A STRUCTURED PROGRAMMING PREPROCESSOR FOR A VARIETY OF BASE LANGUAGES

by

Mary L. Love

B.S., Kansas State University, 1975

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

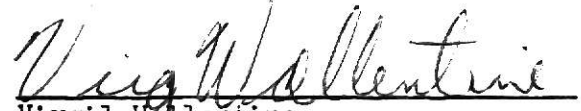
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1977

Approved by:



Virgil Wallentine

TABLE OF CONTENTS

I. Introduction

Project Definition 1

Motivation 2

Organization of Remainder of Paper 3

II. Techniques of Design and Implementation

Design 5

Implementation 22

III. User Information Overview

Input. 26

Output 29

Error Conditions 31

IV. Observations

Good and Bad Aspects of Project. 33

Overall Appraisal of Direction 34

Time, Effort, and Money Involved 35

Elements to Be Changed 35

Possible Further Developments. 36

Appendix

I. Detailed Implementation Information. 41

II. Detailed User Information. 82

III. Job Control Language Listings. 97

IV. Example Base Language Specifications 101

V. Preprocessor Source Code Listing 105

ILLEGIBLE DOCUMENT

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

THIS IS THE BEST COPY AVAILABLE

FIGURES

Figure II.1. Preprocessor and Language Translator Relationship . . 6

Figure II.2. Flow of Data in Preprocessor. 18

Figure II.3. Modularity of the Preprocessor. 20

TABLES

Table III.1. Prototype Parameters. 27

Table III.2. Definitions Required for Token Recognition and Code Generation. 28

Table A.I.1. Error Exits with Additional Information Provided. . . 5A

Table A.I.2. Scanning Routines 14A

Table A.I.3. Production Language for Parser. 31A

ACKNOWLEDGMENT

The author wishes to express her deep thanks to her major professor, Dr. Virgil Wallentine, and the other members of her committee, Drs. Linda Shapiro and William Hankley. A sincere thank you also goes to the other students in the Department of Computer Science who listened and gave suggestions and support which contributed toward the completion of this report.

SECTION I. INTRODUCTION

PROJECT DEFINITION

This report describes the design and implementation of a preprocessor which facilitates structured programming in a variety of base languages. The preprocessor is intended to allow and encourage the use of structured programming techniques. It attempts to achieve these goals by translating structured programming control structures, not currently implemented in a particular programming language, into semantically equivalent statements which are standard in the language. The preprocessor also includes a text-replacement macro facility which allows the user to provide additional control constructs.

The user provides three types of input to the preprocessor. The first type consists of information about the base language including prototypes of several simple types of statements, operators, and other syntactic information. The second type of input consists of macro definitions. The third type of input is a program written in an extended version of a programming language. The extended version of a language includes all the usual features of that language plus the structured programming control constructs made available by the preprocessor and any macros for which the user has provided definitions.

The output of the preprocessor is a program which can be translated by the usual compiler for the language which was being extended. This program is provided both in printed form and in a temporary data set which can be used as input to the language translator. The output program is logically equivalent to the input program which was written in the extended language.

MOTIVATION

The first and foremost incentive behind this project is to make the techniques and desirable features of structured programming [Dah72,Lec74] available to programmers despite the programming language they may use. While there currently exist a number of processors designed to provide structured programming control structures for only FORTRAN [ONe74,Hig75], the author's preprocessor may be used with a variety of programming languages. The current implementation works with the PL/I, FORTRAN, and SNOBOL programming languages.

There are several reasons for using a preprocessor to accomplish this task. First, a preprocessor can be built to accomodate multiple languages. Second, the language translator currently being used may be retained because the preprocessor outputs code equivalent to the code written in the extended language but which is standard in the base language. Since the user need not acquire a new translator, programmers do not have to learn an entirely new language, just simple modifications to the current one. Furthermore, old programs do not have to be recoded in a new language. Finally, a preprocessor such as described here is easy to use in conjunction with the existing language translator because the preprocessor's output can be input directly into the existing translator.

A new preprocessor was implemented instead of using an existing general-purpose macro processor [Bro67,Bro69] because the new preprocessor is intended to encourage the use of structured programming as well as make it available. To do this, eight control structures are implemented in the preprocessor, thus relieving the programming of the task of having to define them for use with a general-purpose macro processor. This

definition of control structures would be necessary once for each language being used, whereas they are already defined for all languages in the new preprocessor.

Since the author recognizes the fact that no one person can think of all the structured programming control structures which may possibly be desirable for use at one time or another, a text-replacement macro facility [Sol74] has also been incorporated into the preprocessor. This facility allows the user to define macros which will aid in the programming function. Inclusion of the macro facility introduces the possibility that the preprocessor's goals of providing structured programming capabilities and encouraging their use may be undermined by a user who defines macros which conflict with structured programming principles. Because of this possibility a set of guidelines for macro definitions is provided in Appendix II. If the user follows these guidelines in writing macro definitions, the likelihood of violating structured programming principles in user-defined macros will be decreased.

The preprocessor is written in SNOBOL4 and run under the SPITBOL compiler [Dew71]. This language was chosen because the pattern-matching capability provided is well-suited to token recognition, a major part of this application.

ORGANIZATION OF REMAINDER OF PAPER

The remainder of this paper is organized into three sections. The first section describes the techniques of design and implementation of the preprocessor. The second section presents an overview of information required to use the preprocessor. The third section is a summary of the project. The Appendix of the paper includes details

of both the implementation and use of the preprocessor, the Job Control Language statements required to run the preprocessor, example base language specifications, and a listing of the preprocessor source code.

SECTION II. TECHNIQUES OF DESIGN AND IMPLEMENTATION

DESIGN

Design of Processor

The project was intended to effect a mapping from an extended language to the corresponding base language. The extended version of a programming language consists of all the features normally found in the language plus the structured programming control constructs made available by the processor and macros for which the user has provided definitions. To effect this mapping for a variety of languages, a generalized tool, that is, one capable of processing different languages, is required. Since a preprocessor could be designed to accept multiple languages for processing, the first design decision was to use a preprocessor.

There were other factors supporting this decision. The preprocessor could output code which is standard in the base language. As a result, the user need not acquire a new translator and programmers need to learn only simple modifications to the language they currently use. Programs already in use would not have to be recoded in a new language. Finally, since the preprocessor's output could be input directly into the language translator, it would be easy to use with the existing translator. Figure II.1 illustrates the relationship between the preprocessor and the language translator.

The decision to implement a new preprocessor rather than use an existing general-purpose macro processor stemmed from the fact that one of the goals of the project is to encourage the use of structured programming techniques. If an existing macro processor were used, the

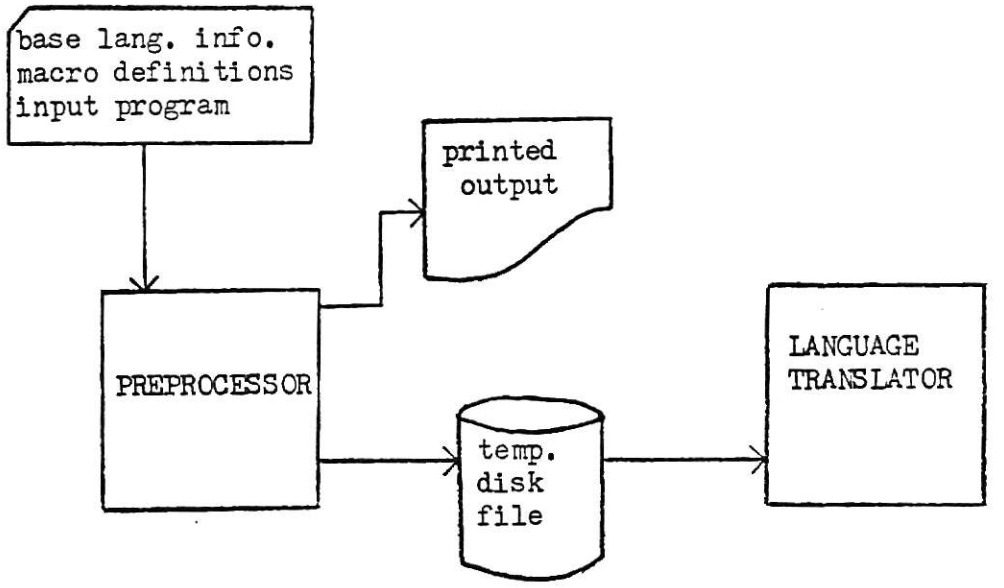


Figure II.1. Preprocessor and Language Translator Relationship.

user would have to define the extended constructs each time the macro processor was used. Furthermore, the extended constructs would have different definitions for each language. Selected structured programming control constructs could be implemented within a new preprocessor. Consequently, the need to redefine the constructs at each use and for each language would be eliminated.

Another design decision made to enable the project to be generalized was to use a table-driven structure parser. The parser was designed based on Production Language [Gri71]. Using a table-driven parser would enable the parser to recognize constructs in different languages because the tokens would be recognized by a scanner using the appropriate definitions.

Eight structured programming control structures were chosen for implementation in the preprocessor. Each of these eight was chosen

either because it receives support in the literature as being a structure necessary for structured programming [Kel,Lec74,Led75,Weg75] or because it is practical to use. The structures included are:

IF - THEN - ENDIF	IF - THEN - ELSE - ENDIF
WHILE - ENDWHILE	REPEAT UNTIL - ENDREPEAT
CASE OF - ENDCASE	DO (iterative) - ENDDO
DOGROUP - ENDDOGROUP	QUITLOOP

The decision to include a text-replacement type macro facility was a difficult one to make. On the one hand it seemed obvious that no one person could think of all the structured programming control structures which may possibly be desirable for use at one time or another. However, it also appeared that to include such a facility would defeat the purpose of the project by allowing users to define any constructs they wished, whether or not those constructs conflicted with the principles of structured programming.

After much deliberation it was decided to include the macro facility. A set of guidelines is provided along with this facility for the user to follow in defining macros. The author feels that by following these guidelines, the user can write macro definitions which have little chance of violating structured programming principles.

A final consideration in the design of the preprocessor was keeping its development and testing time and effort to a minimum. A project of this size, written in a language which does not currently support many structured programming techniques, would certainly be unmanageable if written as one large program. Therefore, to keep the coding and testing as simple as possible, the preprocessor was designed

to be modular. The final version of the preprocessor contains fifty modules in addition to the driver routine.

Design of Execution

The preprocessor may perform one or two parses in processing the input program to produce the final output. If the user has provided any macro definitions, the preprocessor will make a second parse using the output of the first parse as its input. The second parse is required because users are allowed to use the extended constructs which are built into the preprocessor within macro definitions. This second parse will expand these occurrences of the extended constructs into standard code.

Although the main processing of the input program will be the same for all languages, there are some cases for specific languages which must be handled separately. For example, in SNOBOL a logical not operator requires post processing to make the output produced by the preprocessor legal in the standard language. In many other languages the logical not may be formed in a manner similar to an equal relation, that is, with an argument followed by the operator and a second argument, but utilizing a null first argument. In SNOBOL, however, post processing is required to eliminate the comma which appears illegally when a null first argument is used in a logical not relation. That is, the form ¬((,arg2)) must be replaced by ¬((arg2)). Special functions such as this are accomplished by doing post processing for each language which requires it. Currently, the post processing is written into the driver of the preprocessor. However, for generality

it could be provided as a call to an external routine. The appropriate routine would have to be linked with the preprocessor when it is loaded for execution.

Another function of the preprocessor is to insert declarations, if required, for any new variables it introduces. To allow declarations to be properly placed in a program, the user includes an indication of the appropriate location by inserting a special statement which will be replaced by declarations. If the declarations are not required, the special statement should be omitted and the prototype for a declaration statement should be blank.

Design of Extended Constructs

The code which is generated for each of the eight extended constructs is explained briefly in the following paragraphs. Along with each explanation is a simple example designed to aid the reader in visualizing the replacement strategy while eliminating the details of any particular language and the preprocessor's exact output.

IF - THEN - ENDIF: If the negation of the Boolean expression provided in the IF statement is true, a branch is taken to a unique label which will be placed on a null statement at the position of the ENDIF statement. The code following the THEN is inserted following the test of the negation.

IF bool THEN assign	if not bool then go to $unilab_i$
	assign
ENDIF	$unilab_i$: null

IF - THEN - ELSE - ENDIF: If the negation of the Boolean expression provided in the IF statement is true, a branch is taken to a unique

label which will be placed on a null statement at the position of the ELSE. The code following the THEN is inserted following the test of the negation. An unconditional branch to a second unique label ($unilab_{i+1}$) is added at the end of this code. The second unique label will be placed on a null statement at the position of the ENDIF. The code following the ELSE is inserted after the unconditional branch to the second unique label.

IF bool THEN assign	if not bool then go to $unilab_i$ assign go to $unilab_{i+1}$
ELSE branch	$unilab_i$: null branch
ENDIF	$unilabel_{i+1}$: null

DOGROUP - ENDDOGROUP: No replacement code is generated for this construct. The code necessary to delineate the code between the DOGROUP and ENDDOGROUP statements as a separate entity is included within the replacement code for the other constructs.

DOGROUP	
assign	assign
call	call
ENDDOGROUP	

DO (iterative) - ENDDO: The four elements extracted from the DO statement are the index variable (INDEX), the beginning value (BEGIN), the ending value (END), and the increment value (INCREMENT). Three statements are inserted at the position of the DO statement. The first statement sets a unique variable ($univar_i$) equal to END minus INCREMENT. The second sets INDEX equal to BEGIN less INCREMENT. The third statement has a unique label ($unilab_i$) and assigns INDEX plus INCREMENT to INDEX.

A set of five statements is inserted at the position of the ENDDO statement. The first tests if the incrment value is positive and branches to a unique label ($unilab_{i+1}$) if that is true. The next statement branches to $unilab_i$ if INDEX is greater than or equal to $univar_i$. The third statement is an unconditional branch to another unique label ($unilab_{i+2}$). The fourth statement has $unilab_{i+1}$ as the label and branches to $unilab_i$ if INDEX is less than or equal to $univar_i$. The fifth statement has $unilab_{i+2}$ as the label and assigns INDEX the value of INDEX plus INCREMENT. This rather complex set of statements at the ENDDO position is required because the increment value may be either positive or negative, and the condition for ending the loop differs according to the sign of the increment value.

DO index = begin TO end BY incr	$univar_i$ ← end - incr
	index ← begin - incr
	$unilab_i$: index ← index + incr
assign	assign
call	call
ENDDO	if incr gt 0 then go to $unilab_{i+1}$
	if index ge $univar_i$ then go to $unilab_i$
	go to $unilab_{i+2}$
	$unilab_{i+1}$: if index le $univar_i$ then go to $unilab_i$
	$unilab_{i+2}$: index ← index + incr

WHILE - ENDWHILE: A unique label is placed on a statement which branches to a second unique label if the negation of the Boolean expression provided in the WHILE statement is true. An unconditional branch to the first unique label and a null statement with the second unique label are placed at the position of the ENDWHILE. A WHILE - ENDWHILE will not

be executed if the condition is not satisfied when the loop is encountered.

WHILE bool	$unilab_i$: <u>if not</u> bool <u>then go to</u> $unilab_{i+1}$
assign	assign
ENDWHILE	<u>go to</u> $unilab_i$
	$unilab_{i+1}$: <u>null</u>

REPEAT UNTIL - ENDREPEAT: A null statement with a unique label is placed at the position of the REPEAT UNTIL statement. A statement is inserted at the position of the ENDREPEAT which branches to that unique label if the negation of the Boolean expression specified in the REPEAT UNTIL statement is true. A REPEAT UNTIL - ENDREPEAT will always be executed at least once.

REPEAT UNTIL bool	$unilab_i$: <u>null</u>
assign	assign
ENDREPEAT	<u>if not</u> bool <u>then go to</u> $unilab_i$

CASE OF - ENDCASE: A conditional branching statement is inserted at the position of the CASE OF statement which branches to a unique label if the index of the CASE statement is not equal to a test value which is initialized to one. Two statements are inserted after each statement or statement group within the CASE which is treated as a separate entity. The first is an unconditional branch to a unique label which will be placed on the second null statement at the position of the ENDCASE. The second statement has the unique label which was the destination of the last conditional branch and branches to another unique label if the index of the CASE is not equal to the test value which has been incremented by one. These two statements are repeated until all cases for this CASE structure have been processed. At the ENDCASE a null statement with the unique label

which was the destination in the last conditional branch is placed before the null statement described above.

