <ANY SYMBOL>      -999
L 2      PUT      ***** BLOCK END *****      530      # 527
L 1      SKIP      528      530
L 1      :      529      530
L 1      SKIP      -335      530
L 0      <ANY SYMBOL>      -999      21
L 0      <ANY SYMBOL>      -999
L 1      TAB      ***** BLOCK END *****      535      # 533
L 0      <ANY SYMBOL>      -340      21
L 0      <ANY SYMBOL>      -999
L 1      SINGLE      ***** BLOCK END *****      538      # 536
L 0      <ANY SYMBOL>      -680      42
L 0      <ANY SYMBOL>      -999
L 1      DOUBLE      ***** BLOCK END *****      541      # 539
L 0      <ANY SYMBOL>      -685      42
L 0      <ANY SYMBOL>      -999
L 1      F      ***** BLOCK END *****      544      # 542
L 1      f      543      544
L 1      F      -345      544
L 0      <ANY SYMBOL>      -999      21
L 0      <ANY SYMBOL>      -999
L 1      A      ***** BLOCK END *****      549      # 547
L 1      f      548      549
L 1      A      -350      549
L 0      <ANY SYMBOL>      -999      21
L 0      <ANY SYMBOL>      -999
L 2      IDENTIFIER      ***** BLOCK END *****      555      # 552
L 1      :      553      555
L 0      <ANY SYMBOL>      -125      5
L 0      <ANY SYMBOL>      -999
L 2      PROCEDURE STATEMENT      ***** BLOCK END *****      559      # 556
L 1      :      557      559
L 1      STOP      558      559
L 2      BEGIN STATEMENT      560      562

```

```

L 1 1 561 562 # 560
R 1 1 0 562
L 2 563 565 # 562
L 1 564 565 # 563
L 1 565 565 # 564
R 1 566 568 # 565
L 1 567 568 # 566
R 1 568 568 # 567
L 2 569 571 # 568
L 1 570 571 # 569
R 1 571 571 # 570
L 2 572 574 # 571
L 1 573 574 # 572
R 1 574 574 # 573
L 2 575 577 # 574
L 1 576 577 # 575
R 1 577 577 # 576
L 2 578 580 # 577
L 1 579 580 # 578
R 1 580 580 # 579
L 2 581 583 # 580
L 1 582 583 # 581
R 1 583 583 # 582
L 2 584 586 # 583
L 1 585 586 # 584
R 1 586 586 # 585
L 2 587 589 # 586
L 1 588 589 # 587
R 1 589 589 # 588
L 2 590 592 # 589
L 1 591 592 # 590
R 1 592 592 # 591
L 2 593 595 # 592
L 1 594 595 # 593
R 1 595 595 # 594
L 2 596 598 # 595
L 1 597 598 # 596
R 1 598 598 # 597
L 2 599 601 # 598
L 1 600 601 # 599
R 1 601 601 # 600
L 2 602 604 # 601
L 1 603 604 # 602
R 1 604 604 # 603
L 2 605 607 # 604
L 1 606 607 # 605
R 1 607 607 # 606
L 2 608 610 # 607
L 1 609 610 # 608
R 1 610 610 # 609
L 2 611 613 # 610
L 1 612 613 # 611
R 1 613 613 # 612
L 2 614 616 # 613
L 1 615 616 # 614
R 1 616 616 # 615
L 2 617 619 # 616
L 1 618 619 # 617
R 1 619 619 # 618
L 2 620 622 # 619
L 1 621 622 # 620
R 1 622 622 # 621
L 2 623 623 # 622
L 1 624 624 # 623
R 1 625 625 # 624
L 2 626 626 # 625
L 1 627 627 # 626
R 1 628 628 # 627
L 2 629 629 # 628
L 1 630 630 # 629
R 1 631 631 # 630
L 2 632 632 # 631
L 1 633 633 # 632
R 1 634 634 # 633
L 2 635 635 # 634
L 1 636 636 # 635
R 1 637 637 # 636
L 2 638 638 # 637
L 1 639 639 # 638
R 1 640 640 # 639
L 2 641 641 # 640
L 1 642 642 # 641
R 1 643 643 # 642
L 2 644 644 # 643
L 1 645 645 # 644
R 1 646 646 # 645
L 2 647 647 # 646
L 1 648 648 # 647
R 1 649 649 # 648
L 2 650 650 # 649
L 1 651 651 # 650
R 1 652 652 # 651
L 2 653 653 # 652
L 1 654 654 # 653
R 1 655 655 # 654
L 2 656 656 # 655
L 1 657 657 # 656
R 1 658 658 # 657
L 2 659 659 # 658
L 1 660 660 # 659
R 1 661 661 # 660
L 2 662 662 # 661
L 1 663 663 # 662
R 1 664 664 # 663
L 2 665 665 # 664
L 1 666 666 # 665
R 1 667 667 # 666
L 2 668 668 # 667
L 1 669 669 # 668
R 1 670 670 # 669
L 2 671 671 # 670
L 1 672 672 # 671
R 1 673 673 # 672
L 2 674 674 # 673
L 1 675 675 # 674
R 1 676 676 # 675
L 2 677 677 # 676
L 1 678 678 # 677
R 1 679 679 # 678
L 2 680 680 # 679
L 1 681 681 # 680
R 1 682 682 # 681
L 2 683 683 # 682
L 1 684 684 # 683
R 1 685 685 # 684
L 2 686 686 # 685
L 1 687 687 # 686
R 1 688 688 # 687
L 2 689 689 # 688
L 1 690 690 # 689
R 1 691 691 # 690
L 2 692 692 # 691
L 1 693 693 # 692
R 1 694 694 # 693
L 2 695 695 # 694
L 1 696 696 # 695
R 1 697 697 # 696
L 2 698 698 # 697
L 1 699 699 # 698
R 1 700 700 # 699
L 2 701 701 # 700
L 1 702 702 # 701
R 1 703 703 # 702
L 2 704 704 # 703
L 1 705 705 # 704
R 1 706 706 # 705
L 2 707 707 # 706
L 1 708 708 # 707
R 1 709 709 # 708
L 2 710 710 # 709
L 1 711 711 # 710
R 1 712 712 # 711
L 2 713 713 # 712
L 1 714 714 # 713
R 1 715 715 # 714
L 2 716 716 # 715
L 1 717 717 # 716
R 1 718 718 # 717
L 2 719 719 # 718
L 1 720 720 # 719
R 1 721 721 # 720
L 2 722 722 # 721
L 1 723 723 # 722
R 1 724 724 # 723
L 2 725 725 # 724
L 1 726 726 # 725
R 1 727 727 # 726
L 2 728 728 # 727
L 1 729 729 # 728
R 1 730 730 # 729
L 2 731 731 # 730
L 1 732 732 # 731
R 1 733 733 # 732
L 2 734 734 # 733
L 1 735 735 # 734
R 1 736 736 # 735
L 2 737 737 # 736
L 1 738 738 # 737
R 1 739 739 # 738
L 2 740 740 # 739
L 1 741 741 # 740
R 1 742 742 # 741
L 2 743 743 # 742
L 1 744 744 # 743
R 1 745 745 # 744
L 2 746 746 # 745
L 1 747 747 # 746
R 1 748 748 # 747
L 2 749 749 # 748
L 1 750 750 # 749
R 1 751 751 # 750
L 2 752 752 # 751
L 1 753 753 # 752
R 1 754 754 # 753
L 2 755 755 # 754
L 1 756 756 # 755
R 1 757 757 # 756
L 2 758 758 # 757
L 1 759 759 # 758
R 1 760 760 # 759
L 2 761 761 # 760
L 1 762 762 # 761
R 1 763 763 # 762
L 2 764 764 # 763
L 1 765 765 # 764
R 1 766 766 # 765
L 2 767 767 # 766
L 1 768 768 # 767
R 1 769 769 # 768
L 2 770 770 # 769
L 1 771 771 # 770
R 1 772 772 # 771
L 2 773 773 # 772
L 1 774 774 # 773
R 1 775 775 # 774
L 2 776 776 # 775
L 1 777 777 # 776
R 1 778 778 # 777
L 2 779 779 # 778
L 1 780 780 # 779
R 1 781 781 # 780
L 2 782 782 # 781
L 1 783 783 # 782
R 1 784 784 # 783
L 2 785 785 # 784
L 1 786 786 # 785
R 1 787 787 # 786
L 2 788 788 # 787
L 1 789 789 # 788
R 1 790 790 # 789
L 2 791 791 # 790
L 1 792 792 # 791
R 1 793 793 # 792
L 2 794 794 # 793
L 1 795 795 # 794
R 1 796 796 # 795
L 2 797 797 # 796
L 1 798 798 # 797
R 1 799 799 # 798
L 2 800 800 # 799
L 1 801 801 # 800
R 1 802 802 # 801
L 2 803 803 # 802
L 1 804 804 # 803
R 1 805 805 # 804
L 2 806 806 # 805
L 1 807 807 # 806
R 1 808 808 # 807
L 2 809 809 # 808
L 1 810 810 # 809
R 1 811 811 # 810
L 2 812 812 # 811
L 1 813 813 # 812
R 1 814 814 # 813
L 2 815 815 # 814
L 1 816 816 # 815
R 1 817 817 # 816
L 2 818 818 # 817
L 1 819 819 # 818
R 1 820 820 # 819
L 2 821 821 # 820
L 1 822 822 # 821
R 1 823 823 # 822
L 2 824 824 # 823
L 1 825 825 # 824
R 1 826 826 # 825
L 2 827 827 # 826
L 1 828 828 # 827
R 1 829 829 # 828
L 2 830 830 # 829
L 1 831 831 # 830
R 1 832 832 # 831
L 2 833 833 # 832
L 1 834 834 # 833
R 1 835 835 # 834
L 2 836 836 # 835
L 1 837 837 # 836
R 1 838 838 # 837
L 2 839 839 # 838
L 1 840 840 # 839
R 1 841 841 # 840
L 2 842 842 # 841
L 1 843 843 # 842
R 1 844 844 # 843
L 2 845 845 # 844
L 1 846 846 # 845
R 1 847 847 # 846
L 2 848 848 # 847
L 1 849 849 # 848
R 1 850 850 # 849
L 2 851 851 # 850
L 1 852 852 # 851
R 1 853 853 # 852
L 2 854 854 # 853
L 1 855 855 # 854
R 1 856 856 # 855
L 2 857 857 # 856
L 1 858 858 # 857
R 1 859 859 # 858
L 2 860 860 # 859
L 1 861 861 # 860
R 1 862 862 # 861
L 2 863 863 # 862
L 1 864 864 # 863
R 1 865 865 # 864
L 2 866 866 # 865
L 1 867 867 # 866
R 1 868 868 # 867
L 2 869 869 # 868
L 1 870 870 # 869
R 1 871 871 # 870
L 2 872 872 # 871
L 1 873 873 # 872
R 1 874 874 # 873
L 2 875 875 # 874
L 1 876 876 # 875
R 1 877 877 # 876
L 2 878 878 # 877
L 1 879 879 # 878
R 1 880 880 # 879
L 2 881 881 # 880
L 1 882 882 # 881
R 1 883 883 # 882
L 2 884 884 # 883
L 1 885 885 # 884
R 1 886 886 # 885
L 2 887 887 # 886
L 1 888 888 # 887
R 1 889 889 # 888
L 2 890 890 # 889
L 1 891 891 # 890
R 1 892 892 # 891
L 2 893 893 # 892
L 1 894 894 # 893
R 1 895 895 # 894
L 2 896 896 # 895
L 1 897 897 # 896
R 1 898 898 # 897
L 2 899 899 # 898
L 1 900 900 # 899
R 1 901 901 # 900
L 2 902 902 # 901
L 1 903 903 # 902
R 1 904 904 # 903
L 2 905 905 # 904
L 1 906 906 # 905
R 1 907 907 # 906
L 2 908 908 # 907
L 1 909 909 # 908
R 1 910 910 # 909
L 2 911 911 # 910
L 1 912 912 # 911
R 1 913 913 # 912
L 2 914 914 # 913
L 1 915 915 # 914
R 1 916 916 # 915
L 2 917 917 # 916
L 1 918 918 # 917
R 1 919 919 # 918
L 2 920 920 # 919
L 1 921 921 # 920
R 1 922 922 # 921
L 2 923 923 # 922
L 1 924 924 # 923
R 1 925 925 # 924
L 2 926 926 # 925
L 1 927 927 # 926
R 1 928 928 # 927
L 2 929 929 # 928
L 1 930 930 # 929
R 1 931 931 # 930
L 2 932 932 # 931
L 1 933 933 # 932
R 1 934 934 # 933
L 2 935 935 # 934
L 1 936 936 # 935
R 1 937 937 # 936
L 2 938 938 # 937
L 1 939 939 # 938
R 1 940 940 # 939
L 2 941 941 # 940
L 1 942 942 # 941
R 1 943 943 # 942
L 2 944 944 # 943
L 1 945 945 # 944
R 1 946 946 # 945
L 2 947 947 # 946
L 1 948 948 # 947
R 1 949 949 # 948
L 2 950 950 # 949
L 1 951 951 # 950
R 1 952 952 # 951
L 2 953 953 # 952
L 1 954 954 # 953
R 1 955 955 # 954
L 2 956 956 # 955
L 1 957 957 # 956
R 1 958 958 # 957
L 2 959 959 # 958
L 1 960 960 # 959
R 1 961 961 # 960
L 2 962 962 # 961
L 1 963 963 # 962
R 1 964 964 # 963
L 2 965 965 # 964
L 1 966 966 # 965
R 1 967 967 # 966
L 2 968 968 # 967
L 1 969 969 # 968
R 1 970 970 # 969
L 2 971 971 # 970
L 1 972 972 # 971
R 1 973 973 # 972
L 2 974 974 # 973
L 1 975 975 # 974
R 1 976 976 # 975
L 2 977 977 # 976
L 1 978 978 # 977
R 1 979 979 # 978
L 2 980 980 # 979
L 1 981 981 # 980
R 1 982 982 # 981
L 2 983 983 # 982
L 1 984 984 # 983
R 1 985 985 # 984
L 2 986 986 # 985
L 1 987 987 # 986
R 1 988 988 # 987
L 2 989 989 # 988
L 1 990 990 # 989
R 1 991 991 # 990
L 2 992 992 # 991
L 1 993 993 # 992
R 1 994 994 # 993
L 2 995 995 # 994
L 1 996 996 # 995
R 1 997 997 # 996
L 2 998 998 # 997
L 1 999 999 # 998
R 1 1000 1000 # 999

```