CASE OF index	if index ne 1 then go to $unilab_{i+1}$
$case_1$	$case_1$
	go to $unilab_i$
	$unilab_{i+1}$: if index ne 2 then
	go to $unilab_{i+2}$
$case_2$	$case_2$
	go to $unilab_i$
	$unilab_{i+2}$: if index ne 3 then
	go to $unilab_{i+3}$
$case_3$	$case_3$
ENDCASE	$unilab_{i+3}$: null
	$unilab_i$: null

QUITLOOP: A QUITLOOP is replaced by an unconditional branch to a unique label which is placed on a null statement after the inserted code for the enclosing loop.

WHILE -----	while replacement code
$statement_1$	$statement_1$
QUITLOOP	go to $unilab_i$
$statement_2$	$statement_2$
ENDWHILE	endwhile replacement code
	$unilab_i$: null

Design of Input

The following is a brief description of the input to the preprocessor. A more detailed description of the input may be found in Appendix II.

There are three types of input to the preprocessor. The first type consists of information about the base language. This information is provided by the user in a form derived from Backus-Naur Form [Gri71]. These definitions are used by the preprocessor in two ways. First, the preprocessor uses part of this information to recognize tokens in the

input program. Second, the preprocessor outputs statements standard in the base language by utilizing this information. Some of the information is used in only one manner. For example, the definition for end-of-statement is used only to recognize the end of a statement in the input program, and the statement prototypes are used only for producing output standard in the base language. On the other hand, some information is used in both ways. An example is the definition for a comment. The preprocessor uses this definition to recognize comments in the input program so they will not be analyzed as statements and also to produce comments in the output program which allow the user to locate where extended statements have been replaced.

The information required for each definition was determined and a definition form was then derived. In some cases, statement prototypes being an example, the definition simply reflects the proper form of the statement using parameter names where actual values would be placed. In other cases column numbers, lengths, and character strings are appropriate. It is also necessary to specify in some cases whether columns or character strings or both are utilized in the recognition of syntactic tokens. The comment definition is again a good example. It may contain column numbers or character strings or both. This example of a comment definition for PL/I contains an opening character string, an opening column number, and a closing character string.

EXAMPLE: Comment definition for PL/I

⟨COMMENT⟩::='/*' + 10 '*/'

The definition of unique labels contains lengths and character strings required for generation while the definition of language labels contains

character strings and column numbers needed for recognition. The following example of a unique label definition for SNOBOL defines labels of the form 'LABxxxxxxNEW' where xxxxxx is a sequence number. The example of FORTRAN language labels contains column numbers for positioning labels, a character set for the first character of a label, and a character set for the remaining characters of a label.

EXAMPLE: Unique label definition for SNOBOL

⟨UNIQUE LABEL⟩::=LAB 6 NEW

EXAMPLE: Language label definition for FORTRAN

⟨LANGLABEL⟩::=1-5 1-5 '0123456789' '0123456789'

Another piece of information relating to the base language which may be provided by the user concerns whether or not to recognize and expand extended constructs. If a base language has a construct which is recognized by keywords identical to those used by the preprocessor, the user must either specify that the extended construct keywords are to be overridden or not use the feature as provided in the base language. An override card to accomplish this task for the keywords DOGROUP and ENDDOGROUP would appear as follows:

OVERRIDE:DOGROUP,ENDDOGROUP

The second type of input consists of macro definitions. This information is optional. If the user wishes to introduce no macros, this is indicated by omission of any macro definition statements. The user provides a macro name, parameter names, and replacement text in each macro definition. The preprocessor will replace each macro call encountered with the corresponding replacement text, replacing parameter names with the actual arguments of the call.

The third type of input is the program written in the extended version of the programming language for which definitions have been provided. A special statement is included in the program to indicate the proper position for declarations of new variables.

Of some interest is the manner in which the definitions which have been provided are stored in the preprocessor. Each definition has an array associated with it. Although the specific contents of the array elements depend upon the particular definition, the concepts behind the storage of all the definitions are similar. Each array element contains a single piece of information which has been extracted from the definition. In the case of statement prototypes, the first element contains the number of statement components and each of the remaining elements contains one component, either text or a parameter name. For definitions of other syntactic elements, the definition array elements may contain information such as column numbers, character strings, or length requirements provided in the definitions.

The array for the FORTRAN language label definition above would contains the values 2,'COL',2,1,5,1,5,'0123456789', and '0123456789'. The first item indicates a series of column numbers is provided for the label starting position. The next two items indicate a series of column numbers is provided for the label ending position. The next four values specify the two series of columns. The two remaining character strings are the sets of valid characters for the first character of a label and the remaining characters of a label, respectively.

Design of Output

The output produced by the preprocessor is a program logically equivalent to the input program but which is standard in the base language. This output program is produced in two forms, a printed listing and a temporary disk file. The disk file can be input directly into the translator for the base language.

The output program contains at least one comment for each extended construct which is replaced by code produced by the preprocessor. There are also comments placed to indicate code inserted by the preprocessor which is not directly related to the use of a particular extended construct but of which the user should be aware. These comments indicating where new code has been produced are provided to aid the user in determining where errors have been made which relate to the error messages produced by the language translator and in correcting logic errors resulting from incorrect usage of the extended constructs.

The preprocessor's output does not go directly into the print file and the temporary disk file. There are intermediate files which hold these outputs. Figure II.2 illustrates the flow of data in the preprocessor. First the outputs are placed in intermediate files before it is determined whether a second parse will be required. If the second parse is not needed, the files are copied into the next set of intermediate files. If the second parse is required, the output of the second parse is placed in this set of intermediate files. The preprocessor then does any post processing necessary for the language being processed and places the outputs in the third set of intermediate

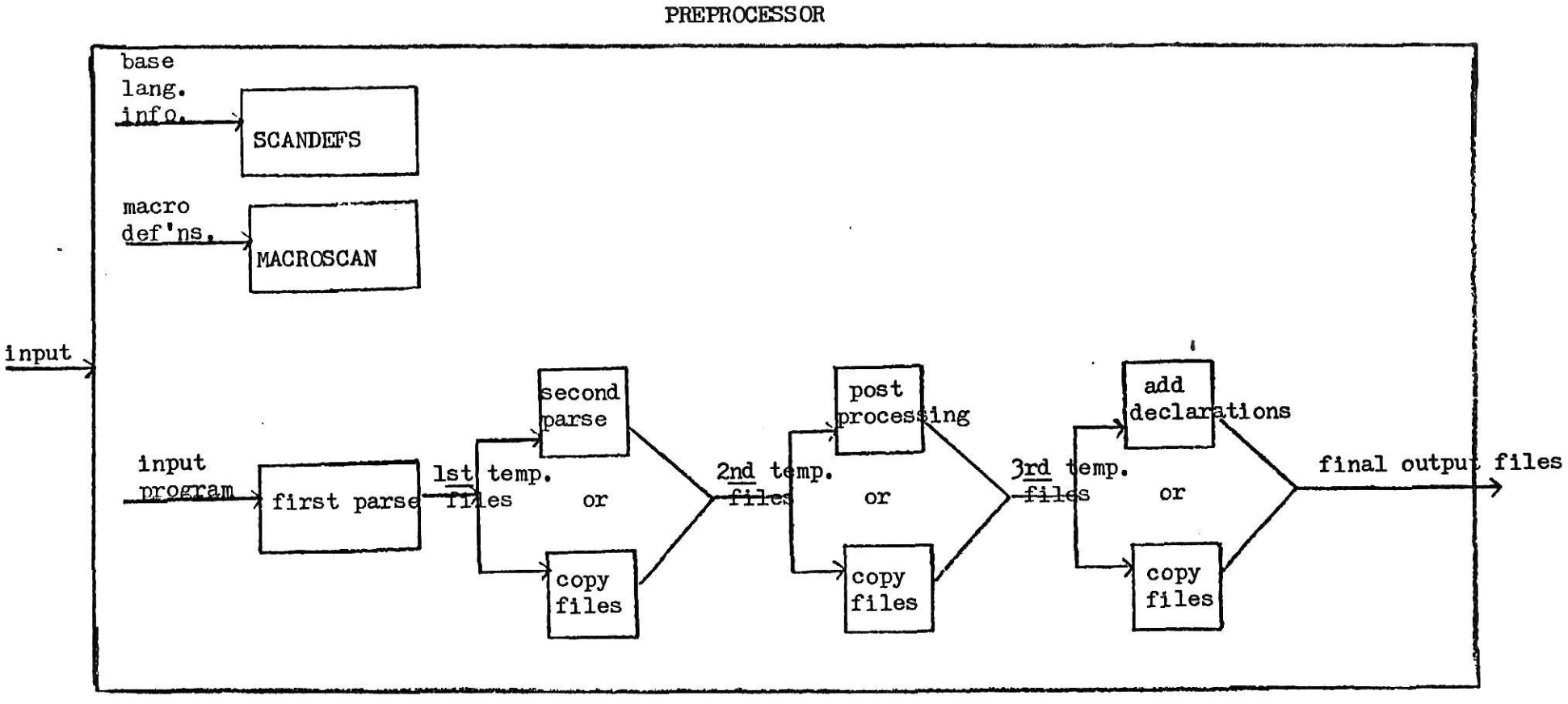


Figure II.2. Flow of Data in Preprocessor.

files. Finally, any required declarations of new variables are added and the outputs are placed in the final output files.

If an error condition occurs at any time, the preprocessor outputs an error message and prints the contents of whichever intermediate punch file contains the results of processing up to the error. This is done so the user can determine how far the preprocessor had progressed and what output had been produced thus far.

Design of Specific Modules

The following is a brief discussion of some of the modules of the implementation. Appendix I contains a detailed description of the final implementation.

Each definition provided by the user has an associated scanning routine which decodes the definition into its components and places them in the appropriate elements of the associated definition array. The scanning of all the definitions except those for macros is controlled by one routine which determines the scanning routine which should be called to process each definition.

The six statement prototypes are used to control the creation of output statements standard in the base language. Each of the prototypes has a statement creation routine with which it is associated which creates one type of statement and is controlled by the prototype.

There are three different insertion routines. One inserts a string with appropriate syntax to be a comment. A second routine inserts a string as a statement, skipping the columns specified in the ignored columns definition. The third insertion routine inserts a string which is already in the proper form for a statement.

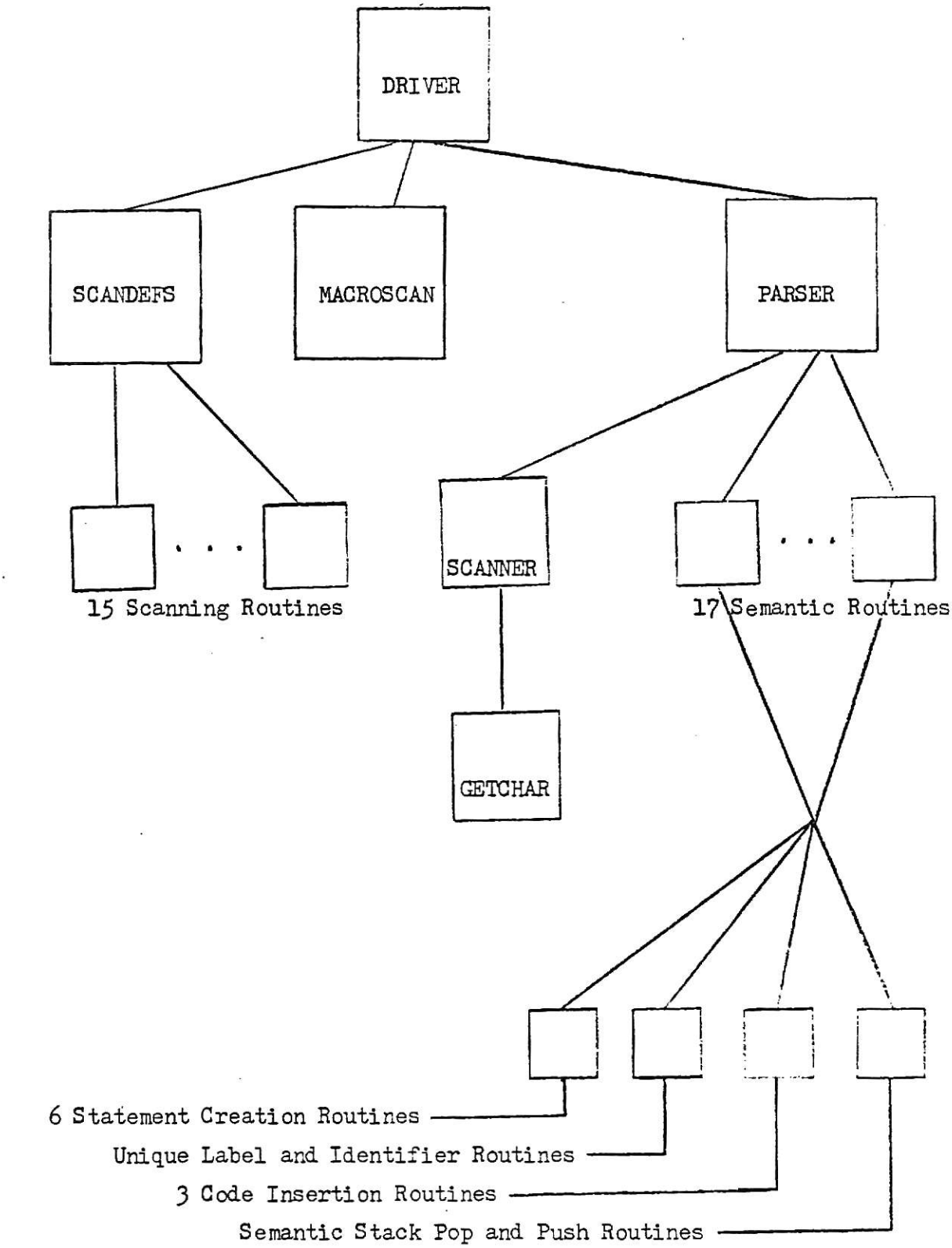


Figure II.3. Modularity of the Preprocessor.

The routine which scans macro definitions is similar in concept to the other definition scanning routines in that it decodes a definition and stores its components in an array. However, the macro scanning routine creates two arrays, one for the parameters in the calling sequence and one for the components of the replacement text. These arrays are placed in elements of a SNOBOL table and are accessed via the macro name.

The structure parser is table-driven and written in a form based on Production Language. It has one syntactic stack which is represented by a character string. It was necessary to order the tests made against this stack such that the terminating symbol of a construct, for example ENDDO, at the stack top would not be mistakenly recognized as the opening symbol, in this case DO. The parser calls a scanning routine to place new tokens at the top of the syntactic stack and seventeen semantic routines to perform code generation and replacement.

The scanner uses much of the information provided by the user about the base language. Literals and comments are recognized as such and, as a result, their contents are not scanned for extended constructs. Another token is placed on the top of the syntactic stack each time the scanner is called. The following example represents the contents of the syntactic stack following the recognition of a DOGROUP statement, an assignment statement, and a REPEAT UNTIL statement. The tokens have been examined and reduced by the parser. The DOGROUP statement is represented by the token DOGROUPSTMT. The assignment statement has been reduced to a structure list (STRLIST) token. The REPEAT UNTIL statement is represented by the REPEATSTMT token. Note that the tokens are in a character string and separated by single blanks.

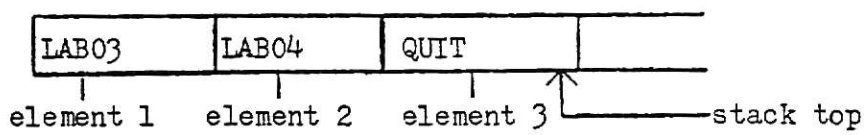
EXAMPLE: Contents of syntactic stack

DOGROUPSTMT STRLIST REPEATSTMT

There are seventeen semantic routines in the preprocessor. The basic strategy of the semantic routines is to make the extended statement into a comment and insert appropriate code which is standard in the base language. One semantic routine copies statements containing no extended constructs or macro calls directly into the output or replaces macro calls with the appropriate replacement text. Several other semantic routines, for example the one which processes the token DOGROUP, simply make extended statements into comments and produce no new code.

The semantic routines use a semantic stack represented by an array to store items to be transmitted between semantic routines. The stack may contain such items as labels, values, variables, and indications of the occurrence of the QUITLOOP construct. The following example of the semantic stack shows two labels and one indicator of the occurrence of a QUITLOOP structure.

EXAMPLE: Contents of the semantic stack.



IMPLEMENTATION

Functional Classification of Modules

There are five functional categories of modules in the preprocessor. The categories are:

1) definition scanners

2) statement, label, and variable creators

3) semantic routines

4) service routines

5) special purpose modules

There are three types of definition scanners. The first type of definition scanning routine consists of the six routines which decode statement prototypes. The only routine of the second type is the routine which scans macro definitions. The third type consists of the routines which decode the remainder of the definitions.

There are six statement creation routines, one label creation routine, and one variable creation routine in the creator category. Each of the statement creator routines creates one type of output statement. The label and variable creators provide unique labels and identifiers, respectively. The semantic routine category contains the seventeen semantic routines. These are the routines which generate output code sequences.

There are six service routines which perform recurring tasks. Among these are two semantic stack maniuplation routines, pop.sem and push.sem, and three string insertion routines. The other routine in this category is the one which retrieves single characters from the input text.

The special purpose modules include the preprocessor driver, the structure parser, the lexical scanner, and the routine which controls definition scanning.

Examples of Output

The following three examples illustrate the preprocessor's output for specific constructs. The first example illustrates the use of a CASE construct in PL/I. The second example is a REPEAT UNTIL structure in SNOBOL. The last example uses FORTRAN to illustrate the IF-THEN-ENDIF construct.

In each of these examples, the assumption is made that the preprocessor has not created any unique labels prior to the processing of the extended construct in the example. Also assumptions are made that the specification examples given for each language in Appendix IV were used by the preprocessor.

EXAMPLE -- PL/I LANGUAGE:

Partial Input Program:

```
CASE OF I;
   B = A + 6;
   B = A * 6;
   B = A - 6;
ENDCASE;
```

Corresponding Output Program:

```
        /*    CASE OF I;*/
L0003    : IF(I       ¬=   1       )THEN GO TO L0002    ;
      B = A + 6;
        /*NEXT TWO STATEMENTS INSERTED FOR CASE PROCESSING*/
L0005    : GO TO L0001    ;
L0002    : IF(I       ¬=   2       )THEN GO TO L0004    ;
      B = A * 6;
        /*NEXT TWO STATEMENTS INSERTED FOR CASE PROCESSING*/
L0007    : GO TO L0001    ;
L0004    : IF(I       ¬=   3       )THEN GO TO L0006    ;
      B = A - 6;
        /*NEXT TWO STATEMENTS INSERTED FOR CASE PROCESSING*/
L0009    : GO TO L0001    ;
L0006    : IF(I       ¬=   4       )THEN GO TO L0008    ;
        /*    ENDCASE;*/
L0008    : ;
L0001    : ;
```

EXAMPLE -- SNOBOL LANGUAGE:

Partial Input Program:

```
    REPEAT UNTIL (LT(I,3))
       I = I / 6
       J = J + I
    ENDREPEAT
```

Corresponding Output Program:

```
*    REPEAT UNTIL (LT(I,3))
L000001NEW
       I = I / 6
       J = J + I
*    ENDREPEAT
L000002NEW    ¬((LT(I,3))    )  :S(L000001NEW  )
```

EXAMPLE -- FORTRAN LANGUAGE:

Partial Input Program:

```
      IF (I.LE.6) THEN        X = X + I
      ENDIF
```

Corresponding Output Program:

```
C      IF (I.LE.6) THEN
10024 IF(          .NOT. (I.LE.6)     ) GO TO 10014
      X = X + I
C      ENDIF
10014 CONTINUE
```

SECTION III. USER INFORMATION OVERVIEW

This overview is not intended to be used as a user's guide for the preprocessor. Appendix II contains the detailed information required for its use.

INPUT

There are three types of input to the preprocessor. The first type of input consists of information about the base language in a form derived from Backus-Naur Form. This portion of the input is described in the following paragraphs.

There are six statement prototypes provided by the user. These are patterns for the preprocessor to follow in creating output statements. The prototypes are as follows:

unconditional branch	prototype for an unconditional branching statement
conditional branch	prototype for statement which tests a condition and branches if the condition is true
declaration	prototype for a statement which identifies a variable to the compiler; if no declarations are required, this prototype is blank
null	prototype for statement which serves no function but to mark a location with a label
assignment with plus	prototype for a statement which does a single addition operation and places the result in a variable
assignment with minus	prototype for a statement which does a single subtraction operation and places the result in a variable

Prototype	Parameters
unconditional branch	label, destination
conditional branch	label, value 1, relation, value 2, destination
declaration	variable name
null	label
assignment with plus	label, variable, value 1, value 2
assignment with minus	label, variable, value 1, value 2

Table III.1 Prototype Parameters.

Each prototype has specific parameter names in it. The parameters for the prototypes are shown in Table III.1.

Each of the parameter names will be replaced by an appropriate value during generation of output code by the preprocessor. All the parameter names for each prototype must be present except in the declaration statement prototype. Since this prototype may be entirely blank under certain conditions, a test is not made for the absence of the parameter name.

The definitions required by the preprocessor for token recognition and output code generation are given in Table III.2. The definition for each of these items is represented in a special form based on the types of information and how much information is required. For example, there are four different ways in which a continuation card may be indicated. The first method is by a special continuation symbol. Another method is to start in a certain column. The third way is to

Definition	Information Provided
end of statement	recognition criteria for end of statement
continuation	recognition criteria for continuation card
language label	recognition criteria for a label in the input program
literal	recognition criteria for a literal string
comment	recognition and generation criteria for a comment
ignored columns	specification of columns whose contents are ignored in recognition and generation of statements
unique label	generation criteria for unique labels
unique variable	generation criteria for unique identifiers
greater than, greater than or equal, less than or equal, not equal, logical not	symbol strings for these Boolean operators

Table III.2. Definitions Required for Token Recognition and Code Generation.

use a special continuation symbol in a particular column. The final way is to have no indication at all, as in PL/I. Consequently, the continuation definition must convey which method is utilized and the information relevant to the method.

The user has the option of overriding the recognition of an extended construct keyword if that keyword will conflict with a

feature already present in the language. For example, if the user wishes to code the IF-THEN and IF-THEN-ELSE constructs already available in PL/I instead of using the extended constructs, it would be necessary to include the keywords IF, THEN, ELSE, and ENDIF in the list on the OVERRIDE card. This would allow the user to include the desired constructs and prevent the preprocessor from mistaking them for extended constructs.

The second type of input is macro definitions. The use of macros is optional. Therefore, this part of the input may be omitted. Each macro definition has a name, parameter names, and replacement text for the macro. When the replacement text is inserted at a macro call, the parameter names in the replacement text are replaced by the actual arguments of the macro call.

There is a set of guidelines for writing macro replacement texts given in Appendix II. The user is encouraged to follow these guidelines to reduce the possibility of writing macro definitions which conflict with the principles of structured programming.

OUTPUT

Output is produced by the preprocessor in two forms, as a printed listing and a temporary disk file. The disk file may be input directly into the translator for the base language.

Two messages will be produced in the printed output which do not appear in the disk file. These messages indicate the success or failure of the parses. If a second parse is not required, a message to that effect is printed.

Each extended statement appears as a comment immediately before the code inserted by the preprocessor at the position of the extended statement. A comment appears every place a statement is inserted by the preprocessor, even if an extended statement has not been replaced by the inserted code. An example of this situation is the insertion of statements between the cases of a CASE structure even though no extended statement appears between the cases.

Each statement label appearing in the input program is detached from the associated statement and placed on a null statement. This null statement immediately precedes the original statement. This is necessary because a label and a statement cannot be reassociated after the statement has been recognized as an extended structure or macro call and replaced by inserted code. To maintain the proper locations of labels, they are placed on the null statements and positioned as described above.

There will be redundant comments and null statements when two parses have been made in processing the program. This is a result of label processing. The preprocessor will place all labels on null statements during the first parse. On the second parse, the label on each null statement created for a label during the first parse will be detached and placed on another null statement, and a comment will be generated which notes this insertion. Consequently, each label in the input program will result in two comments and two null statements when two parses are made during processing.

Post processing may alter this situation. For instance, the post processing for the PL/I language takes the label which has been

removed from the PROCEDURE OPTIONS statements and replaces it on that statement while eliminating the null statement(s) created for the label. For FORTRAN, the statement labels are returned to FORMAT statements in post processing.

ERROR CONDITIONS

The preprocessor does not detect errors in the base language portion of the input program. This task is left to the base language translator.

The preprocessor is a rather unforgiving tool. When an error is detected, the preprocessor outputs information pertinent to the cause of the error and terminates. Depending on what type of error occurred, the contents of the syntactic stack and certain variables may or may not be output. For any type of error, an error message and the output program produced so far are printed.

The common types of errors fall into three categories. The first type of error consists of all errors in specifying information about the base language and macro definitions. Among the errors falling in this category are incorrect definition syntax, missing definitions, invalid definition contents, and a missing ADDED DECLARATIONS statement when one is required. The user must be especially careful to provide a full eighty bytes in each of the six statement prototypes. This area is the only one in which the preprocessor is at all forgiving. If the preprocessor encounters an input record not containing a definition when one is expected, the preprocessor will skip that record. Normally this condition occurs when a statement prototype is not provided as a

full eighty bytes and the preprocessor uses the beginning of the next definition as the end of the previous one. Consequently, the preprocessor will later detect a missing definition and terminate processing.

The second type of error consists of omitting a keyword in a construct -- for example, leaving out the ENDIF following an IF - THEN. The preprocessor does not recognize this error until it attempts to process the ending keyword of an enclosing construct or at the end of the input program.

The third type of error includes syntax errors in writing extended constructs. This type of error generally results in abnormal termination because some expected keywords are not recognized. A good example of this type of error is the mistake of not leaving column one of a SNOBOL statement blank and placing a keyword for an extended construct in that position. The keyword will be recognized as a label instead of the appropriate keyword. This type of error may be considered a subset of the second type since the preprocessor does not recognize this error as a syntax error, but as an omission error.

When the preprocessor determines that an expected keyword is missing, either by omission or syntax error, the current contents of the syntactic stack are printed. This will help the user in determining what keyword was not recognized. The output program produced so far is printed to enable the user to determine at what point in analyzing the input the error was encountered.

SECTION IV. OBSERVATIONS

GOOD AND BAD ASPECTS OF PROJECT

Naturally, there are both good and poor aspects of this project. The good points include the fact that SNOBOL has been implemented on a variety of computers produced by various manufacturers [Gri72], so the preprocessor could be ported from machine to machine without a great deal of revision. Only statements incompatible between SNOBOL and SPITBOL would have to be modified.

Another good aspect of the preprocessor is that it is generalized for use with a variety of languages. Even though the macro feature is somewhat unsophisticated, this feature contributes to the preprocessor's generality. One could also note that the preprocessor is easy to use with regard to the syntax of the input program since the original syntax of the programming language is changed only minimally.

The modular design and implementation of the preprocessor is a good point about the project. This became quite apparent during total system testing. The total system tests were completed in only eight days.

Among the poor aspects of the project is the fact that both the storage utilization and execution of the preprocessor are inefficient. A more experienced SPITBOL programmer could probably have produced a more efficient product. However, efficiency was not a major goal of this project.

The overhead involved in the use of the preprocessor is enormous at this point. To require the user to provide and the preprocessor

to decode all the base language definitions for each execution is undesirable and costly. This matter is addressed later in this section.

The fact that the user must provide the base language information for each execution is admittedly cumbersome and costly, but not overly difficult for the user. Perhaps the first development of the language specifications is somewhat difficult, but this is an initial investment only. However, the user may find it possible to use the language specifications provided as examples in Appendix IV, in which case this poor aspect is minimized.

The author would have liked to use PL/I rather than SPITBOL to code the preprocessor because SPITBOL lacks many structured programming features. However, the pattern matching feature of SPITBOL was of considerable value in writing the preprocessor.

OVERALL APPRAISAL OF DIRECTION

The author feels that the concept behind the project, that is, to create a generalized tool to provide structured programming capability, is a valid one and extremely desirable. However, it is also recognized that the current implementation will likely receive no utilization outside the academic environment. This limited use will be a result of the lack of efficiency of the final product, the cumbersome requirements for specifying base language information, and the high cost of execution. Consequently, the author feels the project was a success as far as demonstrating that the concept is viable, but the final product will receive a minimum amount of practical use.

TIME, EFFORT, AND MONEY INVOLVED

The time which the project required is difficult to provide accurately because the project was stretched out over an extended period of time. From the initial design efforts to the completion of documentation was a time span of two years. Therefore, the following figures are based on person days and not calendar days.

Design	--- 2 months
Coding	--- 1 month
Unit Tests	--- 1 month
Integrated Tests	--- 8 days
Documentation	--- 2 months

Costs for computer time utilized in the testing phases amounted to $642.35. This figure was based on the following rates:

CPU time	1 min. = 9 units
K-byte time	1 min. = .037 unit
cards read	1 card = .0006 unit
lines printed	1 line = .0005 unit
pages printed	1 page = .007 unit
disk EXCP's	1 EXCP = .0002 unit

1 unit = $1

ELEMENTS TO BE CHANGED

The author hopes that one of the first changes to be implemented in the preprocessor will be to allow specifications for a language to be provided once and only once. After being provided once, they would be stored on tape or disk and retrieved when execution begins. A user would also be given the option of providing a new set of specifications to temporarily override those specifications stored in the system. This option was provided in the original design and subsequently eliminated during implementation.

Another desirable change would be to increase the efficiency of the preprocessor, both in storage utilization and execution time. A programmer knowledgeable in the intricacies of SPITBOL could accomplish this task.

A change to the keyword overriding policy would be useful. To allow the user to specify an alternate form of a keyword instead of simply eliminating it completely would alleviate a problem in using the preprocessor with the PL/I language. Currently, to use the macro facility with PL/I requires that there be no THEN keyword in the output program produced by the first parse unless it is part of an extended IF - THEN construct. This means no THEN keyword could be produced during the first parse except in the expansion of a macro call, but many of the extended constructs produce that particular keyword. It seems, therefore, that the user would be able to write very little useful code outside the macros.

POSSIBLE FURTHER DEVLOPMENTS

There are obviously possibilities for further development. One of these is to implement a more sophisticated macro facility. The macro feature should not be made elegant, because the preprocessor was not intended as a general purpose preprocessor. However, the techniques for manipulation of formal parameters and actual arguments might be upgraded to allow similar names for parameters and semantic stack manipulation.

One other possibility is to allow the user to use continuation cards in extended construct statements when continuation is indicated

by a symbol. This would involve deleting the continuation symbol when it is encountered and replacing it in all occurrences of continuation cards.

REFERENCES

Abr75 Abrahams, Paul. "Structured Programming" Considered Harmful, SIGPLAN Notices, Vol. 10, No. 4, April 1975, pp. 13-24.

Bau74 Bauer, F.L., and J. Eickel, eds. Compiler Construction -- An Advanced Course, Springer-Verlag, Berlin, 1974, pp. 356-365.

Bro67 Brown, P.J. ML/I Macro Processor, Comm. ACM, Vol. 10, No. 10, October 1967, pp. 618-623.

Bro69 Brown, P.J. A Survey of Macro Processors, Annual Review in Automatic Programming, Vol. 6, Part 2, 1969, pp. 37-88.

Bro74 Brown, P.J. Macro Processors and Techniques for Portable Software. Wiley and Sons, New York, 1974, pp. 53-58.

Cha71 Chandler, Gerald D. META PI -- A Language for Extensions, SIGPLAN Notices, Vol. 6, No. 12, December 1971, pp. 8-9.

Cah72 Dahl, O.-J., E.W. Dijkstra, and C.A.R. Hoare. Structured Programming. Academic Press, London, 1972, pp. 1-82.