```

L 1 1 ; SWAP 621 # 620
R 1 1 SWAP 622 # 621
L 2 1 BEGIN STATEMENT 623 # 622
L 1 1 ; 625 # 623
L 1 1 ; SWAP 624 # 624
L 2 1 DO 626 # 625
L 1 1 ; 627 # 626
L 1 1 SWAP 628 # 627
L 2 1 PROCEDURE STATEMENT 631 # 628
L 1 1 ; 630 # 629
L 1 1 OR 631 # 630
L 2 1 BEGIN STATEMENT 632 # 631
L 1 1 ; 633 # 632
L 1 1 OR 634 # 633
L 2 1 DO 635 # 634
L 1 1 ; 636 # 635
L 1 1 OR 637 # 636
L 2 1 PROCEDURE STATEMENT 640 # 637
L 1 1 ; 638 # 638
L 1 1 AND 640 # 639
L 2 1 BEGIN STATEMENT 641 # 640
L 1 1 ; 642 # 641
L 1 1 AND 643 # 642
L 2 1 DO 644 # 643
L 1 1 ; 645 # 644
L 1 1 AND 646 # 645
L 2 1 PROCEDURE STATEMENT 647 # 646
L 1 1 ; 648 # 647
L 1 1 LENGTH 649 # 648
L 2 1 BEGIN STATEMENT 650 # 649
L 1 1 ; 651 # 650
L 1 1 DO 652 # 651
L 2 1 LENGTH 653 # 652
L 1 1 ; 654 # 653
L 1 1 LENGTH 655 # 654
L 2 1 PROCEDURE STATEMENT 656 # 655
L 1 1 ; 657 # 656
L 1 1 INDEX 658 # 657
L 2 1 BEGIN STATEMENT 659 # 658
L 1 1 ; 660 # 659
L 1 1 INDEX 661 # 660
L 2 1 DO 662 # 661
L 1 1 ; 663 # 662
L 1 1 INDEX 664 # 663
L 2 1 PROCEDURE STATEMENT 665 # 664
L 1 1 ; 666 # 665
L 1 1 SUBSTR 667 # 666
L 2 1 BEGIN STATEMENT 668 # 667
L 1 1 ; 669 # 668
L 1 1 SUBSTR 670 # 669
L 2 1 DO 671 # 670
L 1 1 ; 672 # 671
L 1 1 SUBSTR 673 # 672
L 2 1 PROCEDURE STATEMENT 674 # 673
L 1 1 ; 675 # 674
L 1 1 IF 676 # 675
L 2 1 BEGIN STATEMENT 677 # 676
L 1 1 ; 678 # 677
L 1 1 IF 679 # 678
L 2 1 DO 680 # 679
682 # 682

```


L 1	;	681	682	# 680
R 1	IF	0	682	# 681
L 2	PROCEDURE STATEMENT	683	685	# 682
L 1	;	684	685	# 683
R 1	WRITE	0	685	# 684
L 2	BEGIN STATEMENT	686	688	# 685
L 1	;	687	688	# 686
R 1	WRITE	0	688	# 687
L 2	DO	689	691	# 688
L 1	;	690	691	# 689
R 1	WRITE	0	691	# 690
L 2	PROCEDURE STATEMENT	692	694	# 691
L 1	;	693	694	# 692
R 1	READ	0	694	# 693
L 2	BEGIN STATEMENT	695	697	# 694
L 1	;	696	697	# 695
R 1	READ	0	697	# 696
L 2	DO	698	700	# 697
L 1	;	699	700	# 698
R 1	READ	0	700	# 699
L 2	PROCEDURE STATEMENT	701	703	# 700
L 1	;	702	703	# 701
R 1	PUT	0	703	# 702
L 2	BEGIN STATEMENT	704	706	# 703
L 1	;	705	706	# 704
R 1	PUT	0	706	# 705
L 2	DO	707	709	# 706
L 1	;	708	709	# 707
R 1	PUT	0	709	# 708
L 2	PROCEDURE STATEMENT	710	712	# 709
L 1	;	711	712	# 710
R 1	GET	0	712	# 711
L 2	BEGIN STATEMENT	713	715	# 712
L 1	;	714	715	# 713
R 1	GET	0	715	# 714
L 2	DO	716	718	# 715
L 1	;	717	718	# 716
R 1	GET	0	718	# 717
L 2	PROCEDURE STATEMENT	719	721	# 718
L 1	;	720	721	# 719
R 1	DO	0	721	# 720
L 2	BEGIN STATEMENT	722	724	# 721
L 1	;	723	724	# 722
R 1	DO	0	724	# 723
L 2	DO	725	727	# 724
L 1	;	726	727	# 725
R 1	DO	0	727	# 726
L 2	PROCEDURE STATEMENT	728	730	# 727
L 1	;	729	730	# 728
R 1	GO	0	730	# 729
L 2	BEGIN STATEMENT	731	733	# 730
L 1	;	732	733	# 731
R 1	GO	0	733	# 732
L 2	DO	734	736	# 733
L 1	;	735	736	# 734
R 1	GO	0	736	# 735
L 2	PROCEDURE STATEMENT	737	739	# 736
L 1	;	738	739	# 737
R 1	RETURN	0	739	# 738
L 2	BEGIN STATEMENT	740	742	# 739

L 1	;	741	742	# 740
R 1	RETURN	0	742	# 741
L 2	DO	743	745	# 742
L 1	;	744	745	# 743
R 1	RETURN	0	745	# 744
L 2	PROCEDURE STATEMENT	746	748	# 745
L 1	;	747	748	# 746
R 1	BEGIN	0	748	# 747
L 2	BEGIN STATEMENT	749	751	# 748
L 1	;	750	751	# 749
R 1	BEGIN	0	751	# 750
L 2	DO	752	754	# 751
L 1	;	753	754	# 752
R 1	BEGIN	0	754	# 753
L 2	PROCEDURE STATEMENT	755	757	# 754
L 1	;	756	757	# 755
R 1	LITERAL	0	757	# 756
L 2	BEGIN STATEMENT	758	760	# 757
L 1	;	759	760	# 758
R 1	LITERAL	0	760	# 759
L 2	DO	761	763	# 760
L 1	;	762	763	# 761
R 1	LITERAL	0	763	# 762
L 2	PROCEDURE STATEMENT	764	766	# 763
L 1	;	765	766	# 764
R 1	CONSTANT	0	766	# 765
L 2	BEGIN STATEMENT	767	769	# 766
L 1	;	768	769	# 767
R 1	CONSTANT	0	769	# 768
L 2	DO	770	772	# 769
L 1	;	771	772	# 770
R 1	CONSTANT	0	772	# 771
L 2	PROCEDURE STATEMENT	773	775	# 772
L 1	;	774	775	# 773
R 1	IDENTIFIER	0	775	# 774
L 2	BEGIN STATEMENT	776	778	# 775
L 1	;	777	778	# 776
R 1	IDENTIFIER	0	778	# 777
L 2	DO	779	781	# 778
L 1	;	780	781	# 779
R 1	IDENTIFIER	0	781	# 780
L 5	DO	782	787	# 781
L 4	;	783	787	# 782
L 3	STATEMENT LIST	784	787	# 783
L 2	END	785	787	# 784
L 1	;	-107	787	# 785
L 0	<ANY SYMBOL>	5	51	# 786
L 5	LABEL	788	793	# 787
L 4	(	789	793	# 788
L 3	IDENTIFIER	790	793	# 789
L 2	)	791	793	# 790
L 1	;	-197	793	# 791
L 0	<ANY SYMBOL>	5	7	# 792
L 4	GO	794	798	# 793
L 3	TO	795	798	# 794
L 2	IDENTIFIER	796	798	# 795
L 1	;	-140	798	# 796
L 0	<ANY SYMBOL>	4	7	# 797
L 4	ITERATION LIMITER	799	803	# 798
L 3	STATEMENT LIST	800	803	# 799

L 2	END	801	803	# 800
L 1	:	-230	803	# 801
L 0	<ANY SYMBOL>	4	11	
L 3	CALL	804	807	# 803
L 2	IDENTIFIER	805	807	# 804
L 1	:	-193	807	# 805
L 0	<ANY SYMBOL>	3	7	
L 3	STOP	808	811	# 807
L 2	CONSTANT	809	811	# 808
L 1	:	-198	811	# 809
L 0	<ANY SYMBOL>	3	7	
L 3	I/O S LIST	812	815	# 811
L 2	:	813	815	# 812
L 1	:	-270	815	# 813
L 0	<ANY SYMBOL>	3	15	
L 3	PUT	816	819	# 815
L 2	SKIP	817	819	# 816
L 1	:	-271	819	# 817
L 0	<ANY SYMBOL>	3	15	
L 3	PUT	820	823	# 819
L 2	PAGE	821	823	# 820
L 1	:	-272	823	# 821
L 0	<ANY SYMBOL>	3	15	
L 3	FORMAT S LIST	824	827	# 823
L 2	:	825	827	# 824
L 1	:	-275	827	# 825
L 0	<ANY SYMBOL>	3	15	
L 3	RETURN	828	830	# 827
L 2	:	-145	830	# 828
L 1	:	831	833	# 830
L 0	<ANY SYMBOL>	-150	833	# 831
L 2	ASSIGNMENT STATEMENT	834	836	# 833
L 1	:	-155	836	# 834
L 0	<ANY SYMBOL>	2	7	
L 2	DO STATEMENT	837	839	# 836
L 1	:	-160	839	# 837
L 0	<ANY SYMBOL>	2	7	
L 2	I/O STREAM STATEMENT	840	842	# 839
L 1	:	-165	842	# 840
L 0	<ANY SYMBOL>	2	7	
L 2	I/O RECORD STATEMENT	843	845	# 842
L 1	:	-170	845	# 843
L 0	<ANY SYMBOL>	2	7	
L 2	OPEN CLOSE STATEMENT	846	848	# 845
L 1	:	-175	848	# 846
L 0	<ANY SYMBOL>	2	7	
L 2	DECLARATION STMT	849	851	# 848
L 1	:	-180	851	# 849
L 0	<ANY SYMBOL>	2	7	
L 2	PROCEDURE BLOCK	852	854	# 851
L 1	:	-185	854	# 852
L 0	<ANY SYMBOL>	2	7	
L 2	BEGIN BLOCK	855	857	# 854
L 1	:	-190	857	# 855
L 0	<ANY SYMBOL>	2	7	
L 2	LABEL	858	860	# 857
L 1	:	-196	860	# 858
L 0	<ANY SYMBOL>	2	7	

LINE	TEXT	ADDRESS	HEX	ASCII
L 2	STOP	861	# 860	
L 1	;	-199	# 861	
L 0	<ANY SYMBOL>	2		
L 0	<ANY SYMBOL>	-999		
	***** BLOCK END *****			
L 5	READ	865	# 864	
L 4	FILE	866	# 865	
L 3	(	867	# 866	
L 2	VARIABLE	868	# 867	
L 1	)	869	# 868	
R 1	KEY	0	# 869	
L 5	WRITE	871	# 870	
L 4	FILE	872	# 871	
L 3	(	873	# 872	
L 2	VARIABLE	874	# 873	
L 1	)	875	# 874	
R 1	KEY	0	# 875	
L 4	LABEL	877	# 876	
L 3	(	878	# 877	
L 2	IDENTIFIER	879	# 878	
L 1	)	880	# 879	
P 1	;	0	# 880	
L 2	I/O S LIST	882	# 881	
L 1	(	883	# 882	
R 1	(	0	# 883	
L 2	I/O S LIST	885	# 884	
L 1	)	886	# 885	
R 1	;	0	# 886	
L 2	FWT S LIST	888	# 887	
L 1	)	889	# 888	
P 1	;	0	# 889	
L 9	OPEN	891	# 890	
L 8	FILE	892	# 891	
L 7	(	893	# 892	
L 6	LITERAL	894	# 893	
L 5	,	895	# 894	
L 4	VARIABLE	896	# 895	
L 3	,	897	# 896	
L 2	VARIABLE	898	# 897	
L 1	)	-580	# 898	
L 0	<ANY SYMBOL>	9	# 900	
L 9	CLOSE	901	# 901	
L 8	FILE	902	# 902	
L 7	(	903	# 903	
L 6	LITERAL	904	# 904	
L 5	,	905	# 905	
L 4	VARIABLE	906	# 906	
L 3	,	907	# 907	
L 2	VARIABLE	908	# 908	
L 1	)	-590	# 909	
L 0	<ANY SYMBOL>	9	# 910	
L 9	SURST	911	# 911	
L 7	(	912	# 912	
L 6	VARIABLE	913	# 913	
L 5	,	914	# 914	
L 4	EXPRESSION	915	# 915	
L 3	,	916	# 916	
L 2	EXPRESSION	917	# 917	
L 1	)	-400	# 917	
L 0	<ANY SYMBOL>	8	# 917	

L 7	DECLARE	920	927	# 919
L 6	IDENTIFIER	921	927	# 920
L 5	(	922	927	# 921
L 4	CONSTANT	923	927	# 922
L 3	,	924	927	# 923
L 2	CONSTANT	925	927	# 924
L 1	)	-675	927	# 925
L 0	<ANY SYMBOL>	7	49	
L 6	INDEX	928	934	# 927
L 5	(	929	934	# 928
L 4	VARIABLE	930	934	# 929
L 3	,	931	934	# 930
L 2	VARIABLE	932	934	# 931
L 1	)	-405	934	# 932
L 0	<ANY SYMBOL>	6	22	
L 6	AND	935	941	# 934
L 5	(	936	941	# 935
L 4	VARIABLE	937	941	# 936
L 3	,	938	941	# 937
L 2	PRIMARY	939	941	# 938
L 1	)	-415	941	# 939
L 0	<ANY SYMBOL>	6	22	
L 6	OR	942	948	# 941
L 5	(	943	948	# 942
L 4	VARIABLE	944	948	# 943
L 3	,	945	948	# 944
L 2	PRIMARY	946	948	# 945
L 1	)	-420	948	# 946
L 0	<ANY SYMBOL>	6	22	
L 6	IDENTIFIER	949	955	# 948
L 5	(	950	955	# 949
L 4	EXPRESSION	951	955	# 950
L 3	,	952	955	# 951
L 2	EXPRESSION	953	955	# 952
L 1	)	-460	955	# 953
L 0	<ANY SYMBOL>	6	26	
L 5	GET	956	961	# 955
L 4	FILE	957	961	# 956
L 3	(	958	961	# 957
L 2	DEVICE	959	961	# 958
L 1	)	-300	961	# 959
L 0	<ANY SYMBOL>	5	19	
L 5	PUT	962	967	# 961
L 4	FILE	963	967	# 962
L 3	(	964	967	# 963
L 2	DEVICE	965	967	# 964
L 1	)	-301	967	# 965
L 0	<ANY SYMBOL>	5	19	
L 5	DECLARE	968	973	# 967
L 4	IDENTIFIER	969	973	# 968
L 3	(	970	973	# 969
L 2	CONSTANT	971	973	# 970
L 1	)	-670	973	# 971
L 0	<ANY SYMBOL>	5	49	
L 4	F	974	978	# 973
L 3	(	975	978	# 974
L 2	CONSTANT	976	978	# 975
L 1	)	-355	978	# 976
L 0	<ANY SYMBOL>	4	21	
L 4	A	979	983	# 978

L 3	(	CONSTANT	980	983	# 979
L 2	)		981	983	# 980
L 1			-360	983	# 981
L 0	<ANY SYMBOL>		4	21	
L 4	x		984	988	# 983
L 3	(		985	988	# 984
L 2	CONSTANT		986	988	# 985
L 1	)		-365	988	# 986
L 0	<ANY SYMBOL>		4	21	
L 4	LENGTH		989	993	# 988
L 3	(		990	993	# 989
L 2	VARIABLE		991	993	# 990
L 1	)		-410	993	# 991
L 0	<ANY SYMBOL>		4	22	
L 4	SWAP		994	998	# 993
L 3	(		995	998	# 994
L 2	VARIABLE		996	998	# 995
L 1	)		-425	998	# 996
L 0	<ANY SYMBOL>		4	22	
L 4	IDENTIFIER		999	1003	# 998
L 3	(		1000	1003	# 999
L 2	EXPRESSION		1001	1003	#1000
L 1	)		-455	1003	#1001
L 0	<ANY SYMBOL>		4	26	
L 4	KEY		1004	1008	#1003
L 3	(		1005	1008	#1004
L 2	VARIABLE		1006	1008	#1005
L 1	)		-630	1008	#1006
L 0	<ANY SYMBOL>		4	38	
L 4	CHARACTER		1009	1013	#1008
L 3	(		1010	1013	#1009
L 2	CONSTANT		1011	1013	#1010
L 1	)		-690	1013	#1011
L 0	<ANY SYMBOL>		4	42	
L 3	(		1014	1017	#1013
L 2	EXPRESSION		1015	1017	#1014
L 1	)		-560	1017	#1015
L 0	<ANY SYMBOL>		3	33	
L 0	<ANY SYMBOL>		-999	-999	
L 1	+	***** BLOCK END *****	-730	1020	#1018
L 0	<ANY SYMBOL>		1	46	
L 0	<ANY SYMBOL>		-999	-999	
L 1	-	***** BLOCK END *****	-735	1023	#1021
L 0	<ANY SYMBOL>		1	46	
L 0	<ANY SYMBOL>		-999	-999	
L 1	*	***** BLOCK END *****	-740	1026	#1024
L 0	<ANY SYMBOL>		1	47	
L 0	<ANY SYMBOL>		-999	-999	
L 1	/	***** BLOCK END *****	-745	1029	#1027
L 0	<ANY SYMBOL>		1	47	
L 0	<ANY SYMBOL>		-999	-999	
L 1	**	***** BLOCK END *****	-750	1032	#1030
L 0	<ANY SYMBOL>		1	48	
L 0	<ANY SYMBOL>		-999	-999	
L 0	***** BLOCK END *****				

L 2	PRIMARY	1090	1092	#1089
L 1	=	1091	1092	#1090
R 1	-	0	1092	#1091
L 2	PRIMARY	1093	1095	#1092
L 1	=	1094	1095	#1093
R 1	+	0	1095	#1094
L 2	PRIMARY	1096	1098	#1095
L 1	=	1097	1098	#1096
R 1	{	0	1098	#1097
L 2	PRIMARY	1099	1101	#1098
L 1	=	1100	1101	#1099
R 1	SWAP	0	1101	#1100
L 2	CHARACTER SEGMENT	1102	1104	#1101
L 1	=	1103	1104	#1102
R 1	SWAP	0	1104	#1103
L 2	PRIMARY	1105	1107	#1104
L 1	=	1106	1107	#1105
R 1	OR	0	1107	#1106
L 2	CHARACTER SEGMENT	1108	1110	#1107
L 1	=	1109	1110	#1108
R 1	OR	0	1110	#1109
L 2	PRIMARY	1111	1113	#1110
L 1	=	1112	1113	#1111
R 1	AND	0	1113	#1112
L 2	CHARACTER SEGMENT	1114	1116	#1113
L 1	=	1115	1116	#1114
R 1	AND	0	1116	#1115
L 2	PRIMARY	1117	1119	#1116
L 1	=	1118	1119	#1117
R 1	LENGTH	0	1119	#1118
L 2	CHARACTER SEGMENT	1120	1122	#1119
L 1	=	1121	1122	#1120
R 1	LENGTH	0	1122	#1121
L 2	PRIMARY	1123	1125	#1122
L 1	=	1124	1125	#1123
R 1	INDEX	0	1125	#1124
L 2	CHARACTER SEGMENT	1126	1128	#1125
L 1	=	1127	1128	#1126
R 1	INDEX	0	1128	#1127
L 2	PRIMARY	1129	1131	#1128
L 1	=	1130	1131	#1129
R 1	SURSTR	0	1131	#1130
L 2	CHARACTER SEGMENT	1132	1134	#1131
L 1	=	1133	1134	#1132
R 1	SURSTR	0	1134	#1133
L 2	PRIMARY	1135	1137	#1134
L 1	=	1136	1137	#1135
R 1	LITERAL	0	1137	#1136
L 2	PRIMARY	1138	1140	#1137
L 1	=	1139	1140	#1138
R 1	CONSTANT	0	1140	#1139
L 2	CHARACTER SEGMENT	1141	1143	#1140
L 1	=	1142	1143	#1141
R 1	IDENTIFIER	0	1143	#1142
L 2	PRIMARY	1144	1146	#1143
L 1	=	1145	1146	#1144
R 1	IDENTIFIER	0	1146	#1145
L 1	=	-725	1148	#1146
L 0	<ANY SYMBOL>	1	45	
L 0	<ANY SYMBOL>	-999	-999	

L 1	≠	***** BLOCK END *****	1151	#1149
L 0	<ANY SYMBOL>	-726	45	
L 0	<ANY SYMBOL>	1	-999	
	***** BLOCK END *****			
L 3	IF	1153	1156	#1152
L 2	LOGICAL EXPRESSION	1154	1156	#1153
L 1	THEN	-210	1156	#1154
L 0	<ANY SYMBOL>	3	9	
L 0	<ANY SYMBOL>	-999	-999	
	***** BLOCK END *****			
L 3	IF CLAUSE	1158	1161	#1157
L 2	STATEMENT	1159	1161	#1158
L 1	ELSE	-215	1161	#1159
L 0	<ANY SYMBOL>	3	10	
L 0	<ANY SYMBOL>	-999	-999	
	***** BLOCK END *****			

APPENDIX IV

Sample Generated Code

```
CRANK:  PROCEDURE;  
        DECLARE  KK(20)  DOUBLE;  
        KK( KK( KK(10) ) ) = KK(5);  
        END;
```

Sample 1 Input.

```

.TITL CRANK
.NREL
.ENT START
.TXTM 1
.RDX 10.

.DATA: .BLK 41
;CONSTANT SECTION
C1008: 5
C1004: 10
C1003: 20
.LTRL: .BLK 0
START: JMPSR

                                L1001
                                JUMP
                                L1002
L1001: SUB1
                                .DATA+ 0
                                2
                                L1005
                                C1004
                                20
                                CDPSP
L1005: 0
                                .DATA+ 40
                                SUB1
                                .DATA+ 0
                                2
                                L1006
                                .DATA+ 40
                                20
                                CDPSP
L1006: 0
                                .DATA+ 40
                                SUB1
                                .DATA+ 0
                                2
                                L1007
                                .DATA+ 40
                                20
                                SUB1
                                .DATA+ 0
                                2
                                L1009
                                C1008
                                20
                                MOVE
L1009: 0
L1007: 0
                                RETRN
L1002: RETRN

```

Sample 1 Generated Code.

```
CRANK:  PROCEDURE;  
        DECLARE I  SINGLE;  
        I = 10;  
/* % */  I = 20;      /* % */  
/* @     I = 30;      @ */  
        END;
```

Sample 2 Input.

```

                                .TITL CRANK
                                .NREL
                                .ENT START
                                .TXTM 1
                                .RDX 10.

.DATA: .BLK 1
;CONSTANT SECTION
C1004: 30
C1003: 10
.LTRL: .BLK 0
START: JMPSR

                                L1001
                                JUMP
                                L1002
L1001:
                                MOVE
                                C1003
                                .DATA+ 0
                                MOVE
                                C1004
                                .DATA+ 0
                                RETRN
L1002:
                                RETRN
                                .END

```

Sample 2 Generated Code.

```
CRANK:  PROCEDURE;  
        DECLARE II SINGLE;  
        DECLARE JJ DOUBLE;  
        II = 20;  
        JJ = 30;  
        II = JJ;  
        END;
```

Sample 3 Input.

```

                                .TITL CRANK
                                .NREL
                                .ENT START
                                .TXTM 1
                                .RDX 10
.DATA:      .BLK      5
;CONSTANT SECTION
C1004:      30
C1003:      20
.LTRL:      .BLK      0
START:      JMPSR
                                L1001
                                JUMP
                                L1002
L1001:      MOVE
                                C1003
                                .DATA+ 0
                                CSPDP
                                C1004
                                .DATA+ 3
                                MOVED
                                .DATA+ 3
                                .DATA+ 1
                                CSPDP
                                .DATA+ 1
                                .DATA+ 3
                                MOVE
                                .DATA+ 3
                                .DATA+ 0
                                RETRN
L1002:      RETRN
                                .END

```

Sample 3 Generated Code.

APPENDIX V

Run Time Routines

Following are brief descriptions of system calls appearing in the generated code samples of Appendix IV.

JMPSE	Lxxxx	; Call a subroutine. Save return information. ; Transfer address
JUMP	Lxxxx	; Simple transfer. No information saved. ; Transfer address
SUBL	.DATA+xxx xx Lxxxx .DATA+xxx xx	; Compute address of singly subscripted variable. ; Array starting address ; Constant element length ; Location to store computed address ; Address of subscript value (single precision) ; Constant, first subscript limit
CDPSP	.DATA+xxx .DATA+xxx	; Convert single precision to double precision ; Single precision variable address ; Single precision variable address
MOVE	.DATA+xxx .DATA+xxx	; Move a word ; Source address ; Destination address
RETRN		; Return to calling routine. Pop stack.
MOVED	.DATA+xxx .DATA+xxx	; Move double words. ; Source address ; Destination address
CSPDP	.DATA+xxx .DATA+xxx	; Convert single precision to double precision ; Single precision address ; Double precision address

APPENDIX VI

Compiler Listing


```

/* TSTK SIZE (=20) */
/* REDUCTIONS TABLE */
DECLARE RDTBL(1200,3) BIN FIXED(15);
/* REDUCTIONS TABLE */
DECLARE RDTBL BIN FIXED(15);
/* RED TABLE POINTER */
DECLARE (RED1,RED2,RED3) BIN FIXED(15);
/* RED TABLE SIZE (=1000) */
DECLARE RDTBL BIN FIXED(15);
/* CODED OUTPUT */
DECLARE CODES(100,3) BIN FIXED(15);
DECLARE CODED OUTOUT */
DECLARE CODED OUTOUT BIN FIXED(15);
/* CODES POINTER */
DECLARE (OPTYP,OPVAL,OPLNKT) BIN FIXED(15);
/* SATTNCS */
DECLARE CLASS1 BIN FIXED(15);
/* VARIABLE CLASS */
DECLARE CLASS2 BIN FIXED(15);
/* LITERAL CLASS */
DECLARE CLASS3 BIN FIXED(15);
/* CONSTANT CLASS */
DECLARE (PR,SM) BINARY FIXED(15);
/* RIGHT CONTEXT */
DECLARE (PRD,SMO) BIN FIXED(15);
/*
DECLARE STKO BIN FIXED(15);
/* END STATUS */
DECLARE (SW1, /* BLANK IGNORE */
SW2, /* LOOK AHEAD IN SCANNER */
SW3,
SW4, /* ON IFF PROCESSING A COMMENT */
SW5, /* ON IFF PROCESSING A LITERAL */
SW6, /* ON IFF PROCESSING AN INCLUSION */ )
BINARY FIXED(15);
/* ERRORS */
DECLARE EPS(20) BIN FIXED(15);
/* FROM VECTOR */
DECLARE FRPTR BIN FIXED(15);
/* LAST USED POS */
DECLARE ESTP BIN FIXED(15);
/* RES SIZE (=20) */
DECLARE GNUM(10) CHAR(1);
DECLARE CNST BIN FIXED(15);
DECLARE ASSIGH CONST AREA */
DECLARE LTRL BIN FIXED(15);
DECLARE ASSIGH LTRL AREA */
DECLARE DATA BIN FIXED(15);
DECLARE ASSIGH VARBL AREA */
DECLARE UNIOE BIN FIXED(15);
DECLARE NAME CHAR(10);
DECLARE FATAL BIN FIXED(15);
DECLARE CDBH BIN FIXED(15);
DECLARE FSTPH BIN FIXED(15);
DECLARE FLNK BIN FIXED(15);
DECLARE FRTR BIN FIXED(15);
DECLARE CLVL BIN FIXED(15);
DECLARE (I,J,K,L,M,N) BINARY FIXED(15);

```

1 22
1 22
1 22
1 23
1 23
1 24
1 24
1 25
1 26
1 26
1 26
1 27
1 27
1 28
1 28
1 29
1 29
1 30
1 30
1 31
1 31
1 32
1 32
1 33
1 33
1 34
1 34
1 35
1 35
1 35
1 35
1 35
1 35
1 36
1 36
1 37
1 37
1 38
1 38
1 39
1 39
1 40
1 41
1 41
1 42
1 42
1 43
1 43
1 44
1 45
1 46
1 47
1 48
1 49
1 50
1 51

KSU'S PL/I NEATENFR AND PRECOMPILER

4

PAGE