Den74a Denning, Peter J. Is It Not Time to Define "Structured" Programming?, Operating Systems Review, Vol. 8, No. 1, pp. 6-7.

Den74b Denning, Peter J. Is "Structured Programming" Any Longer the Right Term?, SIGPLAN Notices, Vol. 9, No. 11, November 1974, pp. 4-6.

Dew71 Dewar, Robert B.K. SPITBOL, Version 2.0. Illinois Institute of Technology, February 1971.

Dij75 Dijkstra, E.W. Guarded Commands, Nondeterminacy, and Formal Derivation of Programs, Comm. ACM, Vol. 18, No.8, August 1975, pp. 453-457.

Fel66 Feldman, Jerome A. A Formal Semantics for Computer Languages and Its Application in a Compiler-Compiler, Comm. ACM, Vol. 9, No. 1, January 1966, pp. 3-9.

Fel68 Feldman, Jerome, and David Gries. Translator Writing Systems, Comm. ACM, Vol. 11, No. 2, February 1968, pp. 77-113.

Fer71 Fernandez, A., J.C. Heliard, and J.D. Ichbiah. Overview of the Syntax Processor SYNPROC, SIGPLAN Notices, Vol. 6, No. 12, December 1971, pp. 51-55.

Fou74 Foulk, C.R. Yet Another Attempt to Define Structured Programming, Operating Systems Review, Vol. 8, No. 2, July 1974, pp. 4-5.

Fri74 Friedman, Daniel P., and Stuart C. Shapiro. A Case for While-Until, SIGPLAN Notices, Vol. 9, No. 7, July 1974, pp. 7-14.

Gri71 Gries, David. Compiler Construction for Digital Computers. Wiley and Sons, New York, 1971, pp. 11-48,155-169.

Gri72 Griswold, Ralph E. Macro Implementation of SNOBOL. W.H. Freeman, San Francisco, 1972, pp. 240-242.

Hig75 Higgins, Donald S. A Structured FORTRAN Translator, SIGPLAN Notices, Vol. 10, No. 2, February 1975, pp. 42-48.

Kel Keller, Roy F. A Modern Beginning Programming Course. Computer Science Department and Ames Laboratory, ERDA, Ames, Iowa.

Lec74 Lecarme, Olivier. Structured Programming, Programming Teaching, and the Language Pascal, SIGPLAN Notices, Vol. 9, No. 7, July 1974, pp. 15-21.

Led75 Ledgard, Henry, and Michael Marcotty. A Genealogy of Control Structures, Comm. ACM, Vol. 18, No. 11, November 1975, pp. 629-639.

Luc73 Lucido, Anthony P. META-T and the Implementation of Special Application Languages, Proc. Computer Science and Statistics: 7th Annual Symposium on the Interface, 1973, pp. 412-417

Mat75 Mathis, Robert F. Flow Trace of a Structured Program, SIGPLAN Notices, Vol. 10, No. 4, April 1975, pp. 33-37.

McC75 McClure, Carma L. Structured Programming in COBOL, SIGPLAN Notices, Vol. 10, No. 4, April 1975, pp. 25-33.

Mei74 Meissner, Loren P. A Compatible "Structured" Extension to FORTRAN, SIGPLAN Notices, Vol. 9, No. 10, October 1974, pp. 29-36.

ONe68 O'Neil, John T., Jr. META PI -- An On-Line Interactive Compiler-Compiler, Proc. Fall Joint Computer Conference, 1968, pp. 201-218.

ONe74 O'Neill, Dennis M. SFOR -- A Precompiler for the Implementation of a FORTRAN-Based Structured Language, SIGPLAN Notices, Vol. 9, No. 12, December 1974, pp. 22-29.

San74 Sanfield, Stuart H. The Scope of Variable Concept: The Key to Structured Programming?, SIGPLAN Notices, Vol. 9, No. 7, July 1974, pp. 22-29.

Shn74 Shneiderman, Ben. The Chemistry of Control Structures, SIGPLAN Notices, Vol. 9, No. 12, December 1974, pp. 29-34.

Sol74 Solntseff, N., and A. Yezerski. A Survey of Extensible Programming Languages, Annual Review in Automatic Programming, Vol. 7, Part 5, 1974, pp. 267-307.

Vau74 Vaughn, W.C.M Another Look at the CASE Statement, SIGPLAN Notices, Vol. 9, No. 11, November 1974, pp. 32-36.

Weg75 Wegner, Eberhard. Control Constructs for Programming Languages, SIGPLAN Notices, Vol. 10, No. 2, February 1975, pp. 34-41.

APPENDIX I. DETAILED IMPLEMENTATION INFORMATION

This Appendix contains detailed information about the implementation of the preprocessor. It is intended as a guide to anyone wishing to make modifications to the source code. Consequently, comprehension will be severely limited unless the reader refers frequently to the source listing in Appendix V. The remainder of this Appendix describes the implementation by categories of modules as follows: driver, definition scanning routines, macro definition scanner, creation routines, service routines, semantic routines, insertion routines, parser, scanner, and character retrieval routine.

Driver

The driver of the preprocessor defines output associations, function calling sequences, and arrays. Patterns and variables are also initialized.

The driver reads the language name and checks it against those which the preprocessor can analyze. Each input card is read and appropriate processing is initiated. An OVERRIDE card is processed directly by the driver. The first definition card encountered causes the preprocessor to call SCANDEFS. The preprocessor calls the MACROSCAN routine when the first card of a macro definition is read. The preprocessor will begin processing the input program when the first card not recognized as a definition or an override card is read. The parse number indicator is set to one, and the parser is called. When control is returned to the driver, a test is made to determine if the parse was successful.

If the parse was unsuccessful, a branch is taken to the error exit for an unsuccessful first parse. If the parse was successful, a message indicating such is printed, the output files are rewound, and new input and output associations are defined. If no macro definitions were provided, a branch is taken around the second parse and the temporary print and punch files are copied to the next intermediate files. Otherwise, variables are reset to appropriate initial values, and the parse number indicator is set to two. The parser is called again, and a test for success is made when control is returned to the driver. If an error has occurred, a branch is taken to an error exit. Whether one or two parses are made, the intermediate files are rewound and new input and output associations are defined.

If the declaration statement prototype is non-blank, declarations are added for all new variables introduced by the preprocessor. The statement containing the string 'ADDED DECLARATIONS' is found in the output files and a declaration is added at that point for each variable name in the array VARLIST. Comments are also inserted to indicate that these declarations were added by the preprocessor. The remainder of the files are then copied. If no ADDED DECLARATIONS statement is found, an error exit is taken.

The next step is to execute post-processing if required for the language being processed. The intermediate files are rewound and new input and output associations are defined. The language type is checked.

If the language is SNOBOL, each statement is scanned for the ¬(symbols followed by blanks and ,(. At each occurrence, the blanks and comma are deleted.

If the language is FORTRAN, all sequences of 'comment-null statement-FORMAT statement' and 'comment-comment-null statement-null statement-FORMAT statement' are found. In each case, the statement number from the first null statement is placed on the FORMAT statement. The comments and the null statements are deleted.

If the language is PL/I, the first records of the output files are scanned for either one or two comments followed by the same number of null statements and a procedure statement. The label from the first null statement is placed on the procedure statement. The comments and the null statements are deleted, and the remainders of the files are copied.

If any other language is being processed (possible only if the language type pattern, LANGPAT, is enlarged to include other languages), the files are simply copied to the final output files.

The preprocessor prints a successful completion message and terminates.

The section to print the appropriate intermediate files on an error condition tests the parse number and rewinds the appropriate intermediate punch file. An input association is defined for the rewound file, and an output association with a real printer file is defined. The contents of the temporary punch file to be displayed are read and printed. An unsuccessful termination message is also printed and execution halts.

All error exits are provided in the driver. An explanatory message is printed for each error. Other information is also provided for certain errors as shown in Table A.I.1. If the preprocessor has begun analyzing the input program, a branch is taken to the section to print the intermediate file contents.

Definition Scanning Routines

One routine, SCANDEFS, controls all definition scanning except that for macro definitions. When the first definition is encountered, the first card of that definition is passed to this controlling routine. The definition is processed as described below, and another definition is read. After all nineteen definitions have been processed, or an attempt has been made to decode nineteen definitions, control is returned to the calling routine.

To process a definition, the SCANDEFS routine determines which one of fifteen individual scanning routines to call, based on the definition name. The definition name, its enclosing angle brackets, and the ::= symbol following it are stripped away. The remainder of the definition is then passed to the routine being called. If the definition is for a statement type, the routine reads another card and passes the entire prototype to the routine. If the definition is that for a language label, the routine determines whether or not the entire definition is contained on the single card. If not, the routine reads another input card and passes the entire definition to the routine for the LANGLABEL definition. If an attempt is made to read a definition, but the card contains an invalid definition name or no definition name in angle

ERROR EXIT	ERROR CONDITION	ADDITIONAL INFORMATION
LANGERR	invalid language type	language type provided in input
ENDPARSEERR	first parse terminated abnormally	contents of syntactic stack
END2ERR	second parse terminated abnormally	contents of syntactic stack
GTHANDEFERR	error in definition of greather-than operation	definition provided in input
LEQUDEFERR	error in definition of less-than-or-equal oper.	definition provided in input
GEQUDEFERR	error in definition of greater-than-or-equal oper.	definition provided in input
NOTDEFERR	error in definition of not operator	definition provided in input
NEQUDEFERR	error in definition of not-equal operator	definition provided in input
MACCALLERR	error in macro calling sequence definition	macro name
PSCANERR	no value returned from scanner	input statement being processed, contents of syntactic stack
PUNIDERR	parser unable to identify contents of syntax stack	input statement being processed, contents of syntactic stack
PSTRERR	failure to parse STRUC to STRLIST reduction	input statement being processed, contents of syntactic stack
PEDOGRPERR	failure to parse ENDDOGROUP	input statement being processed, contents of syntactic stack
PEDOERR	failure to parse ENDDO	input statement being processed, contents of syntactic stack

Table A.I.1. Error Exits with Additional Information Provided.

ERROR EXIT	ERROR CONDITION	ADDITIONAL INFORMATION
PIFTHERR	failure to parse IF - THEN clause	input statement being processed, contents of syntactic stack
PEWHERR	failure to parse ENDWHILE	input statement being processed, contents of syntactic stack
PECASEERR	failure to parse ENDCASE	input statement being processed, contents of syntactic stack
PQUERR	failure to parse QUITLOOP	input statement being processed, contents of syntactic stack
PWHERR	failure to parse WHILE	input statement being processed, contents of syntactic stack
PREPUERR	failure to parse REPUNTIL	input statement being processed, contents of syntactic stack
PREPERR	failure to parse REPEAT	input statement being processed, contents of syntactic stack
PIFERR	failure to parse IF	input statement being processed, contents of syntactic stack
PDGRPERR	failure to parse DOGROUP	input statement being processed, contents of syntactic stack
PDOERR	failure to parse DO	input statement being processed, contents of syntactic stack
PCASEERR	failure to parse CASE	input statement being processed, contents of syntactic stack
PEREPERR	failure to parse ENDREPEAT	input statement being processed, contents of syntactic stack

Table A.I.1. Error Exits with Additional Information Provided.(cont'd).

ERROR EXIT	ERROR CONDITION	ADDITIONAL INFORMATION
ELSEERROR	error in semantic processing of ELSE	portion of current input statement not processed by semantic routine
CASEERROR	error in semantic processing of CASE	portion of current input statement not processed by semantic routine
DOERROR	error in semantic processing of DO	portion of current input statement not processed by semantic routine
IFERROR	error in semantic processing of IF	portion of current input statement not processed by semantic routine
REPERROR	error in semantic processing of REPEAT	portion of current input statement not processed by semantic routine
WHERROR	error in semantic processing of WHERE	portion of current input statement not processed by semantic routine
QUITERROR	error in semantic processing of QUIT	portion of current input statement not processed by semantic routine
ECASEERROR	error in semantic processing of ENDCASE	portion of current input statement not processed by semantic routine
EIFERROR	error in semantic processing of ENDIF	portion of current input statement not processed by semantic routine
EWHILEERROR	error in semantic processing of ENDWHILE	portion of current input statement not processed by semantic routine
EREPERROR	error in semantic processing of ENDREPEAT	portion of current input statement not processed by semantic routine
EDOERROR	error in semantic processing of ENDDO	portion of current input statement not processed by semantic routine

Table A.I.1. Error Exits with Additional Information Provided (cont'd).

ERROR EXIT	ERROR CONDITION	ADDITIONAL INFORMATION
EDOGRPERROR	error in semantic processing of ENDDOGROUP	portion of current input statement not processed by semantic routine
DCLBLKERR	too few blanks in DECLARATION STATEMENT prototype to accommodate actual argument	actual argument
GOTOBLKERR	too few blanks in GO TO STATEMENT prototype to accommodate actual argument	input statement being processed
PLUSBLKERR	too few blanks in ASSIGN PLUS STATEMENT prototype to accommodate actual argument	input statement being processed
MINBLKERR	too few blanks in ASSIGN MINUS STATEMENT prototype to accommodate actual argument	input statement being processed
NULLBLKERR	too few blanks in NULL STATEMENT accommodate actual argument	input statement being processed
IFBLKERR	too few blanks in IF THEN GO TO STATEMENT prototype to accommodate actual argument	input statement being processed

Table A.I.1. Error Exits with Additional Information Provided (cont'd).

brackets, input cards are skipped until another definition is encountered. If the error occurs when the routine is attempting to read the nineteenth definition, control returns to the calling routine without reading any additional input cards.

The individual scanning routines called by SCANDEFS are summarized in Table A.I.2 and fall into four categories. The first category contains those routines which scan statement prototypes. Each of these routines has a single parameter which contains the prototype. Each prototype is broken down by the individual routines into components. Each parameter name and the blanks immediately following it comprise a component. All other parts before and between these components also constitute components. For example, the prototype for a GO TO STATEMENT contains two parameters -- a label (LAB) and a destination (DES). If the prototype for a GO TO STATEMENT were 'LAB BRANCH DES', the contents of the first component would be null. The second component would contain 'LAB '. The third component would contain 'BRANCH ', and 'DES' would be in the fourth component. Additional columns are considered to contain blanks through column eighty. The maximum number of components possible for each definition is based on the number of parameter names in the prototype. The numbers of components are as follows:

DEFINITION NAME	MAX. NUMBER OF COMPONENTS
IF THEN GO TO STATEMENT	11
GO TO STATEMENT	5
NULL STATEMENT	3
ASSIGN PLUS STATEMENT	9
ASSIGN MINUS STATEMENT	9
DECLARATION STATEMENT	3

The components for each prototype are placed in an array. The zeroeth element of each array is used to record the number of components. For the example given above, the zeroeth element would contain 4. The arrays associated with the definitions are as follows:

DEFINITION NAME	ARRAY AND DIMENSIONS
IF THEN GO TO STATEMENT	IFPAT(0:11)
GO TO STATEMENT	GOTOPAT(0:5)
NULL STATEMENT	NULLPAT(0:3)
ASSIGN PLUS STATEMENT	PLUSPAT(0:9)
ASSIGN MINUS STATEMENT	MINUSPAT(0:9)
DECLARATION STATEMENT	DCLPAT(0:3)

Each of the scanning routines which decodes a statement prototype functions in basically the same way. The parameter names for that statement type are placed in a SNOBOL pattern. The definition is searched for pairs of components, the second of which is a parameter name. The first component of the pair may be null, and the second component will include any blanks following the parameter name. If

the entire prototype has been scanned after entering the component pair into the appropriate array, the routine immediately records the number of components. If the last portion of the definition is not a parameter name, a component pair will not be found when that is all that remains to be scanned. Therefore, the remainder of the prototype is placed in the array and the number of components is then recorded. In either case, once the number of components has been placed in the array, the routine tests to see if all the necessary parameters have been specified. The exception to this procedure is in the routine which scans the DECLARATION STATEMENT prototype. Since a blank prototype may be specified for this statement type, no check is made for the appropriate number of parameter names.

Another category of definition scanning routines contains those routines which scan the definitions of unique identifiers, unique labels, literals, and continuations. These scanning routines are similar because the numbers and types of parameters are fairly stable. Both the identifier and label definitions must have a string followed by an integer. Another string may optionally follow the integer. The literal definition contains two strings, and a third string is optional. The continuation definition contains either the single word NONE or a string followed by a column number. Each of these components is placed in an element of the array associated with the definition as indicated by the following:

DEFINITION NAME	ARRAY AND DIMENSIONS
UNIQUE IDENTIFIER	VARPAT(1:3)
UNIQUE LABEL	LABELPAT(1:3)
LITERAL	LITPAT(1:3)
CONTINUATION	CONTPAT(1:2)

The third category of scanning routines contains those routines which scan the definitions for end-of-statement, ignored columns, comment, and language label. Each of these definitions has alternative forms of information, the types of which must be recorded in the array associated with the definition. The arrays and the definitions with which they are associated are:

DEFINITION NAME	ARRAY AND DIMENSIONS
END OF STATEMENT	EOSPAT(0:1)
IGNORE COLUMNS	IGNPAT(0:2)
COMMENT	COMPAT(1:6)
LANGLABEL	LANLABPAT(1:9)

The END OF STATEMENT definition may specify either a symbol string or a column number. Based on the contents of the definition, either 'STR' or 'COL' is placed in the zeroeth element of EOSPAT. The string or column number specified is placed in the next element. The IGNORE COLUMNS definition may specify either a single column number or a sequence of column numbers separated by a hyphen. If only one column number is specified, 'ONE' is placed in the zeroeth element, and the column number is placed in element one. If a sequence is specified,

'SEQ' goes into the zeroeth element, and elements one and two will contain the two column numbers given as the bounds of the sequence.

The COMMENT definition contains two sets of information. Each set may be a symbol string, a column number, or both with a '+' between them. Element one of the associated array contains an indication of what is present for the comment opener. Element four contains similar information for the comment closer. The information is specified as a string with the following meanings:

STRING	MEANING
10	symbol string only
01	column number only
11	symbol string and column number

Elements two and three contain the string and column number, if present, for the opener. Elements five and six contain the string and column number, if present, for the closer.

The language label definition contains four pieces of information. The first may be either a string, column number, or a sequence of columns represented by two column numbers separated by a hyphen. The first element of the language label array contains either 1 or 2, for one column number or a sequence of them, respectively. The fourth element contains the single column number or the first of the two column numbers specified as the sequence bounds. The fifth element contains the second column number of the sequence specification. The second piece of information may be of the same form as the first or may be a symbol string. If the column or column sequence form is specified, the

ROUTINE NAME	'PATTERN' CREATED	PARAMETERS
SCANNULLDEF	NULLPAT(0:3)	NULLDEF
SCANGOTODEF	GOTOPAT(0:5)	GOTODEF
SCANIFDEF	IFPAT(0:11)	IFDEF
SCANPLUSDEF	PLUSPAT(0:9)	PLUSDEF
SCANMINUSDEF	MINUSPAT(0:9)	MINUSDEF
SCANDCLDEF	DCLPAT(0:3)	DCLDEF
SCANVARDEF	VARPAT(1:3)	VARDEF
SCANLABELDEF	LABELPAT(1:3)	LABELDEF
SCANLITDEF	LITPAT(1:3)	LITDEF
SCANCONTDEF	CONTPAT(1:6)	CONTDEF
SCANCOMDEF	COMPAT(1:6)	COMDEF
SCANEOSDEF	EOSPAT(0:1)	EOSDEF
SCANIGNCOLDEF	IGNPAT(0:2)	IGNCOLDEF
SCANLANLABDEF	LANLABPAT(1:9)	LANLABDEF
SCANOPERDEF	GTHAN	TYPEDEF, OPERDEF
	GEQU	
	LEQU	
	NEQU	
	NOT	
SCANDEFS		CARD

Table A.I.2. Scanning Routines.

information is stored as indicated above, but in elements three, six and seven. If a symbol string is given, 'STR' is placed in element two, and the string is placed in element six. The third and fourth pieces of information in this definition are character strings. They are placed in elements eight and nine, respectively.

The fourth and final category of scanning routines contains a single routine which scans the definitions for the five relations -- greater than, less than or equal, greater than or equal, not equal, and not. Each of these definitions has a single variable associated with it in which the symbol string for the appropriate operator is placed. The definitions and their associated variables are as follows:

RELATION	VARIABLE
GREATER THAN	GTHAN
LESS THAN OR EQUAL	LEQU
GREATER THAN OR EQUAL	GEQU
NOT EQUAL	NEQU
NOT	NOT

Macro Definition Scanner

The macro definition scanner (MACROSCAN routine) is not included with the other scanning routines because it has some basic structural and technique differences from those routines. It is called directly from the driver. It is called once for each macro definition and is therefore likely to be called more than once per execution of the preprocessor. The information extracted is stored in a SNOBOL table as opposed to simply an array.

The macro scanning routine is called when the first card of a macro definition is encountered by the driver. This card contains the string '<MACRO CALL>::=' followed by the name of the macro. The macro name is immediately followed by its formal parameters separated by commas and enclosed within a single set of parantheses.

EXAMPLE:

<MACRO CALL>::=WHEN(X,Y)

MACROSCAN deletes the angle brackets and their contents before extracting the macro name and the parameters. A second card is read if the parameter list extends past the end of the first card. No blanks, imbedded or at the end of the first of two cards, are allowed. The zeroeth element of the MPARM array is set equal to the number of formal parameters provided. The first parameter is placed in element one of MPARM and succeeding parameters are placed in consecutive elements of MPARM.

The next card is read and should contain only the string '<MACRO DEFINITION>::='. The cards following are read and their contents placed in the MACREPSTR variable until the end of the definition is found. The end is signified by a card containing only the string 'END MACRO DEFINITION' beginning in column one. This string is not added to MACREPSTR.

The parameter list pattern is started with a series of 81 blanks. The formal parameters are then added to this pattern from the MPARM array. The macro scanner next breaks the MACREPSTR contents into components and stores those components in the MPAT array. The zeroeth element of MPAT is assigned the number of components in the replacement text. A component is defined as for the statement scanning routines.

The contents of the MPAT array are copied into the macro table MACROTAB indexed by the macro name in the form of a string. The parameter list for the macro, currently in the MPARM array, is copied into the element of MACROTAB indexed by the macro name in the form of a string suffixed with the character 'P'. The routine then outputs a message that this macro definition has been processed and returns to the driver.

Creation Routines

There are eight routines which create items for the preprocessor. Six of them create statements. The other two create unique identifiers and labels. Each of the statement-creating routines has a number of parameters equal to the number of parameter names which appear in the corresponding statement prototype. Each of these routines utilizes the array containing the appropriate statement components. By replacing the components containing parameter names with the corresponding arguments supplied in the call to the creating routine and including enough blanks to retain the format and length of the prototype (to eighty characters) in the output, the routines create statements which are standard in the base language.

The routines which create statements, their outputs and parameters are as follows:

ROUTINE NAME	OUTPUT STATEMENT	PARAMETERS
CRNULL	NULL STATEMENT	LAB
CRIF	IF THEN GO TO STATEMENT	LAB,VAL1,RL, VAL2,DES
CRGOTO	GO TO STATEMENT	LAB,DES
CRMINUS	ASSIGN MINUS STATEMENT	LAB,VAR,VAL1, VAL2
CRPLUS	ASSIGN PLUS STATEMENT	LAB,VAR,VAL1, VAL2
CRDCL	DECLARATION STATEMENT	VARNAME

The routines which create unique identifiers and labels also utilize the arrays associated with the definitions provided. VARNUM and LABELNUM are the variables used to make the identifiers and labels unique. Each routine uses the appropriate variable and then increments it by one. Enough leading zeroes are used with these variables, which are initialized to one, to create items of the lengths specified in the definitions. The routine which creates unique identifiers adds each one to the array VARLIST which is indexed using the variable NEXTVAR. This array is later used to add declarations to the program if required. Neither UNILABEL nor UNIVAR has any parameters. The routines and their outputs are as follows:

ROUTINE NAME	OUTPUT
UNIVAR	unique identifier
UNILABEL	unique label

Service Routines

Two service routines are present in the preprocessor which manipulate the semantic stack. The routines are POP.SEM and PUSH.SEM which pop items off and push items onto the semantic stack, respectively.

The semantic stack is represented by the array SEMSTACK which contains ten elements. Element 1 represents the element at the bottom of the stack. The variable TOP contains the subscript of the top element in the stack.

Each of these routines may have from two to seven parameters. The first parameter is always the number of items to be acted upon, that is, the number of items to pop from or push onto the stack. The second through seventh parameters are passed differently in the two routines. Variables are passed to the POP.SEM routine in which the popped items will be returned. These variables' names are passed as character strings, and values are assigned to the corresponding arguments using two levels of indirection. Variables passed to the PUSH.SEM routine as parameters two through seven contain values to be pushed onto the stack. The values are assigned from the corresponding arguments using only one level of indirection.

POP.SEM extracts a value from the top of the stack and decrements the stack pointer, TOP. PUSH.SEM increments TOP, the stack pointer, and assigns a value to the stack's new top element. Each routine loops until the number of values indicated by the first argument have been processed.

Semantic Routines

Most of the semantic routines call the statement creation routines. These routines are always called with a label value. This is necessary because if a symbol is normally used for terminating a label, an extraneous symbol will appear in the output if a null value is passed as the label.

Semantic Routine 0 - This semantic routine is called when the parser recognizes a label on a statement. The routine inserts a null statement with the label just recognized. If a language label is terminated by a symbol, that symbol is removed from the label before the null statement is created. This step is necessary because the terminating symtol is already provided in the null statement prototype.

Semantic Routine 01 - This routine processes items recognized as unidentified-to-end-of-statement (U-ECS) and macro calls. Any sequence of one or more items terminating in the end-of-statement condition and not containing any token recognized as part of an extended construct is reduced to the U-EOS token. This routine checks the string recognized as U-ECS to determine if it is a macro call.

If the U-ECS is a macro call, the actual arguments of the macro call are broken out and placed in an array. The macro's formal parameters and the number of parameters, as well as the macro's replacement text components are retrieved from the macro table. The parameters are placed in a SNOBOL pattern. The components of the replacement text are copied into a string, replacing the formal parameter names with actual arguments. The macro call is made into a comment and inserted into the output, followed by the expanded macro text.

If the sequence recognized as an U-ECS does not contain a macro call, the statement is copied directly from the input to the output.

Semantic Routine 1 - This routine is called by the parser when the extended construct keyword ELSE has been recognized. The routine

pops a label name from the semantic stack and creates another unique label. An unconditional branch to the new unique label and a null statement with the label popped from the semantic stack are inserted into the output, following a comment indicating that the ELSE keyword was processed at that point. The unique label created in this routine is pushed onto the semantic stack.

Semantic Routine 2 - This routine processes the statement which begins a CASE extended construct. The index of the CASE is extracted and two unique labels are created. A counter indicating the nesting level of the structure is incremented by one. Another counter which records the number of CASE structures currently started but not finished and a counter indicating which case within the current CASE structure is being processed are incremented. A single conditional branch is created which branches to the second unique label created in this routine if the value of the index of the CASE is not equal to one (the value of the case counter for this CASE structure). This statement is inserted into the output after the CASE statement in the form of a comment. The index of the CASE, the label which was the destination of the conditional branch, and the first unique label created in this routine are pushed onto the semantic stack.

Semantic Routine 3 - DO statements are processed by this semantic routine. The items of the DO -- the index, beginning value, ending value, and increment value -- are broken out of the statement. If no increment value is specified, a default value of 1 is used. A unique label and a unique identifier are created, and the counter

indicating the nesting level of the structure is incremented. Two assignments with subtraction and one assignment with addition are created and inserted in the output. The first subtraction assigns the unique identifier the value of the ending value less the increment value. The second subtraction assigns the index the beginning value less the increment value. The addition has the unique label and assigns the index the value of the index plus the increment. The DO statement is made into a comment and inserted into the output, followed by the three created statements. The increment value, the index of the DO, the unique identifier, and the unique label are pushed onto the semantic stack.

Semantic Routine 4 - A DOGROUP statement is processed by this routine. The DOGROUP statement is made into a comment and inserted in the output. The counter indicating the structure nesting level is incremented.

Semantic Routine 5 - When the parser recognizes an IF - THEN statement, this routine is called to process it. The Boolean condition is extracted from the statement and a unique label is created. A conditional branch which tests the negation of the Boolean condition and branches to the unique label on truth of the negation is inserted in the output following the IF - THEN statement in the form of a comment. The unique label is pushed onto the semantic stack.

Semantic Routine 6 - This routine processes a REPEAT UNTIL statement. The Boolean condition is extracted, a unique label is created, and the nesting level counter is incremented. A null statement with the unique label is created and inserted after a comment containing the REPEAT UNTIL. The unique label and the Boolean condition are pushed onto the semantic stack.

Semantic Routine 7 - A WHILE statement is processed by this semantic routine. The Boolean condition of the WHILE is extracted and two unique labels are created. The nesting level counter is incremented. A conditional branch with the first unique label is created which branches to the second unique label if the negation of the Boolean condition is true. The WHILE statement is converted to a comment and inserted in the output, followed by the created statement. The second and first unique labels are pushed onto the semantic stack, in that order.

Semantic Routine 8 - This semantic routine processes the QUITLOOP extended construct. A unique label is created and an unconditional branch to that label is created. The QUITLOOP statement is converted to a comment and inserted in the output, followed by the created branching statement. The unique label and the string 'QUIT' are pushed onto the semantic stack.

Semantic Routine 9 - The parser calls this routine when an ENDCASE has been recognized. Three items are popped off the semantic stack. Two null statements are created with the second and first items from the stack as labels. The number of cases for this CASE structure is reset to zero. The number of CASE structures and the nesting level are both decremented by one. The created statements are inserted following the ENDCASE in the form of a comment.

Semantic Routine 10 - This semantic routine is called by the parser to process an ENDIF. One item is popped from the semantic stack. There are two possible types of values for this item, the string 'QUIT' or a label. If the item is the character string 'QUIT', two more items are popped off the stack and the first of

those two items and the string 'QUIT' are pushed back onto the stack. If the item is not the string 'QUIT', it should be a label.

A null statement using the item from the stack is created and inserted into the output following the ENDIF in the form of a comment.

Semantic Routine 11 - This semantic routine processes the ENDWHILE statement. Two items are popped off the semantic stack and the nesting level counter is decremented by one.

If the first item popped off the semantic stack is the character string 'QUIT', the second item is the label which was the destination of the branch created for a QUITLOOP structure occurring within the WHILE - ENDWHILE construct. The variable indicating the presence of an enclosed QUITLOOP structure is set to one. Two more items are popped off the semantic stack. These items are now considered to be the first and second items popped off the stack.

Whether or not a QUITLOOP structure has occurred, an unconditional branching statement is created with the first item popped off the stack as the destination. A null statement with the second item popped off the stack as the label follows the unconditional branch. These two statements are inserted in the output following the ENDWHILE statement in the form of a comment.

If the QUITLOOP structure indicator was set, another null statement is created. This statement has the label which was the destination of the unconditional branch created for the QUITLOOP occurrence and is inserted in the output.

Semantic Routine 12 - The ENDREPEAT statement is processed by this semantic routine. Two items are popped off the semantic stack and the nesting level counter is decremented by one.

If the first item popped off the semantic stack is the character string 'QUIT', the second item is the label which was the destination of the branch created for a QUITLOOP structure occurring within the REPEAT UNTIL - ENDREPEAT construct. The QUITLOOP occurrence indicator is set to 1 and two more items are popped off the semantic stack. These items are now considered to be the first and second items popped off the stack.

Whether or not a QUITLOOP structure has occurred, the second item popped off the stack is used as the destination of a conditional branching statement which tests the negation of the first item popped off the stack. The ENDREPEAT is converted to a comment and inserted in the output, followed by the created conditional statement.

If the QUITLOOP occurrence flag was set, a null statement is created with the label used as the destination of the branching statement which replaced the QUITLOOP statement. This null statement is inserted in the output.

Semantic Routine 13 - The parser calls this semantic routine when an ENDDO has been recognized. Four items are popped from the semantic stack and the nesting level counter is decremented by one.