```

DECLARE TMPTB SINGLE;
DECLARE TS125 SINGLE;
DECLARE PTRBL(1200,3) SINGLE;
DECLARE REDPT SINGLE;
DECLARE PGO1 SINGLE;
DECLARE PGO2 SINGLE;
DECLARE PGO3 SINGLE;
DECLARE PGO4 SINGLE;
DECLARE CODES(100,3) SINGLE;
DECLARE COOPT SINGLE;
DECLARE COPTP SINGLE;
DECLARE COVAL SINGLE;
DECLARE OPLNK SINGLE;
DECLARE CLASS1 SINGLE;
DECLARE CLASS2 SINGLE;
DECLARE CLASS3 SINGLE;
DECLARE PR SINGLE;
DECLARE SM SINGLE;
DECLARE PRO SINGLE;
DECLARE SMO SINGLE;
DECLARE STKO SINGLE;
DECLARE SW1 SINGLE;
DECLARE SW2 SINGLE;
DECLARE SW3 SINGLE;
DECLARE SW4 SINGLE;
DECLARE SW5 SINGLE;
DECLARE SW6 SINGLE;
DECLARE POS(20) SINGLE;
DECLARE COOPTO SINGLE;
DECLARE RSIZE SINGLE;
DECLARE CHUM(10) CHARACTER(1);
DECLARE CNST SINGLE;
DECLARE LTOL SINGLE;
DECLARE DATA SINGLE;
DECLARE UNTOE SINGLE;
DECLARE NAME CHARACTER(10);
DECLARE CCRH SINGLE;
DECLARE FSTRH SINGLE;
DECLARE FLNK SINGLE;
DECLARE FOTO SINGLE;
DECLARE CLVI SINGLE;
DECLARE T SINGLE;
DECLARE J SINGLE;
DECLARE K SINGLE;
DECLARE L SINGLE;
DECLARE M SINGLE;
DECLARE N SINGLE;
DECLARE IRL SINGLE;
DECLARE GETW SINGLE;
DECLARE VALUE SINGLE;
DECLARE LOT SINGLE;
DECLARE CH1 CHARACTER(1);
DECLARE FOF SINGLE;
DECLARE J1 SINGLE;
DECLARE L1 SINGLE;
DECLARE KEY SINGLE;
DECLARE PREFIX CHARACTER(1);

```

1 107
1 108
1 109
1 110
1 111
1 112
1 113
1 114
1 115
1 116
1 117
1 118
1 119
1 120
1 121
1 122
1 123
1 124
1 125
1 126
1 127
1 128
1 129
1 130
1 131
1 132
1 133
1 134
1 135
1 136
1 137
1 138
1 139
1 140
1 141
1 142
1 143
1 144
1 145
1 146
1 147
1 148
1 149
1 150
1 151
1 152
1 153
1 154
1 155
1 156
1 157
1 158
1 159
1 160
1 161
1 162
1 163
1 164

KSJMS PL/1 NEATER AND PRECOMPILER

5

PAGE

```

DECLARE SUFIX CHARACTER(2):
DECLARE TITL SINGLE:
DECLARE CWK SINGLE:
DECLARE ACSS SINGLE:
DECLARE BLWS SINGLE:
DECLARE CDPTR SINGLE:
DECLARE DPLNK SINGLE:
DECLARE ER SINGLE:
DECLARE LHSTR SINGLE:
DECLARE LNK SINGLE:
DECLARE LKNT SINGLE:
DECLARE LTOSP SINGLE:
DECLARE LVL SINGLE:
DECLARE MAT SINGLE:
DECLARE MDTYP SINGLE:
DECLARE PGM SINGLE:
DECLARE POS SINGLE:
DECLARE DTP SINGLE:
DECLARE SPLNK SINGLE:
DECLARE UPLNK SINGLE:
DECLARE YLDI SINGLE:
DECLARE YLND SINGLE:
DECLARE YLOTP SINGLE:
DECLARE OPS(140) CHARACTER(5):
3//** SYSTEM CALL NAMES */OPS(01)='IRETRN':
OPS(02)='JUMPI:
OPS(03)='JMSPI:
OPS(04)='RZ:
OPS(05)='WDF:
OPS(06)='SAND:
OPS(07)='RC:
OPS(08)='IGT:
OPS(09)='SHR1:
OPS(10)='SHR2:
OPS(11)='INX:
OPS(12)='LNT:
OPS(13)='SAND:
OPS(14)='SQ:
OPS(15)='SWAP:
OPS(16)='NEG:
OPS(17)='SHR:
OPS(18)='AND:
OPS(19)='WLT:
OPS(20)='OTVV:
OPS(21)='FX:
OPS(22)='CND:
OPS(23)='LDAL:
OPS(24)='GT30:
OPS(25)='OUT30:
OPS(26)='OPEN:
OPS(27)='CLOSE:
OPS(28)='CSP:
OPS(29)='STOP:
OPS(30)='STOP1:
OPS(31)='QUIT:
OPS(32)='CNSP:
OPS(33)='CNSPCH:

```

1 165
1 166
1 167
1 168
1 169
1 170
1 171
1 172
1 173
1 174
1 175
1 176
1 177
1 178
1 179
1 180
1 181
1 182
1 183
1 184
1 185
1 186
1 187
1 188
1 189
1 190
1 191
1 192
1 193
1 194
1 195
1 196
1 197
1 198
1 199
1 200
1 201
1 202
1 203
1 204
1 205
1 206
1 207
1 208
1 209
1 210
1 211
1 212
1 213
1 214
1 215
1 216
1 217
1 218
1 219
1 220
1 221
1 222

8

KSU'S PL/1 NFATENER AND PRECOMPILER

PAGE

```

      END;
      L1=L1+1;
      SYM(K,L1)=L;
      IF L=-1 THEN
        GO TO SY11;
      GO TO SY00;
      IF L1=5 THEN
        SYM(K,5)=0;
      END;
      J1=J1+1;
      SYM(LHS,J1)=1;
      /* ***** INITIALIZATION PART 5 - INPUT PRODUCTIONS TABLE */
      /* ***** GET SKIP LIST(I) ***** */
      /* # OF WS COMING IN */
      IF I>SIZE THEN
        BEGIN;
          /* TOO MANY, ABORT */
          GO=-12;
          /* PRODUCTIONS TABLE OVERFLOW */
          END;
      GET SKIP;
      DO J=1 BY 1 TO I;
        GET FOUT(PD1,PD2,PD3)(3 F(5,0));
        ROTL(J,1)=PD1;
        ROTL(J,2)=PD2;
        ROTL(J,3)=PD3;
      END;
      /* ***** PSIZE=1 ***** */
      /* KEOO ACTUAL LIMIT */
      /* ***** CONTROL SECTION ***** */
      CALL SCAN;
      /* SET RIGHT CONTEXT CHAR */
      ENTER;
      IF STK00 THEN
        GO TO FINIS;
      /* CONTEXT */
      IF STKPT=STKSZ THEN
        GO TO ENT3;
      /* STACKS FULL */
      STKPT=STKPT+1;
      PDST(STKPT)=PD;
      SWST(STKPT)=SM;
      CALL SCAN;
      /* PD-I AND RIGHT CONTEXT */
      MATCH;
      IF PD=1 THEN
        GO TO FINIS;
      PDPT=1;
      /* START IN JUMP KEY */
      MATCH;
      IF PDPT>PSIZE THEN
        GO;
        GO=-16;
      /* ILLEGAL REF. TAB REF. */
      CALL FOROP;
      END;
      PD1=ROTL(PDPT,1);
      PD2=ROTL(PDPT,2);
      PD3=ROTL(PDPT,3);

```

4 314
2 315
2 316
3 317
3 318
2 319
3 320
3 321
2 322
1 323
1 324
1 325
1 325
2 326
2 326
3 327
3 328
3 328
3 329
3 329
1 330
2 331
2 332
2 333
2 334
2 335
2 336
1 337
1 338
1 338
1 338
2 339
2 339
2 340
2 341
2 341
2 342
1 343
1 343
1 344
1 345
1 346
2 347
2 347
2 348
1 349
2 350
2 350
3 351
3 352
3 353
3 353
3 354
1 355
1 356
1 357

```

IF DCLCO THEN
  GO TO RC1X;
/* RIGHT CONTEXT CHECK */
DCL=REFD1/1024;
/* POSITION ON STACK */
MAT=REFD1-1024*DCL;
/* EQUIPPED SYMBOL */
IF MAT=0 THEN
  GO;
  KEY=REFD2;
/* ANY MATCH OK, BLOCK FAILURE */
GO TO DSPAT;
END;
KEY=REFD3;
/* ASSUME WILL FAIL */
IF STKTPDCL THEN
  GO TO MAT2;
/* STACK TOO SMALL */
K=PRSTK(STKPT-DCL+1);
IF K=MAT THEN
  KEY=REFD2;
/* SUCCESSFUL MATCH */
MAT2: IF KEY=0 THEN
  GO TO DSPAT;
/* SEMANTIC CALL */
PRST=KEY;
GO TO MATON;
END3: CO=-1;
/* PARSE STACK OVERFLOW */
CALL ERROR;
RC1X: MAT=-(DCL+1024);
KEY=REFD3;
/* ASSUME WILL FAIL */
IF MAT<500 THEN
  GO TO RC1;
IF MAT=0 THEN
  KEY=REFD2;
IF KEY=0 THEN
  GO TO DSPAT;
PRST=KEY;
GO TO MATON;
RC1: J=LHSTR;
J=0;
L=0;
J=J+1;
L=L+1;
K=SYM(I,J);
IF K=-1 THEN
  GO TO MAT2;
IF L=MAT THEN
  GO TO RC3;
IF J<4 THEN
  GO TO RC2;
IF SYM(I,5)=-1 THEN
  GO TO MAT2;
I=SYM(I,5);
J=0;
GO TO RC2;

```

KSU'S PL/I NEATEMR AND PRECOMPILER

10

PAGE

```

RC3:      IF K=0 THEN
           GO TO MAT2:
RC4:      J=0:
           J=J+1:
           L=S*(K,J):
           IF L=-1 THEN
               GO TO MAT2:
           IF L=0 THEN
               GO TO RC5:
           IF J<4 THEN
               GO TO RC4:
           J=0:
           K=SVW(K,5):
           GO TO RC4:
RC5:      KEY=RC02:
           /* SUCCESS */
           GO TO MAT2:
           QSDAT:  KEY=-KEY:
                   CHK=KEY:
                   IF KEY=0 THEN
                       GO TO ENTER:
                   IF KEY<500 THEN
                       DO:
                           IF KEY<250 THEN
                               DO:
                                   IF KEY<150 THEN
                                       DO:
                                           IF KEY=100 THEN
                                               GO TO S100:
                                           IF KEY=105 THEN
                                               GO TO S105:
                                           IF KEY=107 THEN
                                               GO TO S107:
                                           IF KEY=110 THEN
                                               GO TO S110:
                                           IF KEY=115 THEN
                                               GO TO S115:
                                           IF KEY=120 THEN
                                               GO TO S120:
                                           IF KEY=125 THEN
                                               GO TO S125:
                                           IF KEY=130 THEN
                                               GO TO S130:
                                           IF KEY=135 THEN
                                               GO TO S135:
                                           IF KEY=140 THEN
                                               GO TO S140:
                                           IF KEY=145 THEN
                                               GO TO S145:
                                           IF KEY=150 THEN
                                               GO TO S150:
                                           END:
                                           IF KEY=155 THEN
                                               GO TO S155:
                                           IF KEY=160 THEN
                                               GO TO S160:
                                           IF KEY=162 THEN
                                               GO TO S162:

```

2 406
2 407
1 408
1 409
1 410
2 411
2 412
2 413
2 414
2 415
2 416
1 417
1 418
1 419
1 420
1 421
1 422
1 423
2 424
2 425
2 426
2 427
4 428
5 429
6 430
7 431
8 432
8 433
8 434
8 435
8 436
8 437
8 438
8 439
8 440
8 441
8 442
8 443
8 444
8 445
8 446
8 447
8 448
8 449
8 450
8 451
8 452
8 453
8 454
8 455
7 456
6 457
6 458
6 459
6 460
6 461
6 462

11

KSIJ'S PI/I NEATNER AND PRECOMPILER

PAGE

```

IF KEY=165 THEN
GO TO S165;
IF KEY=170 THEN
GO TO S170;
IF KEY=175 THEN
GO TO S175;
IF KEY=180 THEN
GO TO S180;
IF KEY=185 THEN
GO TO S185;
IF KEY=190 THEN
GO TO S190;
IF KEY=193 THEN
GO TO S193;
IF KEY=196 THEN
GO TO S196;
IF KEY=197 THEN
GO TO S197;
IF KEY=198 THEN
GO TO S198;
IF KEY=199 THEN
GO TO S199;
IF KEY=200 THEN
GO TO S200;
IF KEY=205 THEN
GO TO S205;
IF KEY=210 THEN
GO TO S210;
IF KEY=215 THEN
GO TO S215;
IF KEY=230 THEN
GO TO S230;
IF KEY=235 THEN
GO TO S235;
IF KEY=240 THEN
GO TO S240;
IF KEY=245 THEN
GO TO S245;
END;
IF KEY=350 THEN
DO;
IF KEY=270 THEN
GO TO S270;
IF KEY=271 THEN
GO TO S271;
IF KEY=272 THEN
GO TO S272;
IF KEY=275 THEN
GO TO S275;
IF KEY=280 THEN
GO TO S280;
IF KEY=285 THEN
GO TO S285;
IF KEY=290 THEN
GO TO S290;
IF KEY=295 THEN
GO TO S295;
IF KEY=300 THEN

```

```

6 463
6 464
6 465
6 466
6 467
6 468
6 469
6 470
6 471
6 472
6 473
6 474
6 475
6 476
6 477
6 478
6 479
6 480
6 481
6 482
6 483
6 484
6 485
6 486
6 487
6 488
6 489
6 490
6 491
6 492
6 493
6 494
6 495
6 496
6 497
6 498
6 499
6 500
5 501
4 502
5 503
6 504
6 505
6 506
6 507
6 508
6 509
6 510
6 511
6 512
6 513
6 514
6 515
6 516
6 517
6 518
6 519
6 520

```

12

KSH'S PL/I NEATENER AND PRECOMPILER

	PAGE
GO TO S300;	6 521
IF KEY=301 THEN	6 522
GO TO S301;	6 523
IF KEY=305 THEN	6 524
GO TO S305;	6 525
IF KEY=310 THEN	6 526
GO TO S310;	6 527
IF KEY=315 THEN	6 528
GO TO S315;	6 529
IF KEY=320 THEN	6 530
GO TO S320;	6 531
IF KEY=325 THEN	6 532
GO TO S325;	6 533
IF KEY=330 THEN	6 534
GO TO S330;	6 535
IF KEY=335 THEN	6 536
GO TO S335;	6 537
IF KEY=340 THEN	6 538
GO TO S340;	6 539
IF KEY=345 THEN	6 540
GO TO S345;	6 541
IF KEY=350 THEN	6 542
GO TO S350;	6 543
END;	5 544
IF KEY=355 THEN	4 545
GO TO S355;	4 546
IF KEY=360 THEN	4 547
GO TO S360;	4 548
IF KEY=365 THEN	4 549
GO TO S365;	4 550
IF KEY=370 THEN	4 551
GO TO S370;	4 552
IF KEY=400 THEN	4 553
GO TO S400;	4 554
IF KEY=405 THEN	4 555
GO TO S405;	4 556
IF KEY=410 THEN	4 557
GO TO S410;	4 558
IF KEY=415 THEN	4 559
GO TO S415;	4 560
IF KEY=420 THEN	4 561
GO TO S420;	4 562
IF KEY=425 THEN	4 563
GO TO S425;	4 564
IF KEY=445 THEN	4 565
GO TO S445;	4 566
IF KEY=450 THEN	4 567
GO TO S450;	4 568
IF KEY=451 THEN	4 569
GO TO S451;	4 570
IF KEY=455 THEN	4 571
GO TO S455;	4 572
IF KEY=460 THEN	4 573
GO TO S460;	4 574
IF KEY=500 THEN	4 575
GO TO S500;	4 576
END;	3 577
IF KEY=675 THEN	2 578

KSII'S PL/I NEATER AND PRECOMPILER

13

PAGE

00:	IF KEY=501 THEN	3	570
	GO TO 5501:	4	580
	IF KEY=502 THEN	4	581
	GO TO 5502:	4	582
	IF KEY=503 THEN	4	583
	GO TO 5503:	4	584
	IF KEY=504 THEN	4	585
	GO TO 5504:	4	586
	IF KEY=505 THEN	4	587
	GO TO 5505:	4	588
	IF KEY=515 THEN	4	589
	GO TO 5515:	4	590
	IF KEY=520 THEN	4	591
	GO TO 5520:	4	592
	IF KEY=525 THEN	4	593
	GO TO 5525:	4	594
	IF KEY=530 THEN	4	595
	GO TO 5530:	4	596
	IF KEY=535 THEN	4	597
	GO TO 5535:	4	598
	IF KEY=540 THEN	4	599
	GO TO 5540:	4	600
	IF KEY=545 THEN	4	601
	GO TO 5545:	4	602
	IF KEY=550 THEN	4	603
	GO TO 5550:	4	604
	IF KEY=555 THEN	4	605
	GO TO 5555:	4	606
	IF KEY=560 THEN	4	607
	GO TO 5560:	4	608
	IF KEY=565 THEN	4	609
	GO TO 5565:	4	610
	IF KEY=580 THEN	4	611
	GO TO 5580:	4	612
	IF KEY=590 THEN	4	613
	GO TO 5590:	4	614
	IF KEY=620 THEN	4	615
	GO TO 5620:	4	616
	IF KEY=625 THEN	4	617
	GO TO 5625:	4	618
	IF KEY=630 THEN	4	619
	GO TO 5630:	4	620
	IF KEY=660 THEN	4	621
	GO TO 5660:	4	622
	IF KEY=665 THEN	4	623
	GO TO 5665:	4	624
	IF KEY=670 THEN	4	625
	GO TO 5670:	4	626
	IF KEY=675 THEN	4	627
	GO TO 5675:	3	628
	END:	2	629
	IF KEY=680 THEN	2	630
	GO TO 5680:	2	631
	IF KEY=685 THEN	2	632
	GO TO 5685:	2	633
	IF KEY=690 THEN	2	634
	GO TO 5690:	2	635
	IF KEY=700 THEN	2	636
	GO TO 5700:	2	

14

KSU'S PL/1 NEATER AND PRECOMPILER

PAGE

```

IF KSV=705 THEN
  GO TO S705: THEN
IF KSV=710 THEN
  GO TO S710: THEN
IF KSV=715 THEN
  GO TO S715: THEN
IF KSV=720 THEN
  GO TO S720: THEN
IF KSV=725 THEN
  GO TO S725: THEN
IF KSV=726 THEN
  GO TO S726: THEN
IF KSV=730 THEN
  GO TO S730: THEN
IF KSV=735 THEN
  GO TO S735: THEN
IF KSV=740 THEN
  GO TO S740: THEN
IF KSV=745 THEN
  GO TO S745: THEN
IF KSV=750 THEN
  GO TO S750: THEN
IF KSV=999 THEN
  GO:
/* BLOCK FAILURE */
  FATAL=FATAL+1:
  PD=102:
/* UNCOMPARABLE STATEMENT. CONTINUE */
  CALL ERROR:
  GO TO ENTER:
  END:
ER=-5:
CALL ERROR:
/* SEMANTIC ROUTINE NOT FOUND */
  STOP:
/* UNAP UP ROUTINE */
  CALL RUC4:
  OUT DATA:
  OUT PAGE:
  EXIT=1:
  I=SYM(TITL,2):
/* COND OFC TO MAIN TITL */
  NAME=CODE(1):
  OUT EDIT(I,TITL,NAMF)(COL(20),A,A):
  OUT EDIT(I,REFL)(COL(20),A):
  OUT EDIT(I,ENT START)(COL(20),A):
  OUT EDIT(I,EXTM 1)(COL(20),A):
  OUT EDIT(I,END 10)(COL(20),A):
  OUT EDIT(I,DATA: .BLK.)(COL(1),A):
  I=SYM(CCRH,4)+SYM(CCRH,5):
  OUT EDIT(I,F15,0):
  OUT EDIT(I;CONSTANT SECTION)(COL(1),A):
  I=COLNK:
/* LINK TO SP DEFINITIONS */
  C1: IF I=0 THEN
    GO TO F2:
/* NO MORE */

```

2 637
2 638
2 639
2 640
2 641
2 642
2 643
2 644
2 645
2 646
2 647
2 648
2 649
2 650
2 651
2 652
2 653
2 654
2 655
2 656
2 657
2 658
2 659
3 660
3 661
3 662
3 663
3 664
3 665
1 666
1 667
1 668
1 668
1 668
1 669
1 669
1 670
1 671
1 672
1 673
1 673
1 674
1 675
1 676
1 677
1 678
1 679
1 680
1 681
1 682
1 683
2 684
2 685
1 686

```

OPTYP=CODES(I,1);
/* CONSTANT LABEL VALUE */
CODES(I,1)=0;
/* CLEAR FOR NEXT PASS */
OPVAL=CODES(I,2);
NAME=CODE(OPVAL);
/* CODE POINTED */
I=CODES(I,3);
/* LINK TO NEXT */
OUT EDIT('C',OPTYP,'',NAME,SUFFIX(COL(1),A(1),F(4),A,A,A));
GO TO F1;
IF OPVAL=0 THEN
  GO TO F3;
/* IF NO S */
I=OPVAL;
OPVAL=0;
SUFFIX='0';
GO TO F1;

F2:
/* PUT OUT LITERALS */
OUT EDIT('LTRL',BLK 0')(COL(1),A);
IF LITCNT=0 THEN
  GO TO F4;
/* NO LITERALS */
OUT EDIT('LITERALS')(COL(1),A);
F4:
/* SYMTITLE,3 */
/* MAIN LABEL VALUE */
OUT EDIT('START',JMSR,'L',I)(COL(1),A,SKIP,COL(20),A,F(4));
IF EATN=0 THEN
  BEGIN;
/* OUTPUT READABLE NOVA CODE */
DO I=1 BY 1 TO CODEPT;
  OPTYP=CODES(I,1);
  OPVAL=CODES(I,2);
  OPVAL=CODES(I,3);
  IF OPTYP=0 THEN
    GO TO FIN3;
/* WAS CONSTANT LINK */
IF OPTYP=10 THEN
  BEGIN;
  PUT EDIT('DATA+',OPVAL)(COL(20),A,F(4));
  GO TO FIN1;
  END;
IF OPTYP=11 THEN
  BEGIN;
  PUT EDIT('C',OPVAL)(COL(20),A,F(4));
  GO TO FIN1;
  END;
IF OPTYP=12 THEN
  BEGIN;
  PUT EDIT('LTRL+',OPVAL)(COL(20),A,F(4));
  GO TO FIN1;
  END;
IF OPTYP=13 THEN
  BEGIN;
  PUT EDIT('L',OPVAL,'',0)(COL(1),A,F(4),A,COL(20),F(1));
  GO TO FIN1;
  END;

```

16

KKSJ'S PL/T NEATER AND PRECOMPILER

PAGE

[illegible]

17

KSIUS PL/I NEATER AND PRECOMPILER

PAGE