If the first item popped off the semantic stack is the character string 'QUIT', the second item is the label created as the branching destination during QUITLOOP processing. Two more items are popped from the semantic stack. These are added to the third and fourth

items popped previously, and the four items are considered to be the first four items popped from the stack. The QUITLOOP occurrence flag is set to one.

Whether or not a QUITLOOP structure has occurred, two unique labels are created. Five statements are created to be inserted in the output following the ENDDO statement in the form of a comment.' The first statement is a conditional branch which tests if the increment value (the fourth item popped from the stack) is positive. The destination of this statement is the first of the two new unique labels. Another conditional branching statement is the second created statement. It has the first item popped off the stack as its destination and tests if the index value (third item from the stack) is greater than or equal to the second item popped from the stack. An unconditional branch with the second unique label as its destination is created next and is followed by another conditional branch. This statement has the first unique label as its label, tests if the index value is less than or equal to the second item popped from the stack, and branches to the first item popped off the stack if the condition is true. The fifth created statement is an assignment statement with an addition operation. The statement's label is the second new unique label and it increments the index value by the increment value.

If the QUITFLAG occurrence flag was set, a null statement is created and inserted in the output. The label on this statement is the destination label of the unconditional branch created during QUITLOOP processing.

Semantic Routine 14 - This semantic routine processes the ENDDOGROUP statement. The statement is converted to a comment and inserted in the output. The nesting level counter is decremented by one.

Semantic Routine 15 - This semantic routine is called whenever the parser reduces an item or series of items to the STRLIST (structure list) token.

The routine returns immediately if there are no unfinished CASE structures or if the preprocessor is processing statements within another structure contained in a CASE structure. Otherwise, the number of cases for this CASE structure is incremented by one, and a unique label is created. Three items are popped from the semantic stack.

An unconditional branch with the first item popped from the semantic stack as the destination is created. A conditional branch with the second item from the semantic stack as the label is created next. This statement tests if the third item from the semantic stack (index of the CASE) is not equal to the counter incremented above and branches to the new unique label if the condition is true.

A comment noting that two statements are being inserted is created and inserted in the output. The two branching statements are inserted immediately following the comment.

The index of the CASE, the new unique label, and the first item popped from the stack are now pushed onto the stack, in that order.

Insertion Routines

There are three routines which insert strings into the output, each in its own way.

Insertcom - The single parameter COMSTR contains a string to be converted to a comment and inserted in the output. The routine will attempt to start the comment in column one. If this column is to be ignored, the routine determines where to start the comment. If a sequence of columns has been specified for the ignored columns in the IGNORE COLUMNS definition, the preprocessor discovers the first column which is not an ignored column. If no sequence was specified, the preprocessor uses the first column following the ignored column.

If a comment begins in a specific column, the columns preceding that column are filled with blanks. If a symbol string starts the comment in that column, the string is added, followed by the parameter. If no symbol string is needed, the parameter is added to the comment being built. The comment is truncated, if necessary, to eighty characters.

If a comment ends in a particular column, the string is truncated or padded with blanks, as necessary. If a comment ends with a certain string, the current comment string is truncated as necessary to maintain only eighty characters. If the comment will end in an ignored column, enough characters are deleted within the parameter string portion of the comment to ensure the comment ender will not appear in an ignored column. The comment closing string is added after the appropriate position has been determined. Enough blanks are added, if necessary, to make the final string eighty characters long. Finally, the comment is inserted in the output.

Insertstmt - The single parameter STR.INSRT contains the string to be inserted in the output as a statement. A counter of the positions in the output string (I) and a counter of the positions

in the parameter string (CHARKTR) are set to zero. The variable TYPE is set depending on whether a single column or a sequence of columns was specified in the IGNORE COLUMNS definition.

CHARKTR and I are both incremented by one. The final portion of the statement has been constructed if CHARKTR is equal to the number of characters in the parameter string. An entire card has been constructed if I is equal to 80. Branches are made if either of these conditions occurs.

A branch is made to check the current position in the output (value of I) against ignored columns. If only one column is ignored and the column currently indicated by I is that column, a blank is inserted. If a sequence of columns is specified and the column indicated by I is one of the ignored columns, a blank is inserted. A branch is made to increment I and continue processing this statement. In either case, if the column to which I points is a usable column, the character in the position of the parameter string indicated by CHARKTR is added to the output string being constructed and a branch is made to increment both CHARKTR and I.

When an entire card image has been constructed, it is inserted in the output. I is reset to 1 and the card image area is nullified. The preprocessor then branches to check for a column to ignore.

When the last portion of a statement has been completed, this portion is padded with blanks and inserted in the output.

Insert - The single parameter INSRT.STR contains the statement to be inserted. This string contains blanks in ignored columns. The statement is broken into sections of eighty characters and

inserted in the output one at a time. If the last section of the statement is less than eighty characters long, this portion is padded with blanks before inserting it in the output.

Parser

The structure parser has no parameters. It is called once by the driver for each parse desired. Control is returned to the point of invocation when the entire input program has been analyzed. Error conditions take special error exits out of the parser.

When the parser is called, it immediately requests a token be placed on the syntactic stack by the scanner. The parser then tests the contents of the syntactic stack, reduces those contents to appropriate tokens, and calls semantic routines and the scanner according to the strategy of the the modified Production Language in Table A.I.3.

Scanner

The scanning routine retrieves single characters from the input program and determines what syntactic tokens are to be presented to the parser. A character string is built of the input characters, and when an item is recognized, a token is placed on the syntactic stack and return is made to the parser. The scanner makes tests in the following order: comment, literal, end of statement, keyword appearing at end of statement, end of a language label, blank character, extended construct keyword followed by a blank. If the scanner is in the middle of a statement, the test for end of a language label is skipped. If the end of statement condition is not recognized, the test for a

LINE NO	STACK TOP	REDUCE TO	CALL SEMANTIC ROUTINE	SCAN	ON SUCCESS, BRANCH TO TEST ON LINE #
1	LABEL	LABEL	ROUT0	yes	1
2	ENDCASE	ENDCASE	--	yes	26
3	ENDIF	ENDIF	--	yes	27
4	ENDWHILE	ENDWHILE	--	yes	29
5	ENDREPEAT	ENDREPEAT	--	yes	30
6	ENDDO	ENDDO	--	yes	31
7	ENDDOGROUP	ENDDOGROUP	--	yes	32
8	ELSE	ELSE	ROUT1	yes	1
9	QUITLOOP	QUITLOOP	--	yes	25
10	CASE	CASE	--	yes	18
11	DO	DO	--	yes	19
12	DOGROUP	DOGROUP	--	yes	20
13	IF	IF	--	yes	21
14	REPEAT	REPEAT	--	yes	22
15	WHILE	WHILE	--	yes	24
16	STRLIST EOF	PROG	--	no	return
17	UEOS	STRUC	ROUT01	no	33
18	CASE UEOS	CASESTMT	ROUT2	yes	1
19	DO UEOS	DOSTMT	ROUT3	yes	1

Table A.I.3. Production Language for Parser.

LINE NO	STACK TOP	REDUCE TO	CALL SEMANTIC ROUTINE	SCAN	ON SUCCESS, BRANCH TO TEST ON LINE #
20	DOGROUP UEOS	DOGROUPSTMT	ROUT4	yes	1
21	IF UTHEN	IFTHEN	ROUT5	yes	1
22	REPEAT UNTIL	REPUNTIL	--	yes	23
23	REPUNTIL UEOS	REPEATSTMT	ROUT6	yes	1
24	WHILE UEOS	WHILESTMT	ROUT7	yes	1
25	QUITLOOP UEOS	STRUC	ROUT8	no	33
26	CASESTMT STRLIST ENDCASE UEOS	STRUC	ROUT9	no	33
27	IFTHCL ENDIF UEOS	STRUC	ROUT10	no	33
28	IFTHELSE ENDIF UEOS	STRUC	ROUT10	no	33
29	WHILE STMT STRLIST ENDWHILE UEOS	STRUC	ROUT11	no	33
30	REPEATSTMT STRLIST ENDREPEAT UEOS	STRUC	ROUT12	no	33
31	DOSTMT STRLIST ENDDO UEOS	STRUC	ROUT13	no	33
32	DOGROUPSTMT STRLIST ENDDOGROUP UEOS	STRUC	ROUT14	no	33
33	LABEL STRUC	STRUC	--	no	33
34	ELSE STRUC	ELSE STRUC	--	no	33

Table A.I.3, Production Language for Parser (cont'd).

LINE NO	STACK TOP	REDUCE TO	CALL SEMANTIC ROUTINE	SCAN	ON SUCCESS, BRANCH TO TEST ON LINE #
35	LABEL ELSESTRUC	ELSESTRUC	--	no	33
36	IFTHCL ELSESTRUC	IFTHELSE	--	yes	1
37	IFTHEN STRUC	IFTHCL	--	yes	1
38	STRLIST STRUC	STRLIST	ROUT15	yes	1
39	STRUC	STRLIST	ROUT15	yes	1

Table A.I.3. Production Language for Parser (cont'd).

keyword appearing at the end of a statement is skipped. If the blank character test is not satisfied, the test for an extended construct keyword followed by a blank is skipped. A minimum amount of backup is used in the scanner. A more detailed description is contained in the remainder of this section.

The scanner begins by recording whether or not an end-of-statement condition (e-o-s) was encountered in the last card scanned. If the variable EOSNEXT is equal to one, that is, the e-o-s condition should be signalled immediately, a branch is taken to return an e-o-s condition. If EOSNEXT is not one, the GETCHAR function is called to return a character and its column number. If this character is the first character of a token or the first non-blank character of a token, the starting column of the token is recorded. If this column is the first non-ignored column of a card, it is determined whether or not all blanks left over from a previous card should be discarded. If only a comment was found on the previous card, these blanks will be retained. Another condition under which the blanks will be retained is when a comment was not the only thing found on the previous card and an e-o-s condition was found in the previous call to the scanner.

Whether or not the blanks are retained, the next step is to add the new character to the current token. The label start position variable and the comment only indicator are set to zero.

The scanner then examines the token string for the beginning of a comment. There are three different forms of comment openers to look for, and the scanner uses whichever one corresponds to the definition provided by the user. When the column opening string is two characters

long, it is necessary for the preprocessor to inspect the character following an occurrence of the first character of the string. If the second character is not the second of the specified string, a backup procedure is invoked by restoring the first character thought to open the comment and its column number. If a comment opener is not recognized, the scanner branches to test for a literal. If the beginning of a comment is found, the e-o-s flag is reset to the value it contained before the call to the scanner, the comment-only indicator is set to one if nothing else has been encountered in this statement, and a search is made for the end of the comment.

The search for the end of the comment also has three possible methods and is controlled by the definition of a comment for this particular language. If the comment closer is a string of two characters, the backup procedure will again be utilized if necessary. When the end of the comment is found, the token, which consists of the text of the comment and any opening and/or closing strings, is added to the current statement string, and the token variable is set to null. A branch is then taken to the point near the beginning of the scanner where another character is retrieved, thus avoiding the manipulation and test of the e-o-s indicator. The comment is not returned to the parser.

If the test for a comment is unsuccessful, the scanner analyzes the input characters for a literal. The literal beginning may be defined as one or two characters. If a two-character opening is used, the backup procedure is used if necessary. If the beginning of a

literal is not recognized, the scanner branches to test for an e-o-s condition.

If the literal opening is found, a search is made for the end of the literal. In searching for the end, special processing is utilized for any special sequence specified in the literal definition. If the first character of the ending string, which is the first character of any special sequence, is encountered, a test is made for the special sequence if one was provided. If a special sequence is found, the scanner continues to search for the literal end. Otherwise, a test is made for the end of the literal at that point. If a literal ender is not found, the scanner continues to search for it. If the end is recognized, the literal is added to the token string and a branch is taken to the portion of the scanner which processes items not identified as extended constructs or comments.

The next section of the scanner tests for the e-o-s condition. The scanner utilizes the information provided by the user in the end of statement definition to analyze the input in this section. If the end of statement is signified by one or two symbols, the scanner tests for these characters, using the backup procedure for a two-character string if necessary. If the end of statement is signalled by a column, the scanner tests if the column number of the most recently retrieved character is that specified in the end-of-statement definition. If the end-of-statement condition is not recognized, a branch is taken to test for a language label. If what appears to be an end-of-statement condition is recognized and that condition is indicated by a column

number, the next input card is read and tested to determine if it is a continuation. If so, a branch is taken to the section of the scanner which tests for a blank character.

If the end-of-statement condition is recognized, the token string is added to the current statement. It is necessary at this point to determine if an extended construct keyword ends at the end of the statement but has not been recognized. This will be the case if e-o-s is indicated by a symbol and a keyword ends immediately preceding that symbol or if e-o-s is indicated by column number and a keywcrd ends immediately preceding a blank in the ending column or in the ending column itself. If a keyword is recognized in this manner, it is necessary to set the EOSNEXT variable to one so the scanner will signal e-o-s immediately upon its next invocation. The keyword found is added to the syntactic stack and control is returned to the parser. If a keyword is not found at the end of the statement, the e-o-s flag is set for this statement, the EOSNEXT flag is reset to zero, the 'unidentified to end of statement' (UEOS) token is added to the syntactic stack, the token string is set to null and control is returned to the parser.

The next portion of the scanner tests for the end of a language label if this is at the beginning of a statement. If there is anything in the current statement string followed by a blank or only blanks occur in the current statement string, the test for the end of a language label is not made, and a branch is made to the section of the scanner which tests for a blank character.

The test for the end of a language label is based on the definition provided for language labels. If the definition specifies a column

number or a series of column numbers as the determination of the end of a language label, the first test will be satisfied if the current column is the single column number or within the specified sequence. Otherwise, a branch is taken to the section of the scanner which tests for a blank character.

If the definition specifies a symbol to terminate a language label, the current character is tested against the specified symbol. If a two-character language label separator is specified, the next character is tested and backup is used if necessary to restore the previous character as the current character. If the symbol string does not match, a branch is taken to the section of the scanner which tests for a blank character.

Once the scanner has determined that this character/column could be terminating a language label, the label is extracted from the token string. It is either the entire token or the last portion of it. The column number the label actually started in is computed. The column number computed is compared with the starting column value(s) specified in the language label definition. If this column number cannot start a language label, any necessary backup is performed and a branch is taken to the section which tests for a blank character. At this point the scanner knows a label has been found if a string is the label terminator. If the label is terminated by column numbers, the first character of the label is checked to see if it is a valid character for the start of a label. Any remaining characters are checked against the characters allowable for the remainder of the label. If either of these tests fails, any necessary backup is performed

and a branch is taken to the section of the scanner which tests for a blank character. If these two tests are passed, the next input character is tested for a label delimiter, that is, a blank or a special character which is allowed directly following a label. If this test fails, backup is performed and a branch is taken to the section which tests for a blank character.

If a label is recognized by the scanner, the label is placed in a comment indicating that it was removed from the statement and placed on a null statement inserted immediately preceding the current statement. This insertion function will be performed by the semantic routine which will be invoked when the scanner returns control to the parser. The label's character positions in the current statement are replaced by blanks, the LABEL token is placed on the syntactic stack, and return is made to the parser.

The next section of the scanner tests for a blank character. A blank encountered at this point in the scanner's execution causes a branch to the section to search for a keyword. If the character is any non-blank character, a branch is made to the BEGSCAN1 label in the scanner. The effect of this branch is to allow the scanner to read the next input character and begin its analysis once again.

The next section tests for a keyword. The first of two tests compares the token string with extended construct keywords which begin or end constructs, such as DO and ENDDO. The second test is against the extended construct keywords which appear in the middle of extended constructs, that is, the keywords THEN and UNTIL. If the token string does not satisfy either of these tests, a branch is taken to the section of the scanner which handles items not yet

recognized. If the keyword for THEN has been recognized, the unidentified-to-then (UTHEN) token is added to the syntactic stack. Otherwise, the keyword itself is added to the syntactic stack. The token string is added to the current statement, the token string is set to null, and control is returned to the parser.

The section which processes unidentified items simply branches to the BEGSCAN1 label in the scanner to read the next input character and begin its analysis again.