```

S1:      CODES(LINK,2)=CODES(LINK,2)+BLKDS:
        LINK=CODES(LINK,3):
        IF LINK=0 THEN
            GO TO S1:
        RETURN:
    END:
/* ***** */
CALLC:  PROCEDURE:
/*  EPTD ONTO FREE STACK */
        SYM(EPTD,1)=0:
        SYM(EPTD,2)=0:
        SYM(EPTD,3)=0:
        SYM(EPTD,4)=0:
        SYM(EPTD,5)=SYMFL:
        SYMFL=EPTD:
        RETURN:
    END:
/* ***** */
KEEP:  PROCEDURE:
        DECLARE I RN FIXED(15):
        DECLARE J RN FIXED(15):
        IF SYMFL=0 THEN
            GO TO K3:
        /* SYM OVERFLOW */
        PTR=SYMFL:
        /* GET FREE ELEMENT */
        SYMFL=SYM(SYMFL,5):
        ON CODE=1 BY 1 TO CSIZE:
        /* GET FREE CODE */
        IF CODE(CDPTD)=1 THEN
            GO TO K1:
        END:
        GO TO K2:
        /* CODE OVERFLOW */
        K1:  CODE(CDPTD)=NAME:
        SYM(EPTD,2)=CDPTD:
        J=SYM(CVL,2):
        I=SYM(J,5):
        /* LINK TO BLOCK DESCRIPTORS */
        SYM(J,5)=PTR:
        /* INSERT AT TOP OF BLOCK */
        SYM(EPTD,5)=I:
        /* RE-LINK OLD ELEMENTS */
        RETURN:
        EQ=-9:
        /* CODE OVERFLOW */
        CALL EQ=0:
        K3:  EQ=-10:
        /* SYMBOL TABLE OVERFLOW */
        CALL EQ=0:
        RETURN:
    END:
/* ***** */
ALKEK:  PROCEDURE:
        CALL RUG1:
        /* BUILD SYMBOL TABLE */
        /* BLOCK EXIT AND WADHIP ROUTINE */

```



```

R12:      CO=12;
/* LABEL UNDEFINED */
CALL ERROR;
GO TO R1;

FINIS:
  OCW=1;
  RETURN;
  END;

/* ***** */
ASPT:      PROCEDURE;
/* ALLOCATE SP. TEMPORARY */
/* RETURN ITS ADDRESS AT TMPTR */
  IF TSPTD=0 THEN
    GO TO A1;
/* NO SP'S LISTED */
  TMPTR=TSK(TSPTD);
/* TOP SP LISTED */
  TSPTD=TSPTD-1;
/* POP OFF */
  RETURN;
  END;

A1:      IF TOPD=TSIZE/* CAN SPLIT DP? */THEN
  GO TO A2;
/* NO, GET NEW CORE */
  TMPTR=TSK(TOPD);
/* ASSIGN FIRST WORD */
  TSPTD=TSPTD+1;
/* SAVE SECOND ON SP'S */
  TSK(TSPTD)=TSK(TOPD)+1;
  TOPD=TOPD+1;
  RETURN;
  END;

A2:      ACT#1;
/* NEED ONE WORD */
/* ALSO RETURN'S TMPTR */
  RETURN;
  END;

/* ***** */
ASPT:      PROCEDURE;
/* ALLOCATE A DP. TEMPORARY */
/* RETURN ITS ADDR AT TMPTR */
/* TSIZE IS STACK SIZE */
  IF TOPD=TSIZE THEN
    GO TO A1;
/* GET CORE */
  TMPTR=TSK(TOPD);
  TOPD=TOPD+1;
  RETURN;
  END;

A1:      ACT#2;
/* NEED TWO WORDS */
  CALL CORE;
/* ALSO RETURN'S TMPTR */
  RETURN;
  END;

/* ***** */
ASPT:      PROCEDURE;
/* DE-ALLOCATE SP. ENTRY */
  IF TSPTD+1=TOPD THEN
    GO TO DD;
  TSPTD=TSPTD+1;
  DD:

```

2 917
2 918
2 918
2 919
2 920
2 921
2 922
2 923
2 923
3 924
3 924
3 924
3 925
3 926
2 926
2 927
2 927
2 928
2 928
3 929
3 930
2 931
2 931
2 932
2 932
2 933
2 933
2 934
2 935
2 936
2 937
2 937
2 938
2 938
2 939
2 940
2 940
3 941
3 941
3 941
3 941
3 942
2 943
2 943
2 944
2 945
2 946
2 947
2 947
2 948
2 948
2 949
2 949
2 950
2 950
3 951
3 951
3 952
2 953


```

/* YES */ RETURN;
/* LITCAL ? */
SF4: 1=ROOL(UNSPEC(SF4),'00010000000000000000'B,'0001'B);
      IF I=0 THEN
        GO TO SF5;
/* NO */ SETPL=-2;
/* YES */ RETURN;
/* SUBSCRIPTED VARIABLE ? */
SF5: 1=ROOL(UNSPEC(SF5),'00000000000100000'B,'0001'B);
      IF I=0 THEN
        GO TO SF6;
/* NO */ SETPL=-1;
/* YES */ RETURN;
/* ASSIGNED STORAGE TEMPORARY ? */
SF6: 1=ROOL(UNSPEC(SF6),'0000000001000000'B,'0001'B);
      IF I=0 THEN
        GO TO SF10;
/* NO */ SETPL=CCRH;
/* YES, MUST BE LOCATED IN CCRH */
      RETURN;
SF10: 1=ESTOH;
      J=SYM(I,2);
/* DODE PTR */
SF7:  J=SYM(J,5);
/* LINK */ IF J=SETW THEN
      GO TO SF9;
/* FOUND IT */ IF J=0 THEN
      GO TO SF7;
      1=SYM(I,3);
      IF I=0 THEN
        GO TO SF8;
      SETPL=SETC;
/* DID NOT FIND IT, ASSUME PASSED */
      RETURN;
      SETPL=1;
      RETURN;
      FOR=-2;
/* PAGE STACK REFERENCE ILLEGAL */
      CALL ERROR;
      RETURN;
SF2:  FOR=-3;
/* PAGE STACK REFERENCE VACANT */
      CALL ERROR;
      RETURN;
      FIN.
/* ***** */
REVER:  RECOMPILER;
/* CREATE ONE CODED LINE OF OUTPUT */
GO:  IF OPTYP=10 THEN
      BEGIN;
/* DATA - DATA BASE REFERENCE, LINK TO HEADER. */
/* DATA MUST HAVE RLOCK CHAIN */
      IF OPTYP=0 THEN
        BEGIN;
          IF SETAL=-3 THEN
            OPTYP=11;
          IF SETAL=-2 THEN
            OPTYP=12;
        END;
      END;
/* CXXX */

```

2 1041
2 1042
2 1042
3 1043
3 1044
2 1045
2 1046
2 1047
2 1047
3 1048
3 1049
2 1050
2 1051
2 1052
2 1052
3 1053
3 1054
2 1055
2 1056
2 1056
2 1057
2 1058
2 1059
2 1059
3 1060
3 1061
3 1062
3 1063
2 1064
3 1065
3 1066
2 1067
2 1068
2 1068
2 1069
2 1070
2 1071
2 1072
2 1072
2 1073
2 1074
2 1075
2 1075
2 1076
2 1077
2 1077
2 1078
2 1078
3 1079
3 1079
4 1080
5 1081
5 1081
5 1081
6 1082
7 1083
7 1084
7 1085
7 1086


```

3 1173 IF J=0 THEN
3 1174 GO TO R1:
2 1175 /* NOT TEMP */
2 1176 /* IS CP, DP, OR CHAR TEMPORARY */
2 1177 TWOPO=SYM(SET#,3):
2 1178 /* HCOL ADDRESS */
2 1179 J=COLL(UNSPEC(1),'00000001000000000000'B,'0001'R):
3 1177 IF J=0 THEN
3 1178 CALL DASPT:
3 1179 ELSE
3 1179 /* DP */
4 1179 BEGIN:
4 1180 /* CHARACTER ? */J=COLL(UNSPEC(1),'0010000000000000'B,'0001'R):
5 1181 IF J=0 THEN
5 1182 CALL DASPT:
5 1183 ELSE
5 1183 /* YES */
5 1184 /* N, SP */
5 1185 /* N, SP */
5 1186 /* N, SP */
5 1187 /* N, SP */
5 1188 /* N, SP */
5 1189 /* N, SP */
5 1190 /* N, SP */
5 1191 /* N, SP */
5 1192 /* N, SP */
5 1193 /* N, SP */
5 1194 /* N, SP */
5 1195 /* N, SP */
5 1196 /* N, SP */
5 1197 /* N, SP */
5 1198 /* N, SP */
5 1199 /* N, SP */
5 1200 /* N, SP */
5 1201 /* N, SP */
5 1202 /* N, SP */
5 1203 /* N, SP */
5 1204 /* N, SP */
5 1205 /* N, SP */
5 1206 /* N, SP */
5 1207 /* N, SP */
5 1208 /* N, SP */
5 1209 /* N, SP */
5 1210 /* N, SP */
5 1211 /* N, SP */
5 1212 /* N, SP */
5 1213 /* N, SP */
5 1214 /* N, SP */

```


KSU'S PL/I NEATER AND PRECOMPILER

28

PAGE

```

/* ***** */
DATA1: PROCEDURE;
/* DATA OR LABEL REFERENCE */
CALL SETUP;
OPTYPE=10;
OPVAL=SETC;
/* ADDRESS DISPLACEMENT */
CALL GENPR;
RETURN;
END;

/* ***** */
DATA2: PROCEDURE;
OPTYPE=10;
/* TEMP. DATA REFERENCE */
OPVAL=TWDP;
OPLINK=CONR;
CALL GENPR;
RETURN;
END;

/* ***** */
CHOP: PROCEDURE;
/* CHAR SOURCE INPUT ROUTINE */
ON ENDFILE(SYSIN)
  BEGIN;
  EOF=15;
  /* EOF REFERENCE LOGICAL END OF PROGRAM */
  CALL ERROR;
  END;

/* ENI = BLANK IGNORE */
/* SW6 = COMMENT SWITCH */
/* SW5 = LITERAL SWITCH */
/* SW6 = 0 = SWITCH */
C1: GET EDIT1(A(1));
PUT EDIT1->,CH1,C-,(A,A,A);
IF SW5=0 THEN
  RETURN;
IF CH1=1 THEN
  DO;
    EOF=1;
    RETURN;
  END;
IF CH1=7 THEN
  GO TO C2;
IF SW1=0 THEN
  RETURN;
IF CH1=8 THEN
  GO TO C1;
RETURN;
C2: GET EDIT1(A(1));
IF CH1=8 THEN
  GO TO C1;
GO TO C2;
END;

/* ***** */
LAPR: PROCEDURE;
/* ASSIGN LBL AS UNIQUE VALUE OF LABEL REFERENCES
   THREADED AT SYM(LINK,4) */
TSYM(LINK,4);

```

2 1258
2 1258
2 1259
2 1259
2 1260
2 1261
2 1262
2 1262
2 1263
2 1264
2 1265
2 1265
2 1266
2 1267
2 1267
2 1268
2 1269
2 1270
2 1271
2 1272
2 1272
2 1273
2 1273
3 1274
3 1275
3 1276
3 1276
3 1277
3 1278
2 1278
2 1278
2 1278
2 1278
2 1279
3 1280
3 1281
3 1282
4 1283
4 1284
4 1285
4 1286
3 1287
3 1288
3 1289
3 1290
3 1291
3 1292
2 1293
2 1294
3 1295
3 1296
2 1297
2 1298
2 1299
2 1299
2 1300
2 1300
2 1300

29

KSIUS PL/T NEATENER AND PRECOMPILER

PAGE

```

181:      IF I=0 THEN
        GO TO LR2:
      CODES(I,2)=LRL:
      I=CODES(I,3):
      GO TO LR1:
    RETURN:
  END:
/* ***** */
THRD2:  PROCEDURE:
/* CONNECT THEAD AT SYM(LINK,4) TO END OF THREAD
   AT SYM(SNO,4): */
  I=SYM(SNO,4):
  I=CODES(I,3):
  IF CODES(I,3)=0 THEN
    GO TO TH2:
  GO TO TH1:
  CODES(I,3)=SYM(LINK,4):
  RETURN:
END:
/* ***** */
AL17:  PROCEDURE:
/* ALLOCATE A SYM ELEMENT AT PTR */
PTR=SYMFL:
IF PTR=0 THEN
  GO TO A1:
  SYMFL=SYM(PTR,5):
  SYM(PTR,5)=0:
  RETURN:
  ER=-10:
/* TABLE OVERFLOW */
CALL ERROD:
RETURN:
END:
/* ***** */
reod:  PROCEDURE:
DECLARE I IN FIXED(15):
IF ER<0 THEN
  GO TO E3:
/* TERMINAL */
IF ER=0 THEN
  GO TO E2:
/* dump */
IF ER>99 THEN
  GO TO E1:
/* NON FATAL */
FATAL=FATAL+1:
/* FATAL, INC COUNTER */
ERR=ERR+1:
IF ERR>FSIZE THEN
  GO TO E3:
ERR(ERRPTR)=ER:
/* WARNING */
ERR2=ERR+1:
/* IF ERR2>FSIZE THEN
  GO TO E3:
ERR2(ERRPTR)=ER:
  RETURN:
  */
/* dump */
DO I=1 BY 1 TO SPTR:

```

3 1301
3 1302
2 1303
2 1304
2 1305
2 1306
2 1307
2 1308
2 1309
2 1309
2 1309
2 1310
3 1311
2 1312
2 1313
2 1314
2 1315
2 1316
2 1317
2 1317
2 1318
2 1318
3 1319
3 1320
2 1321
2 1322
2 1323
2 1324
2 1325
2 1325
2 1326
2 1327
2 1328
2 1328
2 1329
3 1330
3 1331
3 1332
3 1333
3 1334
3 1335
2 1336
2 1336
2 1337
2 1337
3 1338
3 1339
2 1340
2 1341
2 1342
3 1343
3 1344
2 1345
2 1346
3 1347
3 1347

32

KSII'S PL/I NEATFENR AND PRECOMPILER

```

/* NJ, TERMINAL ERROR */
      EQ=8;
/* COMPRESSION TO CHAR ERROR */
      CALL ERROR;
    END;
    NOTVP=112;
/* CCMCH */
    CALL GENPR;
    SET#VL01;
    CALL DATA1;
    CALL ACUT;
/* ALLOCATE A CH STR TEMP */
    CALL DATA2;
/* CM REFERENCE */
    I=TEMP1;
/* HOLD ITS ADDRESS */
/* WAS NO TEMPORARY ? */
    K=COLL(UNSPEC(J),'0000000001000000'B,'0001'R);
    IF K=0 THEN
      BEGIN;
/* YES, RELEASE IT */
      TMRP=SYM(I,3);
/* TEMP (N) ADDRESS */
      CALL DAPRT;
      END;
    SYM(I,3)=L;
/* NEW ADDRESS */
    SYM(I,1)=+010000001000000R;
/* TEMP CHAR */
    RETURN;
  END;
/* ***** */
VLDOP:  DANCEUPE;
        DECLARE Y BIN FIXED(15);
        DECLARE J BIN FIXED(15);
        DECLARE K BIN FIXED(15);
        DECLARE I BIN FIXED(15);
/* YIELD A NP TEMPORARY OF VALUE REFER. THROUGH VL01 */
/* COMPRESSIONS VALID FROM SP OR CHAR */
        I=SMSTK(STROT-VLDI)+1;
        IF I=0 THEN
          BEGIN;
            REFIN;
            CR=6;
/* ERROR */
/* COMPRESSION SOURCE NOT FOUND */
            CALL ERROR;
          END;
          J=SYM(I,1);
/* BIT MCH */
/* ALREADY OK ? */
          K=COLL(UNSPEC(J),'000000010000000000'B,'0001'R);
          IF K=0 THEN
            BEGIN;
              REFIN;
              RETURN;
            END;
/* YES */
/* NO NOTHING */
/* IS IT CHARACTER ? */
          K=COLL(UNSPEC(J),'001100000000000000'B,'0001'R);
          IF K=0 THEN

```

```

/* YES */
/* CCCHPOD */
/* CHAP SOURCE REFERENCE */
/* DP RESULT REFERENCE */
/* WAS CHARACTER STORAGE TEMPORARY ? */
/* YES */
/* OLD TEMP ADDRESS FREE */
/* FREE 6 WORD */
/* NEW TEMP ADDRESS */
/* IS IT INTEGER ? */
/* YES, MUST BE SP */
/* CCCHPOD */
/* SP REFERENCE OUTPUT */
/* DP RESULT REFERENCE */
/* WAS SP TEMPORARY ? */
/* YES */
/* OLD SP TEMP ADDRESS */
/* FREE IT */
/* NEW ADDRESS */
/* BLOCK POINTER */
/* TEMP OP INT */
/* ELSE CONVERSION IMPOSSIBLE */
/* EQ=11:
   CALL ERROR;
   CONVERSION TYPES INVALID
END:

```

```

4 1471
4 1472
4 1473
4 1474
4 1475
4 1476
4 1477
4 1478
4 1479
4 1479
5 1480
5 1480
6 1481
6 1482
6 1483
6 1484
4 1485
4 1486
4 1487
4 1488
4 1489
2 1490
2 1490
3 1491
4 1492
4 1493
4 1493
4 1494
4 1495
4 1496
4 1497
4 1497
4 1498
4 1499
4 1500
4 1500
5 1501
5 1501
6 1502
6 1503
6 1504
6 1504
6 1505
4 1506
4 1507
4 1508
4 1508
4 1509
4 1509
4 1510
2 1511
2 1511
2 1512
2 1513
2 1513

```

```

/* ***** */
VFLD:
DECLARE I1 BIN FIXED(15);
DECLARE I2 BIN FIXED(15);
DECLARE J  BIN FIXED(15);
DECLARE K  BIN FIXED(15);
DECLARE L  BIN FIXED(15);
IF VLD2=0 THEN
  GO TO V1;
  I1=SMSTK(STKPT-VLD1+1);
  I2=SMSTK(STKPT-VLD2+1);
  K=ROQL(UNSPEC(YLDT),0010000000000000'R,0001'R);
  /* CHARACTER WANTED ? */
  IF K=0 THEN
    BEGIN;
    K=ROQL(UNSPEC(SYM(I1,1)),0011000000000000'R,0001'R);
    /* YES */
    /* DO NOT PROCESS IF CHARACTER */
    IF K=0 THEN
      CALL YLDOP;
    /* VLD1 -> NO */
    /* -> CHARACTER */
    YLD1=YLD2;
    K=ROQL(UNSPEC(SYM(I2,1)),0011000000000000'R,0001'R);
    IF K=0 THEN
      CALL YLDOP;
    CALL YLDOP;
    YLDT=ROQL(UNSPEC(SYM(I1,1)),UNSPEC(SYM(I2,1)),0001'R);
    /*
    RETURN;
    */
  END;
  K=ROQL(UNSPEC(YLDT),0100000000000000'R,0001'R);
  IF K=0 THEN
    BEGIN;
    /* YES */
    /* ASSUME THAT IF DP SPECIFIED THEN PREFIX NOT WANTED */
    /* NO WANTED */
    K=ROQL(UNSPEC(YLDT),0000001000000000'R,0001'R);
    IF K=0 THEN
      BEGIN;
      /* YES */
      /* VLD1 -> NO */
      CALL YLDOP;
      YLD1=YLD2;
      CALL YLDOP;
      YLDT=ROQL(UNSPEC(SYM(I1,1)),UNSPEC(SYM(I2,1)),0001'R);
      /* NO NOT WANT APPLY, SUB VAR, IN YLDT */
      RETURN;
      END;
    /* NO NOT WANT */
    /* COMBINE ALL ATTRIBUTES */
    L=ROQL(UNSPEC(SYM(I1,1)),UNSPEC(SYM(I2,1)),0110'R);
    /* EITHER DP OR CHARACTER */
    L=ROQL(UNSPEC(L),0010010000000000'R,0001'R);
    IF L=0 THEN
      BEGIN;
      /* YES, DP RESULTS */
      /* VLD1 -> NO */
      CALL YLDOP;
      YLD1=YLD2;
      CALL YLDOP;
      PREFIX=0;
      /* PREFIX CHAR */
      END;

```

```

2 1514
2 1514
2 1515
2 1516
2 1517
2 1518
2 1519
3 1520
3 1521
2 1522
2 1523
2 1524
3 1525
3 1525
3 1526
4 1527
4 1527
5 1528
5 1528
5 1529
4 1530
4 1531
4 1531
4 1532
5 1533
5 1534
4 1535
4 1536
4 1536
4 1537
4 1538
2 1539
3 1540
4 1541
4 1542
4 1542
4 1542
5 1543
6 1544
6 1545
6 1546
6 1547
6 1548
6 1548
6 1549
6 1549
6 1550
4 1551
4 1551
4 1552
4 1552
5 1553
6 1554
6 1555
6 1556
6 1557
6 1558
6 1559

```

```

      YLPTP=BNCL(UNSPEC(SYM(11,1)),UNSPEC(SYM(12,1)),0110'B
    ):
      RETURN;
    END;

  FR=-11;
  /* CONVERSION TYPES INVALID */
  CALL EPNR2;
  RETURN;
  /* SINGLE OPERAND */
  END;

  /* ***** */
  BUG1: DO COOPE;
  /* DUMP OUT SYMCON TABLE */
    DCL (I,J,K) PRIMARY FIXED (15,0);
    DCL RITS(16) CHAP(15);
    RITS(1)=TABLE ENTRY;
    RITS(2)=INTEGER;
    RITS(3)=CHARACTER;
    RITS(4)=LITERAL;
    RITS(5)=LABEL;
    RITS(6)=CONSTANT;
    RITS(7)=DOUBLE PREC;
    RITS(8)=ARRAY;
    RITS(9)=KEYWORD;
    RITS(10)=TEMPORARY;
    RITS(11)=SUBSCRIPTED VAR;
    RITS(12)=?;
    RITS(13)=?;
    RITS(14)=?;
    RITS(15)=?;
    RITS(16)=?;
    CALL BUG2;
  /* DUMP COOPE ALSO */
  I=FSRSH;
  IF I=0 THEN
    GO TO D4;
  PUT EDIT('BLOCK HEADER')(SKIP(5),A);
  PUT EDIT('A',I,->,(SYM(I,J)DO J=1 TO 5))(SKIP,A,F(3),A,5 F
    (6,0));
  J=SYM(1,2);
  PUT EDIT('COOPE HEADER')(SKIP,A);
  PUT EDIT('A',J,->,(SYM(J,K)DO K=1 TO 5))(SKIP,A,F(3),A,5 F
    (6,0));
  PUT SKIP;
  J=SYM(J,5);
  IF J=0 THEN
    GO TO D3;
  IF SYM(J,1)=-32640 THEN
    DO;
    PUT SKIP EDIT('K W ',COOPE(SYM(J,2)))(A,A);
    PUT EDIT(J)(X(3),F(5));
    J=SYM(J,5);
    GO TO D2;
  END;
  PUT EDIT('...A',J,->,(SYM(J,K)DO K=1 TO 5))(SKIP,A,F(3),A,
    5 F(6,0));
  PUT SKIP EDIT(UNSPEC(SYM(J,1)))(R(16));
  DO K=1 TO 16;

```