A section at the end of the scanner tests for an error condition when the scanner attempts to retrieve another input character at the beginning of the scanner but fails. If the token string is currently all blanks or is of zero length, a test is made to determine whether or not an e-o-s condition was found before this call to the scanner. If so, the failure to return a character indicates the end-of-file condition. The EOF token is added to the syntactic stack and control is returned to the parser. If an e-o-s condition had not been discovered, an unended statement occurs at the end of the input and an error exit is taken. If the token string contains at least one non-blank character when the failure to return a character is encountered, an error exit is taken.

Character Retrieval Routine

The GETCHAR routine retrieves a single character from the input program and returns the character and its column number to the scanner. The two parameters are CHAR and COL, which will contain the character and its column number, respectively, on a successful return. These

arguments are passed as variable names in the form of character strings and one level of indirection is used to assign the appropriate values to them.

The routine first increments the column number counter (global variable COLNUM) modulo 80. If the resultant column number is 1, the routine begins retrieving characters from the next card. This card may have already been read by the scanner in testing for a continuation card. If the card is not available, the routine tests if the end of file condition has been encountered previously, either by this routine or by the scanner in testing for a continuation card. If not, the next card is read into the program. Otherwise, the card which was read for the continuation test is copied into the variable from which the routine retrieves characters. If the end of file condition is encountered when reading a new card in this routine, the GETEND variable is set to 1 and a return with failure is made.

To retrieve a character, the routine tests if the current column is an ignored column as specified in the user's IGNORE COLUMNS definition. If this column is to be ignored, a branch is made to the beginning of the routine so the next character can be analyzed. If this column is a usable one, the character in that column and the column number are assigned to the appropriate arguments and a successful return is made to the scanner.

APPENDIX II. DETAILED USER INFORMATION

The three types of input to the preprocessor are information about the base language, macro definitions (optional), and a program written in the extended version of a programming language. The first type consists of a keyword override card and definitions given in a form similar to Backus-Naur Form. The definitions must be grouped together, and the override card may either precede or follow the definitions.

The override card contains the string 'OVERRIDE:' starting in column one. The keywords to be overridden follow the colon and are separated by commas. The first blank indicates the end of the keywords.

EXAMPLES:

OVERRIDE:IF,THEN,ELSE,ENDIF

overrides the keywords IF, THEN, ELSE, AND ENDIF

OVERRIDE:WHILE

overrides the keyword WHILE

All the definitions about the base language begin with the definition name enclosed in angle brackets and followed by ::=. The definition name and the remainder of the contents of each definition are described below.

UNIQUE LABEL -- The information needed to create unique labels is as follows:

1) Uniform beginning -- a sequence of one or more characters which conforms to the rules for writing labels in the

standard language but which is shorter than the maximum allowed length. The preprocessor will use this uniform beginning to begin any labels it creates.

2) Variable digits -- an integer specifying how many digits should be used to make the labels unique. The preprocessor will begin counting at 1 using these digits (filling with leading zeros) and will imbed them in any labels it creates.

3) Remainder -- a sequence of zero or more characters which conform to the rules for ending labels. This sequence should <u>not</u> include any character(s) used to separate a label from the statement with which it is associated.

The uniform beginning must immediately follow the = and at least one blank must separate the components of the definition.

EXAMPLE:

(for FORTRAN) ⟨UNIQUE LABEL⟩ ::=24 3

This example will create labels such as 24001 and 24002.

UNIQUE IDENTIFIER -- The information needed to create unique variable names is as follows:

1) Uniform beginning -- a sequence of one or more characters which conforms to the rules for naming identifiers in the standard language but which is shorter than the maximum allowed length. The preprocessor will use this uniform beginning to begin the names of any new identifiers it creates.

2) Variable digits -- an integer specifying how many digits should be used to make the identifiers unique. The preprocessor will begin counting at 1 using these digits

(filling with leading zeros) and will imbed the digits in any identifier names it creates.

3) Remainder -- a sequence of zero or more characters which conforms to the rules for ending identifier names.

The uniform beginning must immediately follow the = and at least one blank must separate the components of the definition.

EXAMPLE:

(for PL/I) <UNIQUE IDENTIFIER > ::=LAB 3 XY

This example will create identifiers such as LAB001XY and LAB002XY.

LITERAL -- The information the user must provide about literals is as follows:

1) Literal opener -- a one- or two-character symbol string which begins a literal.

2) Literal closer -- a one- or two-character symbol string which ends a literal.

3) Special sequence (optional) -- a two-character symbol string which does not terminate a literal but would cause an error in determining where a literal ends if no special handling were provided because the first character of this special sequence is the same as the single character which normally terminates a literal. An example of this is the string '' which may appear within a literal in some languages in which a single quote normally terminates a literal.

The literal opener must directly follow the = and at least one blank must separate the definition components.

EXAMPLE:

If ' is both the literal opener and literal closer, and '' is is a sequence requiring special handling (as in PL/I):

⟨LITERAL⟩ ::=' ' ''

COMMENT -- The information the user must provide about comments is as follows:

1) Comment opener -- a one- or two-character symbol string in single quotes which begins a comment; or a column number in which a comment begins; or a one- or two-character symbol string in single quotes which begins a comment followed by a + and the column number in which the symbol string must start to indicate a comment.
2) Comment closer -- a one- or two-character symbol string in single quotes which ends a comment; or a column number in which a comment ends; or a one- or two-character symbol string in single quotes which ends a comment followed by a + and the column number in which the symbol string must start to terminate a comment.

The comment opener must directly follow the = and at least one blank must separate the definition components.

EXAMPLE:

If a C must appear in column 1 to indicate a comment, and a comment ends in column 72 (as in FORTRAN):

⟨COMMENT⟩ ::='C' + 1 72

LANGLABEL -- The information the user must provide about labels on statements is as follows:

1) Label opening delimiter -- the column number of the statement in which the label must begin; or a series of column numbers within which the label must begin represented by two column numbers separated by a hyphen.

2) Label closing delimiter -- a one- or two-character symbol string enclosed in single quotes which separates a label from the statement with which it is associated; or a column number in which a label must end; or a series of column numbers within which a label must end represented by two column numbers separated by a hyphen.

3) Starting character set -- a set of characters enclosed in single quotes which may start a label,

4) Remaining character set -- a set of characters enclosed in single quotes which may be used in all other character positions of a label.

The opening delimiter must begin directly following the = and at least one blank must separate the components of the definition.

EXAMPLES:

(for FORTRAN)

⟨LANGLABEL⟩ ::=1-5 1-5 '0123456789' '0123456789'

(for PL/I)

⟨LANGLABEL⟩ ::=2-72 ':' 'ABCDEFGHIJKLMNOPQRSTUVWXYZ@#$'
'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789@#$_'

(for SNOBOL)

⟨LANGLABEL⟩ ::=1 1-72 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789._'

GREATER THAN, GREATER THAN OR EQUAL,

LESS THAN OR EQUAL, NOT EQUAL, NOT -- The operator for each logical operation should be enclosed in single quotes and directly follow the =.

EXAMPLES:

(for PL/I) ⟨NOT EQUAL⟩ ::=' ='

(for SNOBOL) ⟨LESS THAN⟩ ::='LT'

(for FORTRAN) ⟨NOT⟩ ::='.NOT.'

IGNORE COLUMNS -- The information the user must provide about columns to ignore in creating and scanning statements is as follows:

Column numbers -- the number of a column to be completely ignored; or a series of column numbers for columns to be ignored represented by two column numbers separated by a hyphen.

The definition item must immediately follow the =.

EXAMPLE:

(for SNOBOL, FORTRAN, and PL/I) ⟨IGNORE COLUMNS⟩ ::=73-80

END OF STATEMENT -- The information the user must provide about the end of statement condition is as follows:

End of statement indicator -- a one- or two-character symbol string in single quotes which indicates the end of a statement; or a column number which terminates a statement (either a regular statement or on a continuation card).

The indicator must immediately follow the =.

EXAMPLES:

(for SNOBOL) ⟨END OF STATEMENT⟩ ::=72

(for PL/I) ⟨END OF STATEMENT⟩ ::=';'

CONTINUATION -- The information the user must provide about continuation cards is as follows:

Continuation symbol -- a single character followed by the column number in which that symbol must appear to indicate a continuation card; or the string NONE to indicate that statements may cross card boundaries without specifically indicating continuation.

The continuation symbol or the word NONE must immediately follow the =. If a symbol is indicated, at least one blank must separate it from the column specification.

EXAMPLES:

(for SNOBOL) ⟨CONTINUATION⟩ ::=+ 1

(for PL/I) ⟨CONTINUATION⟩ ::=NONE

GO TO STATEMENT, IF THEN GO TO STATEMENT, ASSIGN MINUS STATEMENT, ASSIGN PLUS STATEMENT, DECLARATION STATEMENT -- In every statement definition, there are certain conventions which must be followed. At least one blank must appear between a parameter name and the items on either side of it. A parameter name may, however, begin in column 1 of the prototype, i.e., directly following the =. The parameter names must be spelled exactly as specified in the description of each definition.

The definition names of the statements, their function, and their parameters are as follows:

NULL STATEMENT -- a null statement.

Parameter names:

LAB -- label on the null statement

GO TO STATEMENT -- a branching statement.

Parameter names:

LAB -- label on the branching statement

DES -- label of the statement which is the destination of the branch

IF THEN GO TO STATEMENT -- a conditional statement in which the result of a true value is a branch.

Parameter names:

LAB -- label on the conditional statement

VAL1 - first value in the condition

RL -- relational operator in the condition

VAL2 - second value in the condition

DES -- label of the statement which is the destination of the branch

ASSIGN MINUS STATEMENT -- an assignment statement in which the expression being evaluated is a subtraction.

Parameter names:

LAB -- label on the assignment statement

VAR -- identifier to be assigned a value

VAL1 - minuend

VAL2 - subtrahend

ASSIGN PLUS STATEMENT -- an assignment statement in which the expression being evaluated is an addition.

Parameter names:

LAB -- label on the assignment statement

VAR -- identifier to be assigned a value

VAL1 - first addend

VAL2 - second addend

DECLARATION STATEMENT -- a declaration statement for one variable which is global in scope and is of type integer.

Parameter names:

VARNAME -- identifier to be declared.

Note: If declarations of identifiers are not required, a blank prototype must be provided.

The first column following the equal sign, column 1 of the prototype, is treated as column 1 of a card image. Therefore, if a label must appear in a particular position and the remainder of the statement must appear in a particular position, the definition must appear in the same form with enough blanks included to position items correctly. Since the user also defines the length of identifier names and labels created by the preprocessor, the user must be sure to allow enough space for these to fit if the parameter name is shorter than the item which will be created to replace it. In some cases, a user's variable will be used in the place of a parameter name. Consequently, if blank spaces are critical, it may be wise to use identifiers with the same number of characters in the names. Since the definition name takes up some space and the preprocessor requires an 80-column prototype, two cards will be needed for each definition. Even if the prototype provided by the user could fit on one card with the definition name, it is necessary to provide two cards, leaving the second one blank.

All variables introduced by the preprocessor are integers and have global scope. Consequently, declarations for these new variables should be placed wherever declarations for global variables are normally

placed in a program written in the base language. To allow the preprocessor to position these declarations correctly a special input statement must be included in the extended program containing only the string ADDED DECLARATIONS. When the preprocessor is ready to add declarations, it will replace this statement with the appropriate declarations. If declarations are not required and such has been specified in the declaration definition, the 'ADDED DECLARATIONS' statement should be omitted.

EXAMPLES:

(for FORTRAN) ⟨NULL STATEMENT⟩ ::=LAB CONTINUE

(for SNOBOL) ⟨GO TO STATEMENT⟩ ::=LAB :(DES)

(for PL/I) ⟨IF THEN GO TO STATEMENT⟩::= LAB :
IF VAL1 RL VAL2 THEN GO TO DES ;

(for PL/I) ⟨ASSIGN MINUS STATEMENT⟩ ::= LAB :
VAR = VAL1 - VAL2 ;

(for SNOBOL) ⟨ASSIGN PLUS STATEMENT⟩ ::=LAB
VAR = VAL1 + VAL2

(for FORTRAN) ⟨DECLARATION STATEMENT⟩ ::=LAB INTEGER VARNAME

Macro definitions have three parts, the macro name and parameters, the replacement text, and an indication of the end of the definition. The first card of a macro definition begins in column 1 with the string '⟨MACRO CALL⟩::=' followed immediately by the macro name and a list of parameter names enclosed in parentheses. The parameter names are separated by commas and no imbedded blanks are allowed. The next card starts with the string '⟨MACRO DEFINITION⟩ ::=' starting in column 1. The remainder of the card is left blank. The next card is the beginning

of the replacement text for the macro. The last card of the replacement text is followed by a card containing only the string 'END MACRO DEFINITION' starting in column 1.

The parameter names appearing in the replacement text will be replaced by the actual arguments of a macro call during code expansion by the preprocessor. Because of storage limitations, there can be no more than twenty components in a macro replacement text, where a component is defined to be a parameter name and following blanks or a string of characters appearing between parameter names.

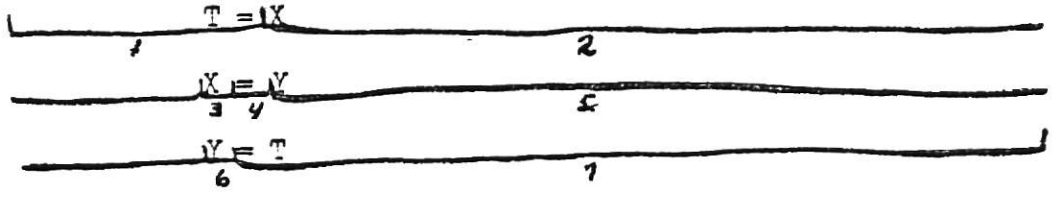
EXAMPLES:

```
⟨MACRO CALL⟩ ::=SWAP(X,Y)
⟨MACRO DEFINITION⟩ ::=
        T = X
        X = Y
        Y = T
    END MACRO DEFINITION
```

In the above example, SWAP is the macro name. X and Y are the formal parameter names. When the preprocessor expands the call SWAP(A,B), the following code will be inserted:

```
        T = A
        A = B
        B = T
```

This macro definition contains the following seven components:



If this macro were to be used in the extended version of SNOBOL, it could not be called with arguments longer than one character because the space before the equal sign would have to be retained. In FORTRAN the arguments could be one or two characters long.

EXAMPLE:

The call SWAP(A1,A2) would result in the following inserted code:

```
T = A1
A1= A2
A2= T
```

No argument may be of longer length than the corresponding parameter name and the blanks immediately following it. A call of this macro with either argument longer than two characters would cause an error during the creation of the code to be inserted.

The same macro could be defined as follows:

```
<MACRO CALL> ::=SWAP(X,Y)
      T = X
      X      = Y
      Y      = T
END MACRO DEFINITION
```

Using this definition, the SWAP macro may be called with arguments of longer length than two characters.

Since the objectives of the preprocessor include encouraging the use of structured programming techniques, the following guidelines are provided for the user to use in writing macro definitions. It is felt that by following these guidelines, a user will be more

likely to write macro definitions of constructs in line with the aims of structured programming.

1) Sequentially execute statements within the definition wherever possible. In other words, avoid using branches among statements within the definition.
2) Avoid branching out of the definition wherever possible. That is, do not specify branches to labels not within the macro definition.
3) Use extended language statements defined in the preprocessor if possible.

Parameter names must be unique and no parameter name may be identical to the beginning of another parameter name of the same macro. This is due to the scanning technique used by the preprocessor in decoding the macro replacement text into its components. For example, the names ABCD and ABCDE may not be used as parameter names in the same macro definition. The preprocessor will fail to recognize ABCDE as a parameter name in the replacement text because it will always assume the first four characters constitutes an occurrence of the parameter name ABCD.

The syntax of the input program is nearly identical to the syntax of the base language involved. Many of the input program statements will be copied directly into the output. Extended statements should follow the same syntax as normal statements, even though the constructs are unknown in the base language. For example, when processing an extended FORTRAN program, the preprocessor will expect statement labels to be entirely numeric and appear in columns 1-5, continuation

to be indicated by a special symbol in column 6, and Boolean expressions to be expressed using the FORTRAN relational operators.