```

      IF SURSTP(UNSPEC(SYM(J,1)),K,1)='0' THEN
        PUT EDIT(RITS(K))(SKIP,COL(10),A);
      END;
      PUT EDIT('.....')(COL(10),A);
      PUT SKIP;
      J=SYM(J,5);
      GO TO D2;
D3:  I=SYM(I,3);
      GO TO D1;
D4:  PUT EDIT('END OF SYMBOL TABLE')(SKIP(3),A);
      RETURN;
END;
/* ***** */
BUG2: PROCEDURE;
/* DUMP CODEP */
      OCL I RN FIXED(15,0);
      PUT EDIT('CODEP ARRAY')(SKIP(3),A);
      DO I=1 TO CSIZE;
        PUT SKIP;
        PUT EDIT('*,I,*,CODEP(I))(X(5),A,F(3,0),A,A);
      END;
      PUT SKIP LIST('CODEP DUMPED');
      END BUG2;
/* ***** */
BUG3: PROCEDURE;
/* DUMP FREE LIST */
      OCL I RN FIXED(15,0);
      PUT EDIT('FREE LIST DUMP')(SKIP(3),A);
      PUT EDIT('SYML = ',SYML)(SKIP,A,F(4,0));
      I=SYML;
      IF I=0 THEN
        GO TO R2;
      PUT EDIT(I)(F(4,0));
      I=SYM(I,5);
      GO TO R1;
R2:  PUT EDIT('END BUG3')(SKIP,A);
      RETURN;
      END BUG3;
/* ***** */
BUG4: PROCEDURE;
/* DUMP WHOLE CODES ARRAY */
      OCL I RN FIXED(15,0);
      PUT EDIT('CODEP = ',CODEP)(SKIP(3),A,F(5));
      PUT EDIT('OPTYP',OPTYP',OPVAL',OPLNK')(SKIP,COL(10),A,COL(20),A,C
      OL(30),A);
      DO I=1 TO CODEP;
        PUT EDIT(I,CODES(I,1),CODES(I,2),CODES(I,3))(SKIP,F(3),CO
        L(10),F(5),COL(20),F(15),COL(30),F(5));
      END;
      PUT EDIT('THATS ALL SHF WRITE')(SKIP,A);
      END BUG4;
/* ***** */
BUG5: PROCEDURE;
/* DUMP 25 ROWS OF CODES ARRAY */
      OCL (I,START) BIN FIXED(15,0);
      PUT EDIT('CODEP = ',CODEP)(SKIP(3),A,F(4));
      PUT EDIT('OPTYP',OPTYP',OPVAL',OPLNK')(SKIP,COL(10),A,COL(20),A,C
      OL(30),A);

```

```

4 1609
4 1610
3 1611
2 1612
2 1613
2 1614
2 1615
2 1616
2 1617
2 1618
2 1619
2 1620
2 1621
2 1622
2 1622
2 1622
2 1623
3 1624
3 1625
3 1626
3 1627
2 1628
2 1629
2 1630
2 1630
2 1631
2 1631
2 1632
2 1633
2 1634
3 1635
3 1636
2 1637
2 1638
2 1639
2 1640
2 1641
2 1642
2 1643
2 1643
2 1644
2 1644
2 1645
2 1646
2 1646
3 1647
3 1648
3 1648
3 1649
2 1650
2 1651
2 1652
2 1652
2 1653
2 1653
2 1654
2 1655
2 1655
2 1655

```

37

KSJHS PL/I NFATENER AND PRECOMPILER

PAGE

```

START=CODEPT-24;
IF START<1 THEN
  START=1;
DO I=START TO CODEPT;
  PUT SKIP;
  PUT EDIT(CODES(I,1),CODES(I,2),CODES(I,3))(COL(10),F(5),C
  OL(20),F(5),COL(30),F(5));
END;
PUT EDIT('THATS THE LAST BATCH')(SKIP,A);
END;
/* ***** */
BUG6: PROCEDURE;
/* DUMP TEMP STACK */
DO I=1 TO TSIZE;
  PUT SKIP;
  PUT EDIT('I=',TSTK(I))(X(5),A,F(3),A,F(3));
END;
END BUG6;
/* ***** */
BUG7: PROCEDURE;
/* DUMP PARSE STACKS */
DO I=1 TO FIXED(15);
  PUT EDIT('STKPT = ',STKPT)(SKIP(3),A,F(3));
  PUT EDIT('STKSZ = ',STKSZ)(SKIP,A,F(3));
  PUT EDIT('PRSTK','SMSTK')(SKIP,COL(10),A,COL(20),A);
  DO I=1 TO STKPT;
    PUT EDIT('PRSTK(I),SMSTK(I))(SKIP,F(5),COL(10),F(5),COL(
    20),F(5));
  END;
END;
PUT EDIT('END OF STACKS')(SKIP,A);
END BUG7;
/* ***** */
BUG8: PROCEDURE;
/* WISC VARIABLES */
PUT SKIP LIST('*****');
PUT SKIP(2)DATA(SETA,SETB,SETC,SETD,SETE,SETF);
PUT SKIP(2)DATA(NAME,PR,SM);
PUT SKIP(2)DATA(UNIQUE,STKPT,REDPT,RED1,RED2,RED3);
PUT SKIP(2)DATA(CCBH);
PUT SKIP LIST('*****');
END BUG8;
/* ***** */
SCAN6: PROCEDURE;
DECLARE INLEN BIN FIXED(15);
IF EOF=1 THEN
  NO;
  STKO=-1;
  SM=0;
  PR=0;
  RETURNS;
END;
/* RETURNS: SM (SEM PTR) PR (SYNTAX VALUE) */

```



```

SC05:      NAME=      *;
           SW1=1;
/* BLANK IGNORE SWITCH ON */
           SW5=0;
/* LITERAL SWITCH */
SC010:      IF SW2=1 THEN
           GO TO SC030;
/* LOOK AHEAD ON */
SC020:      SW4=0;
/* COMMENT SWITCH */
           CALL CHAP;
/* NEXT INPUT CHAR INTO CH1 */
SC030:      SW2=0;
/* LOOK AHEAD OFF. */
           IF SW5=0 THEN
           IF CH1='*' THEN
           GO TO SC080;
/* RE-ENTER COMMENT FLUSH */
           INLEN=1;
           SW1=0;
/* BLANK IGNORE OFF */
           IF CH1='.' THEN
           GO TO SC140;
/* LITERAL */
           IF CH1='0' THEN
           IF CH1='9' THEN
           GO TO SC199;
/*CNST*/      IF CH1='A' THEN
           IF CH1='Z' THEN
           GO TO SC130;
/* ASSUME VAR*/
SC040:
/* SPECIAL CHARACTER. */
           SUBSTR(NAME,1,1)=CH1;
           IF CH1='*' THEN
           GO TO SC050;
/* SEE IF ** */
           IF CH1='/' THEN
           GO TO SC070;
/* SEE IF COMMENT START */
           IF CH1='<' THEN
           GO TO SC110;
/* SEE IF <= */
           IF CH1='>' THEN
           GO TO SC110;
/* SEE IF >= */
           IF CH1='-' THEN
           GO TO SC110;
/* SEE IF -= */
           GO TO SC135;
/* SINGLE SPECIAL CHARACTER */
SC050:      CALL CHAP;
           IF CH1='*' THEN
           GO TO SC060;
SC055:      SW2=1;
/* LOOK AHEAD SWITCH ON */
           GO TO SC135;
/* CHECK TERMINAL TABLE */

```

```

2 1704
2 1705
2 1706
2 1706
3 1707
3 1708
2 1709
2 1709
2 1710
2 1710
2 1711
2 1711
3 1712
3 1712
4 1713
4 1714
2 1715
2 1715
2 1716
3 1717
3 1717
3 1718
3 1719
3 1719
4 1720
4 1721
3 1722
4 1723
4 1724
2 1725
2 1725
2 1725
2 1725
3 1726
3 1727
3 1728
3 1728
3 1729
3 1729
3 1730
3 1730
3 1731
3 1732
3 1732
3 1733
3 1734
3 1734
3 1735
2 1736
2 1736
2 1737
2 1737
3 1738
3 1739
2 1740
2 1741
2 1741
2 1742

```

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE 39

```

SC060:  SUBSTR(NAME,2,1)=CH1;
/* PUT IN SECOND CHAR. */
INLEN=2;
GO TO SC135;
CALL CHAR;
IF CH1='*' THEN
GO TO SC055;
/* LOOK AHEAD ON. CHECK IT */
SC090:
/* START OF A COMMENT. FLUSH, THEN REENTER SCAN. */
SW4=1;
/* COMMENT SWITCH */
CALL CHAR;
/* BLANK. IGNORE STILL OFF */
IF CH1=' ' THEN
GO TO SC05;
IF CH1='*' THEN
GO TO SC090;
/* LOOK FOR AST-SLASH */
CALL CHAR;
IF CH1='/' THEN
GO TO SC05;
IF CH1='*' THEN
GO TO SC090;
SW1=1;
/* BLANK. IGNORE ON */
GO TO SC020;
SC110:  CALL CHAR;
/* CHECK FOR INCOMING = < > = ~ = */
IF CH1='=' THEN
GO TO SC055;
/* NOT THERE. */
SC120:  SUBSTR(NAME,2,1)=CH1;
INLEN=2;
GO TO SC135;
/* CHECK TABLE */
SC130:
/* ENCOUNTERED A VARIABLE NAME.(POSSIBLY A KEYWORD) */
SUBSTR(NAME,INLEN,1)=CH1;
SW1=0;
/* BLANK. IGNORE OFF */
CALL CHAR;
/* BLANK. IGNORE SWITCH STILL OFF */
IF CH1=' ' THEN
GO TO SC135;
SW2=1;
/* LOOK AHEAD ON */
/* CHECK FOR ALPHA */
IF CH1<'A' THEN
GO TO SC138;
IF CH1>'Z' THEN
GO TO SC138;
GO TO SC139;
/* TOKEN END */
/* CHECK FOR NUMERIC */
IF CH1<'0' THEN
GO TO SC135;
IF CH1>'9' THEN

```

```

      GO TO SC135;
SC139:  SW2=0;
/* LOOK AHEAD OFF */
      IF INLEN=9 THEN
        DO;
/* NAME TOO LONG */
          ER=101;
          CALL FRROR;
          GO TO SC130;
        END;
        INLEN=INLEN+1;
/* SHOULD CHECK LENGTH. */
        GO TO SC130;
SC135:  CLVL=CCBH;
/* START SEARCH LOCALLY */
        CALL VFIND;
/* LOOK FOR NAME */
        IF SMO=0 THEN
          GO TO SC136;
/* YES - PRESENT */
        CLVL=CCBH;
/* KEEP WITHIN CURRENT BLOCK */
        CALL KEEP;
/* MISSING, PUT IT IN. */
/* RETURNS SMO POINTER INTO TABLE */
        SM=PTR;
        IF TITL=0 THEN
          TITL=SM;
/* MAIN */
        SYM(SM,1)=BINARY('1000000000000000'B);
/* SYNTAX DESCRIPTOR - VARIABLE CLASS */
/* ? */
        PUT EDIT('ENTERED IN SYM. ',NAME,' CLASS 1 AT ',SM)(SKIP,A,A
        ,A,F(3));
/* ? */
        RETURN;
/* NEW TERMINAL ENTERED AND NOTED. */
SC136:  I=SYM(SMO,1);
        J=BNDL(UNSPEC(1),'0000000010000000'B,'0001'B);
/* KEYWORD ? */
        IF J=0 THEN
          GO TO SC137;
/* USER DEFINED */
        PR=SYM(SMO,3);
/* KEYWORD SYNTACTIC VALUE */
        SM=SMO;
        RETURN;
SC137:  PR=CLASS1;
/* USER VARIABLE */
        SM=SMO;
        RETURN;
SC140:
/* ENCOUNTERED A LITERAL IN STREAM */
        SW5=1;
/* LITERAL SWITCH */
        INLEN=0;
SC150:  CALL CHAR;
        IF CH1=... THEN
          GO TO SC170;
/* END OR EMBEDDED QUOTE */

```

```

3 1781
2 1782
3 1783
3 1783
4 1784
4 1785
4 1785
4 1786
4 1787
4 1788
2 1789
2 1790
2 1790
2 1791
2 1792
2 1792
3 1793
3 1793
3 1794
2 1795
2 1795
2 1796
2 1796
2 1797
2 1797
2 1797
3 1798
3 1799
2 1800
2 1801
2 1802
2 1802
2 1802
2 1803
2 1804
2 1804
2 1805
2 1806
3 1806
3 1806
3 1807
2 1808
2 1808
2 1809
2 1809
2 1810
2 1811
2 1812
2 1812
2 1813
2 1814
2 1814
2 1814
2 1815
2 1815
2 1816
3 1817
3 1818
2 1819

```

41

KSU'S PL/1 NEATENER AND PRECOMPILER

```

SCS160:      INLEN=INLEN+1;
              ** SHOULD CHECK LENGTH. */
              SJSTR(NAME,INLEN,1)=CH1;
              GO TO SCS150;
SCS170:      CALL CHAP;
              IF CH1=*** THEN
                GO TO SCS160;
              ** NOT END OF LITERAL. I IN. */
              SW2=1;
              ** WAS END. SET LOOK AHEAD ON. */
              SW5=0;
              ** LITERAL SWITCH OFF */
              ** ENTERED LITERAL INFORMATION IN SYMBOL TABLE.
              ** PUT LITERAL IN FILE FOR MATCHING. CALCULATE ITS
              DISPLACEMENT FROM THE LITERAL BASE, THEN UPDATE BASE. */
              IF SYMFL=0 THEN
                DO;
                  ER=-10;
                  CALL ERROR;
                END;
              SW=SYMFL;
              SYMFL=SYM(SYMFL,5);
              SET#=#SW;
              SETC=LTRL;
              STATA=BINARY('0011000000000000'B);
              CALL SEIT;
              PUT FILE(SYSPRINT)EDIT('LITERAL IN FILE ',NAME,' AT + ',LTRL
              )(SKIP,A,A,F(3));
              LTRL=LTRL+(INLEN+2)/2;
              ** INLEN IS LITERAL LENGTH */
              PUT EDIT('LENGTH = ',INLEN,' NEW LTOSP =',LTOSP(X(5),A,F(3))
              ),A,F(3));
              PR=CLASSZ;
              ** CLASS SYNTAX VALUE */
              RETURN;
              ** LITERAL HAS BEEN PROCESSED. */
SCS190:      ** PROCESS A CONSTANT */
              SJSTR(NAME,INLEN,1)=CH1;
              SW1=0;
              ** BLANK IGNORE OFF */
              VALUE=VALUE*10+INDEX(STR,CH1)-1;
              CALL CHAR;
              IF CH1=' ' THEN
                SW2=1;
              ** LOOK AHEAD ON */
              IF CH1>'9' THEN
                GO TO SC200;
              IF CH1<'0' THEN
                GO TO SC200;
              SW2=0;
              ** LOOK AHEAD OFF */
              INLEN=INLEN+1;
              IF INLEN>9 THEN
                DO;
                  ER=100;
                  CALL ERROR;
                END;
              ** CONSTANT TRUNCATED */

```


44

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

/* PROCEDURE STATEMENT ->
LABEL
PROCEDURE */
/* *JUMP, */
OPTYP=101;
CALL GENER;
/* JUMP AROUND PROCEDURE BODY */
CALL UNAME;
/* ASSIGN LABEL VALUE */
OPTYP=22;
/* LABEL REFERENCE NON-THREADED */
OPVAL=UNIOE;
CALL GENER;
SFT#=2;
/* GENERATE PROC LABEL */
CALL SETUP;
OPTYP=20;
/* STATEMENT LABEL */
OPVAL=SFTC;
CALL GENER;
/* PROC BLOCK START */
CALL REDUCE;
/* REDUCE QUEUES */
CALL BLKEN;
/* PUSH NEW BLOCK HEADER */
SETD=UNIOE;
/* REMEMBER LABEL AFTER BLOCK */
SETR=16;
/* STATEMENT TYPE */
CALL SETIT;
GO TO MATCH;
/* ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** *
S115;
/* BEGIN STATEMENT ->
LABEL
BEGIN
SET#=2;
CALL SETUP;
OPTYP=20;
/* GENERATE LABEL */
OPVAL=SETC;
/* LABEL VALUE */
CALL GENER;
CALL REDUCE;
/* REDUCE QUEUES */
CALL BLKEN;
/* CREATE NEW BLOCK */
SETR=17;
/* STATEMENT TYPE */
CALL SETIT;
/* SET# SET FROM REDUCE */
GO TO MATCH;
/* ** ** ** *
S120;
/* BEGIN STATEMENT ->
/* CALL REDUCE;
CALL BLKEN;
SETR=17;
BEGIN
1 1937
1 1937
1 1937
1 1937
1 1938
1 1939
1 1939
1 1940
1 1941
1 1941
1 1942
1 1943
1 1944
1 1944
1 1945
1 1946
1 1947
1 1948
1 1948
1 1949
1 1949
1 1950
1 1950
1 1951
1 1951
1 1952
1 1952
1 1953
1 1954
1 1954
1 1954
1 1954
1 1954
1 1955
1 1956
1 1957
1 1957
1 1958
1 1958
1 1959
1 1960
1 1960
1 1961
1 1961
1 1962
1 1962
1 1963
1 1963
1 1964
1 1964
1 1964
1 1965
1 1966

```

45

KSUP'S PL/I NEATENER AND PRECOMPILER

PAGE

```

/* STATEMENT TYPE */
CALL SETIT;
GO TO MATCH;
/* ***** */
S125:
/* LABEL ->
IDENTIFIER
:
*/
/* CHECK FOR ALREADY OTHER TYPE */
SET#2:
CALL SETUP;
I=0001(UNSPEC(SETA),'0111011111100000'B,'0001'B);
/* LABEL */IF I=0 THEN
GO TO S125A;
/* OK, NO CONFLICTS */
FP=10;
/* ATTRIBUTE CONFLICT */
CALL ERROR;
S125A: SETA=BINARY('1000100000000000'B);
/* LABEL */
/* RE-PUT IT */
IF SETC=0 THEN
GO TO S125C;
/* ALREADY ASSIGNED */
CALL UNAME;
SETC=UNIQF;
S125R: IF SETD=0 THEN
GO TO S125D;
/* END OF THREAD */
CODE(SETD,21)=SETC;
SETD=CODE(SETD,3);
GO TO S125R;
S125C: FR=13;
/* DUPLICATE LABEL */
CALL ERROR;
S125D: CALL SETIT;
/* VALUE AND NO THREAD */
CALL REDUCE;
CALL SETIT;
CALL BUG5;
GO TO MATCH;
/* ***** */
S130:
/* STATEMENT LIST ->
STATEMENT LIST
STATEMENT
*/
S135:
/* STATEMENT LIST ->
STATEMENT
*/
SET#1:
CALL SETUP;
/*GET STATEMENT TYPE */
CALL REDUCE;
/* REDUCE QUEUES */
CALL SETIT;
/* KEEP AS LAST STATEMENT TYPE */
GO TO MATCH;

```

1 1967
1 1967
1 1968
1 1969
1 1969
1 1969
1 1969
1 1969
1 1969
1 1969
1 1970
1 1971
2 1972
2 1973
1 1974
1 1974
1 1975
1 1975
1 1976
1 1977
1 1977
2 1978
2 1978
2 1979
1 1980
1 1980
1 1981
2 1982
2 1983
1 1984
1 1984
1 1985
1 1986
1 1987
1 1988
1 1988
1 1989
1 1990
1 1990
1 1991
1 1992
1 1993
1 1994
1 1994
1 1994
1 1994
1 1994
1 1994
1 1994
1 1994
1 1995
1 1996
1 1996
1 1997
1 1997
1 1998
1 1998


```

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
S140:
/* STATEMENT ->
GO
TO
IDENTIFIER
;
/* SET#2;
CALL SETUP;
/* IDENTIFIER INFO */
I=ROUL(UNSPEC(SFTA),01110111111111111111'B,0001'B);
IF I=0 THEN
GO TO S140B;
/* CANT RE LABEL */
OPTYP=101;
/* JUMP */
CALL GENER;
IF SETC=0 THEN
GO TO S140A;
/* DEFINED */
OPTYP=21;
/* LINK TO REFERENCES */
CALL GENER;
GO TO S140C;
S140A: OPTYP=22;
/* LABEL VALUE */
CALL GENER;
S140C: SFTA=HINARY('1000100000000000'B);
S140D: CALL SETIT;
SETR=1;
/* STMT TYPE */
CALL SETIT;
GO TO MATCH;
S140B: ER=11;
/* NOT REFERENCEABLE AS LABEL */
CALL ERROR;
GO TO S140D;
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */
S145:
/* STATEMENT ->
RETURN
;
/* OPTYP=100;
/* GENERATE RETURN */
CALL GENER;
CALL REDUCE;
SETR=2;
/* STATEMENT TYPE */
CALL SETIT;
GO TO MATCH;
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */
S150:
/* STATEMENT ->
IF STATEMENT
CALL REDUCE;
SETR=3;
/* STATEMENT TYPE */
CALL SETIT;

```

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

      GO TO MATCH;
/* ***** */
S155: /* STATEMENT ->
      ASSIGNMENT STATEMENT
      CALL REDUCE;
      SETR=4;
/* STATEMENT TYPE */
      CALL SETIT;
      GO TO MATCH;
/* ***** */
S160: /* STATEMENT ->
      DU STATEMENT
      CALL REDUCE;
      SETR=5;
/* STATEMENT TYPE */
      CALL SETIT;
      GO TO MATCH;
/* ***** */
S165: /* STATEMENT ->
      I/O STREAM STATEMENT
      CALL REDUCE;
      SETR=6;
/* STATEMENT TYPE */
      CALL SETIT;
      GO TO MATCH;
/* ***** */
S170: /* STATEMENT ->
      I/O RECORD STATEMENT
      CALL REDUCE;
      SETR=7;
/* STATEMENT TYPE */
      CALL SETIT;
      GO TO MATCH;
/* ***** */
S175: /* STATEMENT ->
      OPEN CLOSE STATEMENT
      CALL REDUCE;
      SETR=8;
/* STATEMENT TYPE */
      CALL SETIT;
      GO TO MATCH;
/* ***** */
S180: /* STATEMENT ->
      DECLARATION STATEMENT
      CALL REDUCE;
      SETR=9;
/* STATEMENT TYPE */
      CALL SETIT;
      GO TO MATCH;
/* ***** */
S185: /* STATEMENT ->

```

```

1 2032
1 2033
1 2033
1 2033
1 2033
1 2034
1 2035
1 2035
1 2036
1 2037
1 2037
1 2037
1 2037
1 2038
1 2039
1 2039
1 2040
1 2041
1 2041
1 2041
1 2042
1 2043
1 2043
1 2044
1 2045
1 2045
1 2045
1 2046
1 2047
1 2047
1 2048
1 2049
1 2049
1 2049
1 2049
1 2049
1 2050
1 2051
1 2051
1 2052
1 2053
1 2053
1 2053
1 2054
1 2055
1 2055
1 2056
1 2057
1 2057
1 2057

```


50

KSU'S PL/I NFATENER AND PRECOMPILER

PAGE

```

/* PUT OUT CONSTANT */
CALL REDUCE;
SETR=15;
CALL SETIT;
GO TO MATCH;
/* ***** */
S199:
/* STATEMENT -> STOP: */
OPTYP=128;
/* STOP */
CALL GENR;
CALL REDUCE;
SETR=15;
CALL SETIT;
GO TO MATCH;
/* ***** */
S200:
/* IF STATEMENT ->
IF CLAUSE
STATEMENT */
SET#=2;
CALL SETUP;
CALL CLAUSE INFO */
OPTYP=20;
/* STATEMENT LABEL */
OPVAL=SETC;
CALL GENR;
CALL REDUCE;
SETR=3;
/* STATEMENT TYPE */
CALL SETIT;
GO TO MATCH;
/* ***** */
S205:
/* IF STATEMENT ->
TO THE PART
STATEMENT */
SET#=2;
CALL SETUP;
OPTYP=20;
/* STATEMENT LABEL */
OPVAL=SETD;
/* ARD AND FALSE */
CALL GENR;
CALL REDUCE;
SETR=3;
/* STATEMENT TYPE */
CALL SETIT;
GO TO MATCH;
/* ***** */
S210:
/* IF CLAUSE ->
IF
LOGICAL EXPRESSION
THEN */
OPTYP=103;
/* '87' */
CALL GENR;

```

1 2124
1 2124
1 2125
1 2126
1 2127
1 2128
1 2128
1 2128
1 2129
1 2129
1 2130
1 2131
1 2132
1 2133
1 2134
1 2134
1 2134
1 2134
1 2134
1 2135
1 2136
1 2136
1 2137
1 2137
1 2138
1 2139
1 2140
1 2141
1 2141
1 2142
1 2143
1 2143
1 2143
1 2143
1 2144
1 2145
1 2146
1 2146
1 2147
1 2147
1 2148
1 2149
1 2150
1 2150
1 2151
1 2152
1 2152
1 2152
1 2152
1 2152
1 2153
1 2153

51

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

SET#2:
/* EXPRESSION INFO */
CALL SETUP:
OPTYP=10:
/* .DATA REFERENCE */
OPVAL=SETR:
/* DISPLACEMENT */
CALL GENER:
CALL REDUCE:
CALL UNAME:
SFTC=JNIOF:
/* LABEL AT FALSE BRANCH */
OPTYP=22:
OPVAL=UNIOF:
CALL GENER:
CALL SFTI:
GO TO MATCH:
/* ***** */
S215:
/* TRUE PART ->
IF CLAUSE
STATEMENT
ELSE */OPTYP=101:
/* JUMP */
CALL GENER:
CALL UNAME:
/* LABEL OVER FALSE PART */
OPTYP=22:
/* LABEL NOT TO BE THREADED */
OPVAL=UNIOF:
CALL GENER:
SFT#3:
/* IF CLAUSE */
CALL SETUP:
OPVAL=SETC:
/* LABEL AT FALSE */
OPTYP=20:
CALL GENER:
CALL REDUCE:
SFTD=JNIOF:
/* SAVE LABEL OVER FALSE */
CALL SETUP:
GO TO MATCH:
/* ***** */
S230:
/* IF STATEMENT ->
ITERATION LIMITER
STATEMENT LIST
END
: */ OPTYP=101:
/* JUMP */
CALL GENER:
OPTYP=22:
/* LABEL REFERENCE - UNCHAINED */
SFT#4:
/* ITER INFO */
CALL SETUP:
OPVAL=SETD:

```

1 2154
1 2155
1 2155
1 2156
1 2157
1 2157
1 2158
1 2158
1 2159
1 2160
1 2161
1 2162
1 2162
1 2163
1 2164
1 2165
1 2166
1 2167
1 2167
1 2167
1 2167
1 2167
1 2167
1 2168
1 2168
1 2169
1 2170
1 2170
1 2171
1 2171
1 2172
1 2172
1 2173
1 2174
1 2174
1 2175
1 2176
1 2176
1 2177
1 2178
1 2179
1 2180
1 2180
1 2181
1 2182
1 2182
1 2182
1 2182
1 2182
1 2183
1 2183
1 2184
1 2185
1 2185
1 2186
1 2186
1 2187

52

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

/* LABEL AT INCREMENT */
  CALL GENER;
  CPVAL=SETC;
/* BEYOND RANGE LABEL */
  OPTYP=20;
/* STATEMENT LABEL */
  CALL GENER;
  CALL REDUCE;
  SETR=5;
/* STATEMENT TYPE */
  CALL SFTT;
  GO TO MATCH;
/* ***** */
S235: /* ON 1 -> DO IDENTIFIER = EXPRESSION */
  SET#=3;
  /*CHECK FOR CONFLICTS*/CALL SETUP;
  I=BQCL(UNSPEC(SETA),'0011101111111111'B,'0001'B);
  IF I=0 THEN
    DO;
      FR=10;
      /*ATTRIBUTE CONFLICT*/CALL ERROR;
    END;
  J=SFT#;
  OPTYP=104;
  /*MOVF*/CALL GENER;
  SET#=1;
  /*EXPRESSION INFO*/CALL DATA1;
  /*PUT OUT EXPRESSION REF*/SET#=3;
  CALL DATA1;
  /*PUT IDEN.RFF.*/OPTYP=101;
  /*JUMP*/CALL GENER;
  CALL UNAME;
  SETC=UNIOE;
  OPTYP=22;
  CPVAL=UNIOE;
  /*LABEL AFTER INCREMENT*/CALL GENER;
  OPTYP=20;
  CALL UNAME;
  CPVAL=UNIOE;
  SETR=J;
  /*KEEP VAR.POINTER*/SETD=UNIOE;
  /*LABEL AT INCREMENT*/CALL GENER;
  CALL REDUCE;
  CALL SFTT;
  GO TO MATCH;
/* ***** */
/*DO 2->DO 1 BY EXPRESSION*//*ASSUME BOTH SP*/OPTYP=105;
  /*SADD*/CALL GENER;
  /*EXPR INFO,INCREMENT*/SET#=1;
  CALL SFTT;
  OPTYP=10;
/* DATA REFERENCE, LINK IT */
  CPVAL=SETC;
/* DISPLACEMENT ADDRESS */
  CPLNK=SFT#;
/* BLOCK POINTER */
  CALL GENER;

```

1 2188
1 2188
1 2189
1 2190
1 2190
1 2191
1 2191
1 2192
1 2192
1 2193
1 2194
1 2194
1 2195
1 2195
1 2196
1 2196
1 2196
1 2197
1 2197
1 2198
2 2199
3 2200
3 2201
3 2202
3 2203
1 2204
1 2205
1 2206
1 2207
1 2208
1 2209
1 2210
1 2211
1 2212
1 2213
1 2214
1 2215
1 2216
1 2217
1 2218
1 2219
1 2220
1 2221
1 2222
1 2223
1 2224
1 2225
1 2226
1 2227
1 2227
1 2228
1 2229
1 2230
1 2231
1 2232
1 2232
1 2233
1 2233
1 2234
1 2234

53

PAGE

KSU'S PL/I NEATENER AND PRECOMPILER

```

/* LOOP VARIABLE */
SET#=3;
/* DO 1 INFO */
CALL SETUP;
NPTYP=20;
/* STATEMENT LABEL */
OPVAL=SETC;
/* LABEL AFTER INCREMENT */
CALL GENER;
CALL REDUCE;
/* KEEP ALL INFO */
CALL SETI;
GO TO MATCH;
/* ***** */
5245:
/* ITERATION LIMITER ->
DO 2
TO
EXPRESSION */
NPTYP=106;
/* 'NG' */
CALL GENER;
SET#=1;
/* EXPRESSION INFO */
CALL SETUP;
J=SETC;
/* KEEP LIMIT BLOCK */
I=SETC;
/* KEEP LIMIT ADDRESS */
SET#=3;
/* GET DO 1 INFO */
CALL SETUP;
NPTYP=10;
/* LINKED DATA REFERENCE */
OPVAL=SETA;
/* LOOP VARIABLE ADDRESS */
OPLNK=SETC;
/* BLOCK HEADER POINTER */
CALL GENER;
/* ARG 1 */
NPTYP=10;
/* LINKED DATA REFERENCE */
OPVAL=1;
/* DISPLACEMENT ADDRESS */
OPLNK=J;
/* BLOCK POINTER */
CALL GENER;
/* ARG 2 */
CALL REDUCE;
CALL UNAME;
/* LABEL BEYOND RANGE */
SETC=UNIOE;
/* KEEP FOR LATER */
NPTYP=22;
/* ALSO OUT ARG 3 */
OPVAL=UNIOE;
/* UNLINKED LABEL REF. */
CALL GENER;

```

```

1 2235
1 2235
1 2236
1 2236
1 2237
1 2238
1 2238
1 2239
1 2239
1 2240
1 2241
1 2241
1 2242
1 2242
1 2243
1 2243
1 2243
1 2243
1 2243
1 2243
1 2244
1 2244
1 2245
1 2245
1 2246
1 2246
1 2247
1 2247
1 2248
1 2248
1 2249
1 2249
1 2250
1 2250
1 2251
1 2251
1 2252
1 2252
1 2253
1 2253
1 2254
1 2254
1 2255
1 2255
1 2255
1 2256
1 2256
1 2257
1 2257
1 2258
1 2258
1 2259
1 2259
1 2260
1 2260
1 2261
1 2261
1 2262
1 2262
1 2263
1 2263
1 2264
1 2264

```


55

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

/* PUT OUT EXPRESSION REFERENCE */
OPTYP=40;
/* BIT WORD CONSTANT OUT */
OPVAL=SETA;
CALL GENER;
SET#=3;
/* DEVICE PHRASE INFO */
CALL SETUP;
CALL REDUCE;
/* KEEP LABEL AT FMT LIST */
CALL SETIT;
GO TO MATCH;
/* ***** */
S285:
/* I/O STREAM LIST ->
I/O STREAM LIST
,
EXPRESSION */
/* GENERATE EXPRESSION ADDRESS, THEN BIT WORD DESC. */
SET#=1;
/* EXPRESSION INFO */
CALL DATA;
/* PUT OUT EXPRESSION REFERENCE */
OPTYP=40;
/* BIT WORD OUT */
OPVAL=SETA;
CALL GENER;
SET#=3;
/* STREAM INFO, FMT LABEL */
CALL SETUP;
CALL REDUCE;
CALL SETIT;
GO TO MATCH;
/* ***** */
S290:
/* FORMAT STREAM LIST ->
I/O STREAM LIST
)
(
FORMAT ITEM */
SET#=4;
/* NEED FMT LIST LABEL */
CALL SETUP;
OPTYP=20;
/* STATEMENT LABEL */
OPVAL=SETC;
CALL GENER;
CALL UNAME;
/* GET LABEL NEXT STATEMENT */
OPTYP=22;
/* UNCHAINED LABEL OFF. */
OPVAL=UNIOE;
CALL GENER;
SET#=1;
/* NOW FMT ITEM */
CALL SETUP;
OPTYP=40;
/* CONSTANT */

```

```

1 2293
1 2293
1 2294
1 2294
1 2295
1 2295
1 2296
1 2296
1 2297
1 2297
1 2298
1 2298
1 2299
1 2299
1 2300
1 2301
1 2301
1 2301
1 2301
1 2301
1 2301
1 2301
1 2302
1 2302
1 2303
1 2303
1 2303
1 2304
1 2304
1 2305
1 2305
1 2306
1 2307
1 2307
1 2307
1 2308
1 2308
1 2309
1 2309
1 2310
1 2311
1 2311
1 2311
1 2311
1 2311
1 2311
1 2311
1 2311
1 2311
1 2312
1 2312
1 2312
1 2313
1 2313
1 2314
1 2314
1 2315
1 2315
1 2316
1 2316
1 2317
1 2317
1 2317
1 2318
1 2318
1 2319
1 2320
1 2321
1 2321
1 2322
1 2322
1 2323

```

56

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

      NPVAL=SETR;
/* DESCRIPTOR */
      CALL GENER;
      CALL REDUCE;
      SETC=UNIQ;
/* LABEL NEXT STATEMENT */
      CALL SETIT;
      GO TO MATCH;
/* ***** */
S295:
/* FMT STREAM LIST ->
FMT STREAM LIST
      FMT ITEM */
      SET#=1;
/* FMT ITEM INFO */
      CALL SFTUP;
      OPTYP=40;
      NPVAL=SETR;
/* DESCRIPTOR */
      CALL GENER;
      SET#=3;
      CALL SETUP;
      CALL SAME INFO;
      CALL REDUCE;
      CALL SETIT;
      GO TO MATCH;
/* ***** */
S300:
/* DEVICE PHRASE ->
GET
FILE
(
      DEVICE
      ) */ OPTYP=107;
/* *GET */
S300A: CALL GENER;
      SFT#=2;
/* DEVICE INFO */
      CALL SETUP;
      OPTYP=40;
/* CONSTANT */
      NPVAL=SETR;
/* DEVICE # */
      CALL GENER;
      CALL UNAME;
/* ASSIGN FMT LABEL */
      OPTYP=22;
/* UN-CHAINED */
      NPVAL=UNIQ;
      CALL GENER;
      CALL REDUCE;
      SETC=UNIQ;
/* SAVE FMT LABEL */
      CALL SETIT;
      GO TO MATCH;
/* ***** */
S301:

```

```

1 2323
1 2324
1 2324
1 2325
1 2326
1 2327
1 2327
1 2328
1 2329
1 2329
1 2329
1 2329
1 2329
1 2329
1 2329
1 2329
1 2329
1 2329
1 2330
1 2331
1 2332
1 2333
1 2333
1 2334
1 2335
1 2336
1 2336
1 2337
1 2338
1 2339
1 2339
1 2339
1 2339
1 2339
1 2339
1 2340
1 2341
1 2342
1 2342
1 2343
1 2344
1 2344
1 2345
1 2345
1 2346
1 2347
1 2347
1 2348
1 2348
1 2349
1 2350
1 2351
1 2352
1 2352
1 2353
1 2354

```



```

S340:
/* FORMAT ITEM ->
  TAR /* CALL REDUCE;
      SETR=4096;
      CALL SEITI;
      GO TO MATCH;
/* *****
S345:
/* FORMAT ITEM ->
  F /* CALL REDUCE;
      SETR=2048;
      CALL SEITI;
      GO TO MATCH;
/* *****
S350:
/* FORMAT ITEM ->
  A /* CALL REDUCE;
      SETR=1024;
      CALL SEITI;
      GO TO MATCH;
/* *****
S355:
/* FORMAT ITEM ->
  F
  (
    CONSTANT
  ) /*
  SETR=2;
  CALL SETUP;
  CALL REDUCE;
  SETR=2048*SETO;
  /* .CONST VALUE */
  CALL SEITI;
  GO TO MATCH;
/* *****
S360:
/* FORMAT ITEM ->
  A
  (
    CONSTANT
  ) /*
  SETR=2;
  CALL SETUP;
  CALL REDUCE;
  SETR=1024*SETO;
  /* CONSTANT VALUE */
  CALL SEITI;
  GO TO MATCH;
/* *****
S365:
/* FORMAT ITEM ->
  X
  (
    CONSTANT
  ) /*
  SETR=2;
  CALL SETUP;
  CALL REDUCE;
  SETR=512*SETO;
  /* CONSTANT VALUE */

```

65

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

CALL SFIT1:
GO TO MATCH:
/* ***** */
S400:
/* CHARACTER SEGMENT ->
SUBSTR
(
VARIABLE
,
EXPRESSION
,
EXPRESSION
) *1
=50: /* JMP LXXX+3
LXXX: */
CALL UNAME:
OPVAL=UNIOE:
CALL GENER:
CALL YLOC:
YLD1=6:
SET#6:
CALL DATAL:
YLD1=4:
CALL YLOSP:
SET#4:
CALL DATAL:
YLD1=2:
CALL YLOSP:
SET#2:
CALL DATAL:
SETC=UNIOE:
CALL SFIT1:
GO TO MATCH:
/* KEEP DOPE LABEL */
/* ***** */
S405:
/* PSEUDO VARIABLE ->
INDEX
(
VARIABLE
,
VARIABLE
) */
/* OPTYP=10:
/* INDEX: */
CALL GENER:
SET#4:
CALL DATAL:
/* REF TO FIRST VAR */
SET#2:
/* SECOND VARIABLE INFO */
CALL DATAL:
/* REF TO SECOND VAR */
CALL ASPT:
/* SP TEMPORARY */
/* BASE ON YIELD RESULT */
CALL DATA2:
/* REF TO TEMP */
CALL REDUCE:

```

60

KSU'S PL/I NEATENR AND PRECOMPILER

PAGE

```

SETA='0100000001000000'B:
/* TEMP-SP-INT */
SETC=TMPTR:
/* ADDRESS */
SETC=CCRH:
/* BLOCK POINTER */
CALL SFTIT:
GO TO MATCH:
/* ***** */
S410:
/* PSEUDO VARIABLE ->
LENGTH
(
VARIABLE
) */
/* 'LENTH' */
CALL GENER:
SET#2:
CALL DATA1:
/* REF TO VARIABLE */
CALL ASPT:
/* TMPTR RETURNED */
/* BASE ON YIELD RESULT */
/* REF TO TEMPORARY */
CALL REDUCF:
/* RELEASE TEMP CORE */
SETA='0100000001000000'B:
/* TEMP-SP-INTEGER */
SETC=TMPTR:
/* ADDRESS */
SETC=CCRH:
/* BLOCK POINTER */
CALL SETIT:
GO TO MATCH:
/* ***** */
S415:
/* PSEUDO VARIABLE ->
AND
(
VARIABLE
,
PRIMARY
) */
YLD1=4:
YLD2=2:
YLOTP='0100000000000000'B:
/* INTEGER OPERANDS */
CALL YIELD:
OPTYP=112:
/* 'SAND' */
S415A: CALL GENER:
SET#4:
CALL DATA1:
/* VAR REF */
SET#2:
/* PRIMARY */
CALL DATA1:

```

1 2441
1 2442
1 2442
1 2443
1 2443
1 2444
1 2444
1 2445
1 2446
1 2446
1 2446
1 2446
1 2446
1 2446
1 2446
1 2447
1 2447
1 2448
1 2449
1 2450
1 2451
1 2451
1 2452
1 2452
1 2453
1 2453
1 2454
1 2454
1 2455
1 2455
1 2456
1 2456
1 2457
1 2458
1 2458
1 2458
1 2458
1 2458
1 2458
1 2458
1 2459
1 2460
1 2461
1 2462
1 2463
1 2463
1 2464
1 2465
1 2466
1 2466
1 2467
1 2467

61

KSU'S PL/I NEATENR AND PRECOMPILER

PAGE

```

/* PRIMARY REF */
CALL ASPT;
/* SP TEMP AT TMPTR */
/* BASE ON YIELD RESULT */
CALL DATA2;
/* RESULT ADDRESS PFF */
CALL REDUCE;
/* REDUCE TO SP TEMP */
SETA=YLOTP;
/* TEMP-SP-INTEGER */
SETC=TMPTR;
/* ADDRESS */
SETC=CCRH;
/* BLOCK POINTER */
CALL SETIT;
GO TO MATCH;
/* ***** */
S420:
/* PSEUDO VARIABLE ->
OR
(
VARIABLE
,
PRIMARY
)
/* YLD1=4;
YLD2=2;
YLOTP=0100000000000000R;
/* WANT INTEGER OPERANDS */
CALL YIELD;
OPTYP=113;
/* 'SOR' */
GO TO S415A;
/* ***** */
S425:
/* PSEUDO VARIABLE ->
SWAP
(
VARIABLE
)
/* YLD1=2;
CALL YLOSP;
/* MUST NOW BE SP */
/* 'SWAP' */
CALL GENER;
SET#2;
CALL DATA1;
/* VARIABLE REFERENCE */
CALL DATA2;
/* RESULT ADDRESS REF */
CALL REDUCE;
SETA=+100000001000000R;
SETC=TMPTR;
/* TEMP ADDRESS */
SETC=CCRH;
/* BLOCK POINTER */
CALL SETIT;
GO TO MATCH;

```

1 2468
1 2468
1 2469
1 2469
1 2469
1 2470
1 2470
1 2471
1 2471
1 2472
1 2472
1 2473
1 2473
1 2474
1 2474
1 2475
1 2476
1 2476
1 2476
1 2476
1 2476
1 2476
1 2477
1 2478
1 2479
1 2479
1 2480
1 2481
1 2481
1 2482
1 2482
1 2482
1 2482
1 2482
1 2482
1 2483
1 2484
1 2484
1 2485
1 2485
1 2486
1 2487
1 2488
1 2488
1 2489
1 2490
1 2490
1 2491
1 2492
1 2493
1 2493
1 2494
1 2494
1 2495

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE 63

```

CALL YLDSP;
OPTYP=108;
/* 'SUR1' */
CALL GENER;
SFT#4;
/* IDENTIFIER INFO */
/* CHECK FOR ARRAY */
CALL DATA;
/* PUT IDENTIFIER REF */
I=SYM(SET#4);
/* PUT ELEMENT LEN CONSTANT */
OPVAL=SYM(1,5);
/* STORE LENGTH AS CONSTANT */
S455A: OPTYP=40;
/* CALL GENER;
/* PUT OUT CONSTANT */
CALL UNAME;
/* GET A LOCATION LABEL TO STORE CALC ADDR */
OPTYP=22;
/* UNTHREADED LABEL */
OPVAL=UNIQUE;
CALL GENER;
/* OUTPUT FORWARD REFERENCE */
SET#2;
/* EXPRESSION INFO */
CALL DATA;
/* PUT EXPRESSION REF */
SET#4;
CALL SETUP;
I=SYM(SETD,1);
/* POINTER TO CONSTANT ELEMEN */
OPVAL=SYM(1,4);
/* NUMERIC VALUE */
OPTYP=40;
CALL GENER;
CALL REDUCE;
SFTC=UNIQUE;
SETA=POOL(LUNSPEC(SETA),'0110001000000000'8,'0001'R);
SETA=POOL(LUNSPEC(SETA),'0000000000100000'8,'0111'R);
CALL SETTI;
GO TO MATCH;
/* ***** */
S460:
/* SUBSCRIPTED VARIABLE ->
IDENTIFIER
(
EXPRESSION
EXPRESSION
) */
VLD1=4;
CALL YLDSP;
VLD1=2;
CALL YLDSP;
OPTYP=109;
/* 'SUR2' */
CALL GENER;
SFT#6;

```

```

1 2530
1 2531
1 2532
1 2532
1 2533
1 2534
1 2534
1 2534
1 2535
1 2535
1 2535
1 2536
1 2536
1 2537
1 2537
1 2538
1 2538
1 2539
1 2539
1 2540
1 2540
1 2541
1 2541
1 2542
1 2542
1 2543
1 2543
1 2544
1 2544
1 2545
1 2545
1 2546
1 2546
1 2547
1 2547
1 2548
1 2548
1 2549
1 2549
1 2550
1 2551
1 2552
1 2553
1 2554
1 2554
1 2555
1 2555
1 2556
1 2556
1 2557
1 2557
1 2557
1 2557
1 2557
1 2557
1 2557
1 2557
1 2557
1 2557
1 2558
1 2558
1 2559
1 2560
1 2561
1 2562
1 2562
1 2563

```

64

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

/* CHECK FOR ARRAY */
CALL DATA1;
/* IDENTIFIER ADDRESS */
J=SYM(SET#4);
/* DOPE PTR */
I=SYM(J,5);
/* CONSTANT LENGTH POINTER */
IF I=0 THEN
  NPVAL=1;
ELSE
  NPVAL=SYM(I,4);
/* LENGTH VALUE */
NPVAL=40;
CALL GNER;
CALL UNAME;
/* LABEL TO KEEP COMPUTED ADDRESS AT */
NPVAL=22;
/* UNTHREADED LABEL */
NPVAL=UNTOE;
CALL GNER;
/* FORWARD LABEL REFERENCE */
SET#4;
/* FIRST EXPRESSION */
CALL DATA1;
/* EXPRESSION ADDR */
I=SYM(J,1);
/* PTR FIRST CONSTANT DIMENSION */
NPVAL=40;
NPVAL=SYM(I,4);
CALL GNER;
SET#2;
/* SECOND SUBSCRIPT EXPRESSION */
CALL DATA1;
/* SECOND EXPRESSION */
I=SYM(J,2);
/* CONSTANT SECOND DIMENSION SIZE */
NPVAL=40;
NPVAL=SYM(I,4);
CALL GNER;
SET#6;
/* IDENTIFIER INFO */
CALL SETUP;
/* KEEP INFO */
CALL REJICE;
/* REDUCE TO SUBSCRIPTED VARIABLE */
SETC=UNTOE;
/* CALCULATED ADDRESS WORD ADDRESS */
SETA=ROOT(UNSPEC(SETA),'0110001000000000'B,'0001'B);
SETA=ROOT(UNSPEC(SETA),'000000000100000'B,'0111'B);
CALL SETIT;
GO TO MATCH;
/* ***** */
S500;
/* ASSIGNMENT STATEMENT ->
PRIMARY
=
EXPRESSION */
SET#3;

```

```

1 2564
1 2564
1 2565
1 2565
1 2566
1 2566
2 2567
2 2567
2 2568
2 2569
2 2569
1 2570
1 2570
1 2571
1 2572
1 2573
1 2573
1 2574
1 2574
1 2575
1 2576
1 2576
1 2577
1 2577
1 2578
1 2578
1 2579
1 2579
1 2579
1 2580
1 2581
1 2582
1 2583
1 2583
1 2584
1 2584
1 2585
1 2585
1 2586
1 2587
1 2587
1 2588
1 2589
1 2589
1 2590
1 2590
1 2591
1 2591
1 2592
1 2592
1 2593
1 2594
1 2595
1 2596
1 2596
1 2596
1 2596
1 2596

```

65

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

CALL SETUP:
YLDTP=SETA:
/* TARGET ATTRIBUTES */
YLD1=3:
YLD2=1:
CALL YIELD:
OPTYP=104:
/* 'MOVE' */
CALL GENER:
SFT#1:
CALL DATA1:
SFT#3:
CALL DATA1:
CALL REDUCE:
SETR#4:
/* STATEMENT TYPE */
CALL SETIT:
GO TO MATCH:
/* ***** */
SRO5:
/* LEFT PART -> VARIABLE */
CALL TRVIA:
GO TO MATCH:
/* ***** */
SRI5:
/* EXPRESSION ->
TERM */SETR#1:
CALL SETUP:
CALL REDUCE:
CALL SETIT:
GO TO MATCH:
/* ***** */
S520:
/* EXPRESSION ->
ADDING OPERATOR
TERM */SETR#2:
CALL SETUP:
IF SETR#1 THEN
GO TO S520A:
/* +? */
/* IF UNARY + THEN SIMPLY REDUCE */
OPTYP=115:
/* 'NEG' */
CALL GENER:
SFT#1:
/* TERM INFO */
CALL DATA1:
/* TERM REFERENCE */
CALL ASPT:
/* SP TEMP */
CALL DATA2:
/* RESULT ADDRESS */
SETC=TMPTR:
SETR#CCRM:
S520A: CALL REDUCE:
/* INFO EITHER SAME OR NEW */
CALL SETIT:
GO TO MATCH:

```


69

KKSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

SFT#:=4;
/* FIRST VAR REFERENCE */
CALL DAT1;
SFT#:=2;
/* SECOND VAR REFERENCE */
CALL DAT1;
CALL REDUCE;
SET#:=8;
/* STMT TYPE */
CALL SET1;
GO TO MATCH;
/* ***** */
S$90:
/* OPEN CLOSE STATEMENT TYPE -> CLOSE .... */
OPTYP:=126;
/* 'CLOSE' */
GO TO S$80A;
/* ***** */
S$20:
/* I/O RECORD STATEMENT ->
READ
FILE
(
VARIABLE
)
RECORD KEY */
OPTYP:=123;
/* ' GET30' */
S$20A: CALL GENER;
SET#:=3;
/* PUT OUT VARIABLE REFERENCE */
CALL DAT1;
CALL REDUCE;
/* NO INFO KEPT */
GO TO MATCH;
/* ***** */
S$25:
/* I/O RECORD STATEMENT -> WRITE .... */
OPTYP:=124;
/* 'PUT30' */
GO TO S$20A;
/* ***** */
S$30:
/* RECORD KEY ->
(
VARIABLE
) */
YLD1:=2;
CALL YLDSP;
/* MUST 9F SP */
OPTYP:=122;
/* 'LOAD' */
CALL GENER;
SET#:=2;
/* PUT OUT A VARIABLE REFERENCE */
CALL DAT1;
CALL REDUCE;
GO TO MATCH;

```


02

KSU'S PL/I NEATENER AND PRECOMPILER

```

2 2752 /* NO INFO KEPT */
2 2752 S660:
2 2752 /* OCL STATEMENT ->
2 2752 OCL PAOT
2 2752 OPTION
2 2752
2 2752 BEGIN:
2 2752 DECLARE (I1,I2,I3) BIN FIXED(15);
2 2752 SET#1:
2 2752 /* OPTION INFO */
2 2752 CALL SETUP:
2 2752 I1=SETA:
2 2752 /* OPTION BIT WORD */
2 2752 I2=SETD:
2 2752 /* ELF. LEN */
2 2752 SET#2:
2 2752 CALL SETUP:
2 2752 I3=BOOL(UNSPEC(SETA),'0010000000000000'R,'0001'B):
2 2760 IF I3=0 THEN
3 2761 I2=(I2+2)/2;
3 2761 IF SETD=0 THEN
3 2763 GO TO S660R;
3 2764 /* NOT SUBSCRIPTED */
2 2765 S660A:
2 2765 SYM(SETD,5)=I2;
2 2765 /* SAVE LENGTH FOR SP, DP, AND CHAR ARR*/
2 2766 I3=SYM(SETD,1);
2 2766 /* PTR FIRST DIM CONSTANT */
2 2767 I2=I2*SYM(I3,4);
2 2768 /* LEN/ELFM * 1 ST DIM */
2 2768 I3=SYM(SETD,2);
2 2768 IF I3=0 THEN
3 2769 I2=I2*SYM(I3,4);
3 2770 /* INCLUDE 2ND SUB */
2 2771 S660R:
2 2771 SFTC=SYM(CCBH,4);
2 2772 /* NEXT FREE LOC THIS BLOCK */
2 2772 SYM(CCBH,4)=SYM(CCBH,4)+I2;
2 2773 /* ALLOCATE STORAGE FOR ARRAY OR SCALAR */
2 2773 CALL REDUCE:
2 2774 /* MERGE INFO */
2 2774 SETA=BOOL(UNSPEC(SETA),'1000000000000000'R,'0111'B):
2 2774 /* MERGE OPTION INFO TOO */
2 2775 SETA=BOOL(UNSPEC(I1),UNSPEC(SETA),'0111'B):
2 2775 SET#3=SETA:
2 2776 /* POINTER TO ID */
2 2777 SFTF=SYM(SFT#1,5);
2 2777 /* SAVE ID LINK */
2 2778 SFTF=SYM(SET#2,2);
2 2779 /* CODE POINTER */
2 2779 CALL SETIT:
2 2780 GO TO WATCH:
2 2781 END:
2 2781
1 2782 S665:
1 2782 /* OCL PART ->
1 2782 DECLARE
1 2782 IDENTIFIER */
1 2782 SET#1:
1 2782

```



```

CALL GENPR;
SFT#2;
/* COMPARISON TYPE */
CALL SETUP;
OPTYP=40;
/* CONSTANT AS ARG */
OPVAL=SETB;
/* OPTYP */
CALL GENPR;
SET#1;
/* FIRST EXPRESSION INFO */
CALL DATA1;
/* PUT OUT ITS REFERENCE */
SET#2;
/* SECOND EXPRESSION INFO */
CALL DATA1;
/* PUT OUT ITS REFERENCE */
CALL ASPT;
/* ADDRESS FOR BOOLEAN RESULT */
CALL DATA2;
/* PUT OUT ITS REFERENCE */
SET#TMPTR;
/* THIS HAS NOT BEEN SAVED*** */
CALL REDUCE;
CALL SFTIT;
GO TO MATCH;
/* ***** */
S705:
/* LOGICAL OP -> < */
LOT=1;
/* TYPE =1 */
GO TO S705A;
S710:
/* LOGICAL OPERATOR -> > */
LOT=2;
/* TYPE =2 */
GO TO S705A;
S715:
/* LOGICAL OP -> <= */
/* TYPE =3 */
GO TO S705A;
S720:
/* LOGICAL OP -> >= */
LOT=4;
/* TYPE =4 */
GO TO S705A;
S725:
/* LOGICAL OP -> = */
LOT=5;
/* TYPE =5 */
GO TO S705A;
/* ***** */
S726:
/* LOGICAL OP -> ~= */
LOT=6;
/* TYPE =6 */
S705A: SFT#1;
CALL SETUP;

```

LOT=3

**THIS BOOK WAS
BOUND WITH TWO
PAGES NUMBERED
172. THESE PAGES
ARE DIFFERENT.**

**THIS IS AS
RECEIVED FROM
CUSTOMER.**

APPENDIX VII

Reductions Program

The reductions table generator is extremely easy to use. This appendix contains both a listing of the program and a brief description of the data input requirements.

This program, as described in Chapter III, generates a structured table useful for a certain type of syntactic scan scheme called reductions analysis. This is a bottom-up deterministic algorithm which assumes a Type 2 context free grammar with unique right hand sides. Input to the program consists of a list of terminal symbols, a list of the nonterminal symbols, and a coded set of production rules defining the language to be recognized. Output from the program is a list of the terminals, a list of the nonterminals, an echo of the numerically coded productions, a more readable character representation of the same productions, a listing of the left hand sets of all nonterminals, and both a literal listing and a tabular listing of the reductions rules derived from the set of productions. The card punching format requirements of the input data are described below.

A: HEADER.

The first three cards supplied as input to the program are centered and output as header information at the top of each section of output. These cards may contain any information the user wishes.

B: TERMINALS.

All terminal symbols which appear in the strings to be recognized must be input with an associated terminal number. Terminal numbers must be assigned sequentially starting at 500, one per card, the list being

terminated with a blank terminal symbol. For example, to assign the value 503 to the terminal string 'PROCEDURE', the following card would be punched:

```
503 'PROCEDURE'
```

C: NONTERMINALS.

Nonterminal symbols are input exactly like terminals, but must be numbered consecutively starting with the value 1. These strings are used only to improve the readability of the printed output. For example, the string nonterminal 'PROCEDURE BLOCK' would appear as follows:

```
1 'PROCEDURE BLOCK'
```

D: PRODUCTIONS.

The set of productions used to define the language is coded numerically, and each single rewrite rule is assigned a unique and somewhat arbitrary number between 1 and 999. Again, the production number is useful for determining which reduction a matches have been satisfied and dispatching accordingly. Each production is coded on a separate card and read with the (PL/I) format '(25)f(3)', which includes the production number. A sample production card might appear as follows:

```
c      c
c      c
1      9
100001503
```

where the fields have the following meaning:

<u>column</u>	<u>value</u>	<u>meaning</u>
1-3	100	User assigned production number.
4-6	001	Nonterminal left hand side of the production, here representing 'PROCEDURE BLOCK'
7-9	503	First (and only) symbol on the right hand side of the production.
10-12	xxx	Second symbol on right hand side.
13-15	xxx	Addition symbols as needed.

The set of productions is terminated by an all-zero production.

A sample set of input data is given below for the simple grammer presented in Chapter II.

```
//GO.SYSIN DD *
HEADER INFORMATION CARD 1.
HEADER INFORMATION CARD 2.
HEADER INFORMATION CARD 3.
500 '='
501 '+'
502 '*'
503 '**'
504 '1'
505 '2'
506 '3'
507 'A'
508 'B'
509 'C'
510 ' '
1 'S'
2 'E'
3 'T'
4 'F'
5 'P'
6 'V'
7 ' '
101001006500002
102002003
```

103002501003
104002002501003
105003004
106003003502004
107004005
108004004503005
109005504
110005505
111005506
112006507
113006508
114006509
0 0 0
/*

1

K&S PL/I NEATENER AND PRECOMPILER

PAGE

```

1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
10 1
11 1
12 1
13 1
14 1
15 1
16 1
17 1
18 1
19 1
20 1
21 1
22 1
23 1
24 2
25 2
26 2
27 2
28 2
29 3
30 3
31 2
32 1
33 1
34 1
35 1
36 1
37 2
38 3
39 3
40 3

CRANK:  PROC OPTIONS(MAIN);
        DECLARE PUNCH FILE STREAM OUTPUT;
        DECLARE FST BIT(1);
        DECLARE ORDER(100,-3:20) BINARY FIXED(15);
        DECLARE OUTSTR CHAR(130) VARYING;
        DECLARE DVEFC(-3:20) BINARY FIXED(15);
        DECLARE
        (KNTR,KNTR2,L1,L2,LEN,LV,NTRCX,PAR,TPS) BINARY FIXED(31,0),
        UPROD(200) BINARY FIXED,
        SEMAN(100,4) BINARY FIXED,
        (T,S1,T1,T2,#SEM,TF,TFTPS) BINARY FIXED,
        TRIP(2000,3) BINARY FIXED,
        TTL(3) CHAR(80),
        CARD CHAR(80),
        BLANKS CHAR(130) INIT(' '),
        TERM(500:700) CHAR(20) VARYING,
        #TERM BINARY FIXED(31,0),
        NTERM(1:199) CHAR(20) VARYING,
        #NTERM BINARY FIXED(31),
        ABBREV BIN FIXED(15),
        PRODS(200,-3:20) BINARY FIXED(15),
        #PROD BINARY FIXED(31);
        DECLARE LHSB(500:700) BIT(1);
        DECLARE SW BIT(1);
        DECLARE LHS(25000),LHSP(200) BINARY FIXED (15);
        DECLARE BITS(200) BIT(1);
        FORMAT(A,SKIP(0),9 A(1));
        FORMAT(F(3),X(2),A,SKIP(0),(100)A(1));
        FORMAT(SKIP,F(4),X(2),A,SKIP(0),X(6),(100)A(1));
        FORMAT(COL(1),(25)F(3));
        FORMAT(A,SKIP(0),(100)A(1));
        FORMAT(SKIP,(24)F(5));
        FORMAT(A,SKIP(0),(100)A(1));
        FORMAT(4 F(5));
        ON ENDOFILE(SYSIN)
        GO TO FINIS;
        TRIP(*,*)=0;
        OPEN FILE(SYSPRINT) LINESIZE(130);
        DO I=1 TO 3;
            GET EDIT(TTL(I))(COL(1),A(80));
            END;
        PROCEDURE;
        PUT PAGE;
        DO I=1 TO 3;
            PUT EDIT(TTL(I))(COL(25),A(80));
            END TITLE;
        /* INPUT TERMINAL SYMBOLS. */
        TERM(*)=**;
        I=499;
        I=I+1;
        GET LIST(J,TERM(I));
        IF J=I THEN
            DO;
                PUT SKIP EDIT(/** TERMINAL NUMBERING NOT CONSISTANT WITH
                'I,' EXPECTED INPUT,*,TERM(I),'-.-',J,*, BUT 'I,' EXPECTED
                '.,',SKIP(10),A,SKIP,A,A,F(4),A,F(4),A);
                STOP;
            END;
        END;
        INMS:

```

```

END;
IF TERM(I)=-1 THEN
  GO TO IN5;
#TERM=I-500;
CALL TITLE;
PUT SKIP(5);
PUT EDIT('TERMINALS', (('_' DO I=1 TO 9))(R(FMT3));
K=500;
DO I=1 TO #TERM;
  PUT SKIP EDIT(K,TERM(K))(R(FMT4));
  K=K+1;
END;
/* INPUT ALL NONTERMINALS. */
/* NUMBERED FROM 1 TO 199. */
/* THE FIRST QF WHICH MUST BE THE
SENTENTIAL SYMBOL. */
NTERM(*)=0;
I=0;
I=I+1;
GET LIST(J,NTERM(I));
IF J=-1 THEN
  DO;
    PUT EDIT('** NONTERMINAL NUMBERING NOT CONSISTANT WITH EX
    PECTED INPUT',NTERM(I),'.',J,' BUT',I,' EXPECTED. ');
    {SKIP(10),A,SKIP,A,A,F(4),A,F(4),A};
    STOP;
  END;
IF NTERM(I)=0 THEN
  GO TO IN8;
NTERM=I-1;
CALL TITLE;
PUT SKIP(5);
PUT EDIT('NONTERMINALS', (('_' DO I=1 TO 12))(R(FMT8));
DO I=1 TO #NTERM;
  PUT EDIT(I,NTERM(I), (('_' DO J=1 TO LENGTH(NTERM(I)))(R(FMT
  6)));
END;
/* GET IN CODED PRODUCTIONS. */
CALL TITLE;
PUT SKIP(5);
PUT EDIT('PRODUCTIONS', (('_' DO I=1 TO 11))(R(FMT8));
PRODS(*,*)=0;
UPROD(*)=0;
I=0;
IN12: I=I+1;
/* INPUT USER ASSIGNED PROD # AND PROD. */
GET EDIT(UPROD(I), (PRODS(I,K) DO K=1 TO 20))(R(FMT7));
DO J=1 TO 20;
  IF PRODS(I,J)=0 THEN
    GO TO L3;
END;
GO TO IN14;
L3: PUT EDIT('PRODS(I,K) DO K=1 TO 20))(R(FMT9));
GO TO IN12;
IN14: #PROD=I-1;
/* THE ZERO' TH ELEMENT OF A PRODUCTION ROW IS THE # OF SYMBOLS
IN THE ENTIRE PRODUCTION. */

```

41 3
42 2
43 2
44 1
45 1
46 1
47 1
48 1
49 2
50 2
51 2
52 2
53 1
54 1
55 1
56 1
57 2
58 3
59 3
60 3
61 3
62 2
63 2
64 1
65 1
66 1
67 1
68 1
69 2
70 2
71 2
72 1
73 1
74 1
75 1
76 1
77 1
78 1
79 1
80 2
81 3
82 3
83 2
84 1
85 1
86 1
87 1
88 2
89 2
90 2

3

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

DO I=1 TO #PROD;
  DO K=1 TO 20;
    IF PRODS(I,K)=0 THEN
      GO TO IN16;
    END;
    PRODS(I,0)=K-1;
  END;
END;
/* NOW ECHO PRODUCTIONS IN A CHARACTER FOR
   */
CALL TITLE;
PUT SKIP(5);
PUT EDIT('PRODUCTIONS',('_'_))DO I=1 TO 11)((R(FMT10)));
DO I=1 TO #PROD;
  PUT SKIP;
  IF UPPOD(I)=0 THEN
    PUT EDIT('UPPOD(I)',UPPOD(I)),'')(SKIP,A,F(3),A);
  ELSE
    PUT EDIT('')(SKIP,A);
  DO J=1 TO PRODS(I,0);
    IF PRODS(I,J)>500 THEN
      PUT EDIT('TERM(PRODS(I,J))')(X(1),A);
    ELSE
      PUT EDIT('<NTERM(PRODS(I,J))>')(X(1),A,A,A);
    IF J=1 THEN
      PUT EDIT('::')(X(1),A);
    END;
  END;
END;
/* INPUT USER DEFINED SEMANTIC CALLS. */
/* SEMANTIC SUBSTITUTIONS ARE FORMATED AS FOLLOWS:
   COL 1-5 TOP OF PARSE STACK SYMBOL.
   COL 6-10 USER ASSIGNED PRODUCTION #
   COL 11-15 0 OR 1 (SUCCESS / FAILURE BRANCH)
   COL 16-20 SEMANTIC ROUTINE CALL #
   */
SEMANI(*,*)=0;
S1=0;
IN17: S1=S1+1;
GET EDIT('SEMANI',I)DO I=1 TO 4)((4)F(5));
DO J=1 TO 4;
  IF SEMANIS1,J)=0 THEN
    GO TO IN17;
  END;
#SEM=S1-1;
/* GENERATE LEFT HAND SIDES AND ASSOCIATED POINTERS.
   NOTE-- EVERY NONTERMINAL HAS A LHS.
   */
LHS(*)=0;
LHSP(*)=0;
L1,L2=1;
DO I=1 TO #NTERM;
  LHSR(*)=0;
  BITS(*)=0;
  DO J=1 TO #PROD;
    IF PRODS(J,1)=I THEN
      BITS(J)=1;
    END;
  SH=0;
  DO J=1 TO #PROD;
    IF-BITS(J)THEN

```

88 2
89 3
90 4
91 4
92 3
93 2
94 2
95 1
96 1
97 1
98 2
99 2
100 3
101 3
102 3
103 3
104 4
105 4
106 4
107 4
108 4
109 3
110 2
111 1
112 1
113 1
114 1
115 2
116 3
117 3
118 2
119 1
120 1
121 1
122 1
123 2
124 2
125 2
126 3
127 4
128 4
129 3
130 2
131 3
132 4

4

PAGE

KSU'S PL/I NEATENER AND PRECOMPILER

```

      GO TO IN20;
      DO K=1 TO #PROD;
      IF (PRODS(K,1)=PRODS(J,2)&=BITS(K)) THEN
      DO;
      BITS(K)=1'B;
      SW=1'B;
      END;
      END;
      IF SW THEN
      GO TO IN18;
      LHSP(L2)=L1;
      L2=L2+1;
      DO J=1 TO #PROD;
      IF (PRODS(J,2)>=500&BITS(J)) THEN
      IF (-LHSP(PRODS(J,2))) THEN
      DO;
      LHSP(L1)=PRODS(J,2);
      L1=L1+1;
      LHSP(PRODS(J,2))=1'B;
      END;
      ELSE
      ;
      ELSE
      ;
      END;
      LHSP(L1)=L1;
      L1=L1+1;
      END;
      CALL TITLE;
      PUT SKIP(5);
      PUT EDIT('LEFT HAND SIDES',(' ',J)DO J=1 TO 15)) (X(25),A,SKIP(0)
      ,X(25), (15)A(1));
      DO I=1 TO #TERM;
      PUT EDIT(INTERM(I),(' ',J)DO J=1 TO LENGTH(INTERM(I))) (SKIP(1)
      ,X(5),A,SKIP(0),X(5), (100)A(1));
      J=LHSP(I);
      K=20;
      PUT EDIT(' ',COL(20),A(1));
      IF LHSP(J)=1 THEN
      GO TO LOOPB;
      PUT EDIT(INTERM(LHSP(J)),(' ',J)(A,A);
      K=K+LENGTH(INTERM(LHSP(J)))+1;
      IF K>100 THEN
      DO;
      K=20;
      PUT SKIP EDIT(' ',COL(20),A);
      END;
      J=J+1;
      GO TO LOOPA;
      END;
      LOOPB:
      /* INITIALIZE JUMP KEY OF TRIP TABLE. */
      1 POSITION & MATCH
      POSITION 1 - 20, -1
      MATCH 1 - 1023
      VALUE = POSITION*1024 +/- MATCH

```

4 133
4 134
5 135
6 136
6 137
6 138
6 139
4 140
3 141
3 142
3 143
2 144
2 145
3 146
4 147
5 148
6 149
6 150
6 151
6 152
6 153
5 154
5 155
4 156
2 157
2 158
2 159
1 160
1 161
1 162
1 163
1 163
2 164
2 165
2 165
2 166
2 167
2 168
3 169
3 170
2 171
2 172
3 173
4 174
4 175
4 176
4 177
2 178
2 179
2 180
1 181
1 181
1 181
1 181
1 181

5

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

JUMP / SEMANTIC CALL
JUMP - ABSOLUTE RELATIVE TO BASE. 1 - 32000
CALL -1 - -32000
CALL -1 - -32000
T1,T2=1;
K=0;
TR20: K=K+1;
IF K>#NTERM THEN
GO TO TR40;
TRIP(T2,1)=1024+K;
/* MATCH & POSITION. */
TRIP(T2,3)=T2+1;
/* FAIL, JUMP TO NEXT */
T2=T2+1;
/* SUCCESS ACTIONS GENERATED. */
GO TO TR20;
TR40: K=499;
TR60: K=K+1;
IF K>#NTERM+499 THEN
GO TO TR80;
TRIP(T2,1)=1024+K;
TRIP(T2,3)=T2+1;
T2=T2+1;
GO TO TR60;
TR80: T=T2;
/* COLLECT PRODUCTIONS ACCORDING TO THE
SYMBOL AT THE TOP OF THE PARSE STACK. */
CALL TITLE;
PUT SKIP(5);
PUT EDIT('REDUCTIONS TABLE',(' ',)DO I=1 TO 16)('A,SKIP(0),(16)
A(1));
DO I=1 TO #NTERM,500 TO(499+#NTERM);
IF I<500 THEN
PUT SKIP EDIT(INTERM(I),(' ',)DO J=1 TO(LENGTH(INTERM(I))))
)('SKIP,A,SKIP(0),(50)A(1));
IF I>=500 THEN
PUT SKIP(2)EDIT(INTERM(I),(' ',)DO J=1 TO(LENGTH(INTERM(I))))
)('A,SKIP(0),(50)A(1));
KNTR=0;
ORDER=0;
DO J=1 TO #PRD;
DO K=2 TO(PRD(J,0));
IF-(I=PRD(J,K))THEN
GO TO LOOPC;
KNTR=KNTR+1;
/* NEXT ROW OF ORDER */
IF KNTR>100 THEN
DO;
PUT EDIT('REDUCTIONS TABLE OVERFLOW')(SKIP(10),A
);
STOP;
END;
ORDER(KNTR,*)=PRD(J,K);
/* COPY PRODUCTION. */
ORDER(KNTR,-1)=K;
/* POSITION OF TOP PARSE SYM. */
ORDER(KNTR,-2)=UPROD(J);

```

1 181
1 181
1 181
1 181
1 181
1 182
1 183
2 184
2 185
1 186
1 187
1 187
1 188
1 188
1 189
1 189
1 190
1 191
2 192
2 193
1 194
1 195
1 196
1 197
1 198
1 199
1 199
1 199
1 200
1 201
1 201
1 202
2 203
3 204
3 205
3 205
3 206
3 207
3 207
2 208
2 209
3 210
4 211
5 212
5 213
4 214
5 215
5 215
6 216
6 217
6 217
6 218
6 219
4 220
4 221
4 221
4 222
4 222

6

KXSU'S PL/I NEATENER AND PRECOMPILER

PAGE

[illegible]

7

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

      PUT EDIT(OUTSTR)(COL(10),A);
      END;
END;
PROCEDURE;
  DECLARE (I,J,K,L,M,N) BINARY FIXED (31,0);
  /* COMING IN:
  F ENTIRE PRODUCTIONS' RIGHT HAND SIDE.
  WITHIN THE RIGHT SIDE OF PRODUCTION CONSIDERED TO BE
  THE TOP OF THE PARSE STACK.
  CN #
  :
  DUCTION LENGTH (*100 IF LRI)
  UCE INDICATOR.
  TES READ, X INDICATES REDUCE TO X
  ON #
  THIS IS A DERIVED REDUCTION.
  */
  /* SHIFT ALL RIGHT HAND SIDES RIGHT. */
  ABBREV=0;
  KNTR2=KNTR+1;
  DO I=1 TO KNTR;
    LV=ORDER(I,1);
    ORDER(I,1)=0;
    TPS=ORDER(I,-1);
    DO J=(TPS+2) TO 20;
      /* DELETE ALL ELEMENTS OF RHS BEYOND
      RIGHT CONTEXT CHARACTER. */
      ORDER(I,J)=0;
    END;
    DO J=20 TO 1 BY-1 WHILE(TPS=0);
      ORDER(I,J)=ORDER(I,TPS+1);
      ORDER(I,TPS+1)=0;
      TPS=TPS-1;
    END;
    IF ORDER(I,20)=0 THEN
      ORDER(I,-1)=0;
    ELSE
      ORDER(I,-1)=LV;
    J=0;
    DO K=1 TO 20;
      IF ORDER(I,K)=0 THEN
        J=J+1;
      END;
      IF ORDER(I,20)=0 THEN
        J=J+100;
      ORDER(I,0)=J;
    END;
  /* IF WHOLE BLOCK RIGHT CONTEXT, THEN ABBREVIATE */
  DO I=1 TO KNTR;
    IF ORDER(I,20)=0 THEN
      GO TO RE12;
    END;
  /* TRIP( ,2)=0 IMPLIES GO TO ENTER */
  T1=T1+1;
  /* LEAVE JUMP KEY EMPTY */

```

```

0 LENGTH 0
-1 POSITION 2 275
-2 PRODUCTI 2 275
-3 0 2 275
COMING OUT 2 275
0 CODED RE 2 275
-1 READ/RED 2 275
0 INDICA 2 275
-2 PRODUCTI 2 275
-3 FLAG- IF 2 275
2 275
2 275
2 275
2 276
2 276
3 277
3 278
3 279
3 280
4 281
4 282
4 282
4 282
4 283
4 284
4 285
4 286
4 287
4 288
4 289
4 290
4 291
4 291
3 292
4 293
5 294
5 295
4 296
4 297
4 298
3 299
3 300
3 301
3 301
4 302
4 303
3 304
2 305
2 305
2 306

```


KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

K=0;
DO J=1 TO 20;
  IF ORDER(I,J)=0 THEN
    K=K+1;
  END;
  IF ORDER(I,20)=0 THEN
    K=K+100;
  ORDER(I,0)=K;
END;
PROCEDURE;
/* HAVE GENERATED ORDER ARRAY FOR SOME SYNTACTIC VALUE.
   INSERT TRIPLES IN TRIP AND START ADDRESS IN JUMP KEY. */
/* JUMP KEY TO THIS BLOCK. */
T1=1;
/* NEXT JUMP KEY. */
DO I=1 TO KNTR;
  FST=1;
  TFTP=ORDER(I,19);
  TFTP=ORDER(I,19);
  /* TOP OF PARSE STACK SYMBOL. */
  DO J=1 TO 20;
    TF=ORDER(I,J);
    IF TF=0 THEN
      DO;
        IF FST THEN
          DO;
            /* SET FAILURES OF LAST REDUCTION HERE. */
            IF TRIP(K,3)=0 THEN
              TRIP(K,3)=T2;
            END;
            FST=FST;
            T=T2;
            END;
          K=20-J;
          /* CALCULATE POSITION WITHIN REDUCTION. */
          IF K=0 THEN
            K=-1;
          /* -1 IF RIGHT CONTEXT CHARACTER. */
          IF K>0 THEN
            TRIP(T2,1)=1024*K+TF;
          IF K<0 THEN
            TRIP(T2,1)=1024*K-TF;
          /* MAY BE NONTERMINAL. */
          /* TRIP(T2,2)=T2+1;
          /* SUCCESS JUMP TO NEXT. */
          /* DO K=1 TO #SEM;
          /* SEARCH FOR OVER-RIDE. */
          IF SEMAN(K,1)=TFTP THEN
            DO;
              /* TOP OF STACK SYMBOL.
              IFUPROD(ORDER(I,-2))-SEMAN(K,2)) THEN
                DO;
                  /* PROD OK.
                  IF SEMAN(K,3)=0 THEN
                    /* SUCCESS OVER-RIDE.
                    TRIP(T2,2)=SEMAN(K,4);
                    IF SEMAN(K,3)=1 THEN
                      /* FAIL OVER-RIDE
                      TRIP(T2,3)=SEMAN(K,4);
                      SEMAN(K,*)=0;

```

3 352
4 353
5 354
5 355
4 356
4 357
4 358
3 359
3 360
3 361
3 362
3 362
3 362
3 363
3 363
4 364
4 364
4 365
4 366
5 367
5 367
5 368
6 369
7 370
8 371
9 372
10 373
10 373
11 374
11 375
10 376
9 377
9 378
9 379
7 380
8 381
8 381
8 382
8 383
8 384
8 385
8 386
7 387
8 388
9 389
10 390
11 391
11 391
12 392
13 393
13 393
13 394
13 394
13 395
13 396
13 396
12 397

10

KSU'S PL/1 NEATENER AND PRECOMPILER

PAGE

```

/* MAY ONLY USE ONCE */
END;

END;
T2=T2+1;
END;

END;
/* SET UP AUTOMATIC REDUCTION */
/* TRIP( ,1)=0
/* TRIP( ,2)= # ITEMS TO POP PARSE STACK
/* TRIP( ,3)= # NEW SYNTACTIC DESCRIPTOR */
/* IF RIGHT CONTEXT CHECK IS SUCCESSFUL, THEN GO TO ENDTER */
IF ORDER(1,20)=0 THEN
DO;
TRIP(T2-1,2)=0;
/* 0 IMPLIES ENTER OVER-RIDES? */
END;
IF ORDER(1,20)=0 THEN
DO;
/* NO RIGHT CONTEXT THEN REDUCE */
DO K=1 TO #SEM;
/* SEARCH FOR SEMANTIC ROUTINE */
/* FIND REDUCTION # WITH NO TPS GIVEN. */
IF (SEMAN(K,2)=ORDER(1,-2)&SEMAN(K,1)=0) THEN
GO TO TAB15;
END;
/* IF NO ROUTINE IS GIVEN, THEN DEFAULT TO USER ASSIGNED # */
TRIP(T2-1,2)=ORDER(1,-2);
/* PROD # */
GO TO TAB17;
/* SET LAST SUCCESS */
/* LAST FAIL STAYS UNSET */
TRIP(T2,1)=0;
/* SET UP REDUCTION INFORMATION FOR SYNTACTIC */
/* REDUCER. */
TRIP(T2,2)=ORDER(1,0);
/* # OF STACK ITEMS TO REDUCE. */
/* TARGET SYMBOL. */
TRIP(T2,3)=ORDER(1,-1);
END;
TAB16:
END;
/* SFT ALL UNDEFINED ACTIONS IN THIS LAST REDUCTION BLOCK TO H
THEN FIND OVER-RIDE. */
DO K=1 TO T2-1;
IF TRIP(K,3)=0 THEN
TRIP(K,3)=T2;
END;
DO K=1 TO #SEM;
IF (SEMAN(K,1)=TFPS&SEMAN(K,2)=0) THEN
DO;
TRIP(T2,2),TRIP(T2,3)=SEMAN(K,4);
SEMAN(K,*)=0;
T2=T2+1;
GO TO TAB20;
END;
END;
/* IF NOT FOUND, DEFAULT = $999 */

```

12 398
12 398
10 399
8 400
7 401
7 402
5 403
4 404
4 404
4 404
5 404
5 404
6 405
6 406
6 407
6 407
5 408
6 409
7 410
7 410
8 411
8 411
8 411
8 411
8 412
7 413
6 414
6 414
6 415
6 416
6 417
6 417
6 417
6 418
6 418
6 419
6 419
6 420
6 421
4 422
4 423
4 423
4 423
5 424
5 425
4 426
4 427
5 428
5 428
6 429
6 430
6 431
6 432
6 433
6 434
4 435
3 436

11

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

TRIP(T2,2),TRIP(T2,3)=-999;
T2=T2+1;
RETURN;
END TABLE;
END REORDER;
/* PRINT SEMANTIC CALLS */
CALL TITLE;
PUT SKIP(5);
PUT EDIT('SUPPLIED SEMANTIC CALLS.')(A);
PUT EDIT('TOP PARSE STACK','PROD #','SUCC/FAIL','JUMP/SEMAN')||
TIC #||(COL(10),A,COL(35),A,COL(45),A,COL(60),A);
DO I=1 TO #SEM;
PUT SKIP(2);
PUT EDIT(SEMAN(I,2),SEMAN(I,3),SEMAN(I,4))(COL(35),F(4),COL(
49),F(1),COL(65),F(5));
IF SEMAN(I,1)<500 THEN
PUT EDIT(INTERM(SEMAN(I,1)))(SKIP(0),COL(10),A);
ELSE
PUT EDIT(INTERM(SEMAN(I,1)))(SKIP(0),COL(10),A);
PUT EDIT(INTERM(SEMAN(I,1)))(SKIP(0),COL(10),A);
PUT EDIT(INTERM(SEMAN(I,1)))(SKIP(0),F(3),A(2),F(3),A);
END;
IF #SEM=0 THEN
PUT EDIT('NONE')(SKIP(10),A);
/* PRINT REDUCTIONS TABLE */
CALL TITLE;
PUT SKIP(5);
PUT EDIT('COMPLETE REDUCTIONS')(COL(30),A);
PUT EDIT('POSITION & MATCH','SUCCESS','FAILURE')(SKIP,COL(10),A
,COL(35),A,COL(45),A);
DO I=1 TO T2-1;
PUT SKIP;
SW=0;
J=TRIP(I,1);
IF J<0 THEN
DO;
KK=-(J/1024);
SW=1;
J=-1024+J;
/* KK= POSITION */
/* J = MATCH */
END;
ELSE
DO;
KK=(J/1024);
J=J-KK*1024;
END;
IF SW THEN
PUT EDIT('R',KK)(SKIP(0),A,F(2));
ELSE
PUT EDIT('L',KK)(SKIP(0),A,F(2));
IF KK=0 THEN
DO;
PUT EDIT('<ANY SYMBOL>','TRIP(I,2),TRIP(I,3))(COL(10),A
,COL(35),F(6),COL(45),F(6));
IF TRIP(I,2)=-999 THEN
PUT EDIT('***** BLOCK END *****')(SKIP,COL(20),A);
GO TO L4;
END;

```

PAGE

11

12

KSU'S PL/I NEATENER AND PRECOMPILER

PAGE

```

IF J<500 THEN
  PUT EDIT(INTERM(J),TRIP(1,2),TRIP(1,3))(COL(10),A,COL(35),
  F(6),COL(45),F(6));
ELSE
  PUT EDIT(INTERM(J),TRIP(1,2),TRIP(1,3))(COL(10),A,COL(35),F
  (6),COL(45),F(6));
  PUT EDIT(' ',1)(X(5),A,F(4));
END;
L4: FINIS: PUT EDIT('FINIS')(PAGE,A);
      PUT FILE(PUNCH)EDIT('///LISTDECK JOB (DP73RG76),''CRANK''')(COL(
      1),A);
      DO I=500 TO(499+INTERM);
/* KEYWORDS OUT */
      PUT FILE(PUNCH)EDIT('.....|TERM(1)|.....|')(COL(1),A,X(5),F(7
      ,0));
      END;
/* LHS */ PUT FILE(PUNCH)EDIT(INTERM)(COL(1),F(5,0));
      DO I=1 TO INTERM;
      J=LHSP(I);
      PUT FILE(PUNCH)EDIT(I)(COL(2),F(4,0));
      PUT FILE(PUNCH)EDIT(LHS(J))(F(5,0));
      IF LHS(J)--1 THEN
        GO TO F2;
      J=J+1;
      GO TO F1;
END;
F2: /* REDUCTIONS TABLE OUT */
      PUT FILE(PUNCH)EDIT((T2-1))(COL(2),F(4,0));
      PUT FILE(PUNCH)SKIP;
      DO I=1 TO(T2-1);
      PUT FILE(PUNCH)EDIT((TRIP(I,J)DO J=1 TO 3))(3 F(5,0));
      END;
      PUT EDIT(' REDUCTIONS TABLE SIZE = ',T2)(PAGE,A,F(13));
      END CRANK;

```

3 483

3 484

3 484

3 485

3 485

3 485

3 486

2 487

2 488

1 489

1 489

2 490

2 491

2 491

2 492

1 493

2 494

2 495

2 496

2 497

3 498

3 499

2 500

2 501

2 502

1 503

1 503

1 504

2 505

2 506

2 507

1 508

1 509

DESIGN AND IMPLEMENTATION OF A SIMPLE
TABLE DRIVEN COMPILER

by

Robert Douglas Crank

B.S. in Computer Science, Kansas State University, 1971

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1973

The PLAN compiler was implemented on the IBM 360/50 at Kansas State University. Although originally intended to be self-compiling, the current version is not strictly so. It does however compile a reasonable subset of PL/I, with procedure blocks, conditional statements, formatted I/O, and single and double precision integer arrays.

The syntax checker used is a deterministic table driven reductions analysis scheme which conceptually reduces a source code input stream according to a set of rules defining the language. No backtracking is necessary. A pushdown stack is maintained for the partially processed stream.

Nova assembler language mnemonics are generated by a number of semantic routines which receive control according to the top elements on the stack. A single source error terminates code generation but scanning continues. Generated code consists of system calls with appropriate arguments.