The syntax of the extended constructs statements will be similar to the syntax of other statements in the base language. There are specific requirements, however, which must be followed.

1) A blank must follow each extended construct keyword or the keyword must appear at the end of a statement.

2) The conditions specified in extended construct statements must be specified in forms appropriate for the base language because these conditions will be copied directly into the expanded text.

3) The conditions in the WHILE and REPEAT UNTIL statements must be enclosed in parentheses.

4) The following pairs of extended construct keywords must appear in the same statement: REPEAT and UNTIL, IF and THEN, CASE and OF. All other extended construct keywords may not appear in a statement with a second keyword. The ending keywords must appear in statements with no other text. The DOGROUP and QUITLOOP keywords are the only construct-opening keywords which must appear alone.

The special statement to position new declarations should be placed wherever the user wishes to have the declarations added. All variables introduced by the preprocessor are integers with global scope. The user should keep this in mind when inserting the special statement. The statement should contain only the string ADDED DECLARATIONS and any end-of-statement specification normally required by the base language translator.

EXAMPLE: (PL/I)

```
TEST:  PROC OPTIONS (MAIN);

ADDED DECLARATIONS:

Y = 5;

DO I = 1 TO 3;

   Y = Y - 1;

ENDDO:

END TEST;
```

Restrictions

The user must keep the following restrictions in mind when using the preprocessor:

1) Only one QUITLOOP structure may be used within another loop.

2) The text following the THEN keyword and its succeeding blank must appear as a normal statement on a separate card. That is, if a specific number of columns must be blank at the beginning of a statement, that same number of columns must me blank in this situation.

3) Array elements may not be used as actual arguments in macro calls.

4) No continuation cards may be used in statements containing extended constructs.

APPENDIX III. JOB CONTROL LANGUAGE LISTINGS

```
//  EXEC SPITBOL
//*
//*     JOB STEP TO EXECUTE PREPROCESSOR FROM SOURCE DECK
//*
//TEMPCH DD DSN=&&TEMPCH,DCB=(BLKSIZE=800,LRECL=80,RECFM=FB),
//   SPACE=(TRK,(5,1)),UNIT=3330,DISP=(NEW,DELETE)
//TEMPCH2 DD DSN=&&TEMPCH2,DCB=(BLKSIZE=800,LRECL=80,RECFM=FB),
//   SPACE=(TRK,(5,1)),UNIT=3330,DISP=(NEW,DELETE)
//TEMOUT DD DSN=&&TEMOUT,DCB=(LRECL=132,BLKSIZE=132,RECFM=F),
//   UNIT=3330,SPACE=(TRK,(5,1)),DISP=(NEW,DELETE)
//TEMOUT2 DD DSN=&&TEMOUT2,DCB=(LRECL=132,BLKSIZE=132,RECFM=F),
//   UNIT=3330,SPACE=(TRK,(5,1)),DISP=(NEW,DELETE)
//FINOUT DD SYSOUT=A,DCB=(BLKSIZE=133,LRECL=133,RECFM=FA)
//FINPUNCH DD DSN=&&FINPCH,DCB=(BLKSIZE=800,LRECL=80,RECFM=FB),
//   DISP=(NEW,PASS),UNIT=3330,SPACE=(TRK,(5,1))
//SYSIN DD *
   ***    PREPROCESSOR SOURCE AND INPUT DATA
/*
```

JCL to execute preprocessor from source deck.

```
//LKED  EXEC  PGM=IEWL,PARM='LET,LIST,MAP'
//*
//*     JOB STEPS TO EXECUTE PREPROCESSOR FROM OBJECT DECK
//*
//*     FIRST STEP LINKAGE EDITS OBJECT MODULE
//*
//*     SECOND STEP EXECUTES TEMPORARY LOAD MODULE
//*
//SYSLMOD DD DSN=&&LOADMOD(PROG),DISP=(NEW,PASS),
//   UNIT=3330,SPACE=(TRK,(8,5,1))
//SYSUT1 DD UNIT=(SYSDA,SEP=SYSLMOD),DCB=BLKSIZE=1024,
//   SPACE=(1024,(200,20))
//SPITPROG DD DSN=SYS1.SPITBOL,DISP=SHR
//SYSLIN DD *
   ***    PREPROCESSOR OBJECT DECK
 INCLUDE SPITPROG(SPITPROG)
 ENTRY OSINT
/*
//GO   EXEC PGM=PROG,
//   PARM='L=16K,H=1000K,R=20K,C=100000,P=2000,T=120,D=10,N=58'
//STEPLIB DD DSN=&&LOADMOD,DISP=(OLD,DELETE)
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD SYSOUT=B,DCB=(BLKSIZE=800,LRECL=80,RECFM=FB,BUFNO=1)
//TEMPCH DD DSN=&&TEMPCH,DCB=(BLKSIZE=800,LRECL=80,RECFM=FB),
//   SPACE=(TRK,(5,1)),UNIT=3330,DISP=(NEW,DELETE)
//TEMPCH2 DD DSN=&&TEMPCH2,DCB=(BLKSIZE=800,LRECL=80,RECFM=FB),
//   SPACE=(TRK,(5,1)),UNIT=3330,DISP=(NEW,DELETE)
//TEMOUT DD DSN=&&TEMOUT,DCB=(LRECL=132,BLKSIZE=132,RECFM=F),
//   UNIT=3330,SPACE=(TRK,(5,1)),DISP=(NEW,DELETE)
//TEMOUT2 DD DSN=&&TEMOUT2,DCB=(LRECL=132,BLKSIZE=132,RECFM=F),
//   UNIT=3330,SPACE=(TRK,(5,1)),DISP=(NEW,DELETE)
//FINOUT DD SYSOUT=A,DCB=(BLKSIZE=133,LRECL=133,RECFM=FA)
//FINPUNCH DD DSN=&&FINPCH,DCB=(BLKSIZE=800,LRECL=80,RECFM=FB),
//   DISP=(NEW,PASS),UNIT=3330,SPACE=(TRK,(5,1))
//SYSIN DD *
   ***    PREPROCESSOR INPUT DATA
/*
```

JCL to execute preprocessor from object deck.

```
//   EXEC PGM=PREPROC,
//   PARM='L=16K,H=1000K,R=20K,C=100000,P=2000,T=120,D=10,N=58'
//*
//*  JOB STEP TO EXECUTE PREPROCESSOR FROM LOAD MODULE
//*
//STEPLIB DD DSN=DSP42.PREPROC.LOADMOD,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD SYSOUT=B,DCB=(BLKSIZE=800,LRECL=80,RECFM=FB,BUFNO=1)
//TEMPCH DD DSN=&&TEMPCH,DCB=(BLKSIZE=800,LRECL=80,RECFM=FB),
//   SPACE=(TRK,(5,1)),UNIT=3330,DISP=(NEW,DELETE)
//TEMPCH2 DD DSN=&&TEMPCH2,DCB=(BLKSIZE=800,LRECL=80,RECFM=FB),
//   SPACE=(TRK,(5,1)),UNIT=3330,DISP=(NEW,DELETE)
//TEMOUT DD DSN=&&TEMOUT,DCB=(LRECL=132,BLKSIZE=132,RECFM=F),
//   UNIT=3330,SPACE=(TRK,(5,1)),DISP=(NEW,DELETE)
//TEMOUT2 DD DSN=&&TEMOUT2,DCB=(LRECL=132,BLKSIZE=132,RECFM=F),
//   UNIT=3330,SPACE=(TRK,(5,1)),DISP=(NEW,DELETE)
//FINOUT DD SYSOUT=A,DCB=(BLKSIZE=133,LRECL=133,RECFM=FA)
//FINPUNCH DD DSN=&&FINPCH,DCB=(BLKSIZE=800,LRECL=80,RECFM=FB),
//   DISP=(NEW,PASS),UNIT=3330,SPACE=(TRK,(5,1))
//SYSIN DD *
   ***    PREPROCESSOR INPUT DATA
/*
```

JCL to execute preprocessor from load module.

APPENDIX IV. EXAMPLE BASE LANGUAGE SPECIFICATIONS

```
PL/I
<GREATER THAN>:::='>'
<NOT>:::='¬'
<NOT EQUAL>:::='¬='
<GREATER THAN OR EQUAL>:::='>='
<LESS THAN OR EQUAL>:::='<='
<NULL STATEMENT>::= LAB       : ;

<GO TO STATEMENT>::= LAB       : GO TO DES       ;

<IF THEN GO TO STATEMENT>::= LAB       : IF(VAL1      RL    VAL2      )THEN GO TO DE
S      ;
<DECLARATION STATEMENT>::= DCL VARNAME ;

<ASSIGN MINUS STATEMENT>::= LAB       : VAR      =VAL1      - VAL2     ;

<ASSIGN PLUS STATEMENT>::= LAB       : VAR       = VAL1     +VAL2     ;

<LITERAL>:::=' ' ''
<COMMENT>:::='/*' + 10 '*/'
<UNIQUE LABEL>:::=L 4
<UNIQUE IDENTIFIER>:::=ID 3 ST
<END OF STATEMENT>:::=';'
<CONTINUATION>:::=NONE
<IGNORE COLUMNS>:::=73-80
<LANGLABEL>:::=2-72 ':' 'ABCDEFGHIJKLMNOPQRSTUVWXYZ@#$'          'ABCDEFGHIJKLMNO
PQRSTUVWXYZ@#$_'
```

Sample language specifications for PL/I.

```
SNOBOL
<NULL STATEMENT>:::=LAB

<GO TO STATEMENT>:::=LAB                :(DES          )

<DECLARATION STATEMENT>:::=

<ASSIGN MINUS STATEMENT>:::=LAB            VAR       = VAL1       - VAL2

<ASSIGN PLUS STATEMENT>:::=LAB           VAR       = VAL1       + VAL2

<IF THEN GO TO STATEMENT>:::=LAB           RL(VAL1          ,VAL2          )  :S(DES
          )
<LITERAL>:::=' '
<COMMENT>:::='*' + 1 72
<UNIQUE LABEL>:::=LAB 6 NEW
<UNIQUE IDENTIFIER>:::=NEWID 3
<GREATER THAN>:::='GT'
<NOT>:::=' ¬'
<NOT EQUAL>:::='NE'
<GREATER THAN OR EQUAL>:::='GE'
<LESS THAN OR EQUAL>:::='LE'
<LANGLABEL>:::=1 1-72 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789._'
<CONTINUATION>:::=+ 1
<END OF STATEMENT>:::=72
<IGNORE COLUMNS>:::=73-80
```

Sample language specifications for SNOBOL.

```
FORTRAN
<IF THEN GO TO STATEMENT>::=LAB    IF(VAL1          RL     VAL2          ) GO TO DES

<DECLARATION STATEMENT>::=       INTEGER VARNAME

<NULL STATEMENT>::=LAB    CONTINUE

<GO TO STATEMENT>::=LAB    GO TO DES

<ASSIGN MINUS STATEMENT>::=LAB    VAR     =VAL1    - VAL2

<ASSIGN PLUS STATEMENT>::=LAB    VAR    = VAL1    +  VAL2

<GREATER THAN>::='.GT.'
<NOT>::='.NOT.'
<NOT EQUAL>::='.NE.'
<GREATER THAN OR EQUAL>::='.GE.'
<LESS THAN OR EQUAL>::='.LE.'
<LITERAL>::=' ' ''
<COMMENT>::='C' + 1 72
<UNIQUE LABEL>::=10 2 4
<UNIQUE IDENTIFIER>::=IJK 2
<CONTINUATION>::=*  6
<END OF STATEMENT>::=72
<IGNORE COLUMNS>::=73-80
<LANGLABEL>::=1-5 1-5 '0123456789' '0123456789'
```

Sample language specifications for FORTRAN.

APPENDIX V. PREPROCESSOR SOURCE CODE LISTING

```
      &STLIMIT = 150000                                                    PRE00010
*                                                                          PRE00020
*   ASSOCIATE OUTPUT VARIABLES WITH A PRINTER FILE, A TEMPORARY            PRE00030
*       'PRINT' FILE, AND A TEMPORARY 'PUNCH' FILE                         PRE00040
*                                                                          PRE00050
      OUTPUT(.FINOUT,'FINOUT',' ')                                         PRE00060
      OUTPUT(.OUTPUT,'TEMOUT')                                             PRE00070
      OUTPUT(.PUNCH,'TEMPCH')                                              PRE00080
*                                                                          PRE00090
*   INITIALIZE KEYWORDS                                                    PRE00100
*                                                                          PRE00110
      &ANCHOR = 0                                                          PRE00120
      &DUMP = 1                                                            PRE00130
      &TRIM = 0                                                            PRE00140
*                                                                          PRE00150
*   DEFINE FUNCTIONS                                                       PRE00160
*                                                                          PRE00170
      DEFINE('SCANCOMDEF(COMDEF)I')                                        PRE00180
      DEFINE('SCANIGNCOLDEF(IGNCOLDEF)')                                   PRE00190
      DEFINE('SCANLANLABDEF(LANLABDEF)SPEC')                               PRE00200
      DEFINE('SCANLITDEF(LITDEF)')                                         PRE00210
      DEFINE('SCANEOSDEF(EOSDEF)')                                         PRE00220
      DEFINE('SCANCONTDEF(CONTDEF)')                                       PRE00230
      DEFINE('GETCHAR(CHAR,COL)')                                          PRE00240
      DEFINE('SCAN()','BEGSCAN')                                           PRE00250
      DEFINE('INSERTCOM(COMSTR)I')                                         PRE00260
      DEFINE('PARSE()','STARTP')                                           PRE00270
      DEFINE('SCANDEFS(CARD)SELECT,STMTTYPE,UNITYPE,I,REST')               PRE00280
      DEFINE('SCANOPERDEF(TYPEDEF,OPERDEF)')                               PRE00290
      DEFINE('SCANNULLDEF(NULLDEF)I,J')                                    PRE00300
      DEFINE('SCANGOTODEF(GOTODEF)I,J')                                    PRE00310
      DEFINE('SCANIFDEF(IFDEF)I,J')                                        PRE00320
      DEFINE('SCANDCLDEF(DCLDEF)I,J')                                      PRE00330
      DEFINE('SCANMINUSDEF(MINUSDEF)I,J')                                  PRE00340
      DEFINE('SCANPLUSDEF(PLUSDEF)I,J')                                    PRE00350
      DEFINE('SCANLABELDEF(LABELDEF)')                                     PRE00360
      DEFINE('SCANVARDEF(VARDEF)')                                         PRE00370
      DEFINE('MACROSCAN(MACCALL)I,MACREPSTR,NAME,MACDEF,PLIST')            PRE00380
      DEFINE('CRMINUS(LAB,VAR,VAL1,VAL2)STMTLEN,I')                        PRE00390
      DEFINE('CRPLUS(LAB,VAR,VAL1,VAL2)STMTLEN,I')                         PRE00400
      DEFINE('CRNULL(LAB)STMTLEN,I')                                       PRE00410
      DEFINE('CRGOTO(LAB,DES)STMTLEN,I')                                   PRE00420
      DEFINE('CRIF(LAB,VAL1,RL,VAL2,DES)STMTLEN,I')                        PRE00430
      DEFINE('CRDCL(VARNAME)STMTLEN,I')                                    PRE00440
      DEFINE('UNILABEL()I')                                                PRE00450
      DEFINE('UNIVAR()I')                                                  PRE00460
      DEFINE('POP.SEM(NOARGS,A1,A2,A3,A4,A5,A6)I')                         PRE00470
      DEFINE('PUSH.SEM(NOARGS,A1,A2,A3,A4,A5,A6)I')                        PRE00480
```

```
        DEFINE('INSERTSTMT(STR,INSRT)TYPE,OUTSTR,CHARKTR,I')                    PRE00490
        DEFINE('INSERT(INSRT,STR)')                                             PRE00500
        DEFINE('ROUT0()')                                                       PRE00510
        DEFINE('ROUT01()SELMAC,STMTLEN,I,PARMLIST,PARM,MACSTR')                 PRE00520
        DEFINE('ROUT1()VAR1,VAR2')                                              PRE00530
        DEFINE('ROUT2()VAR0,VAR1,IND')                                          PRE00540
        DEFINE('ROUT3()IND,BEGIN,END,INC,VAR1,VAR2')                            PRE00550
        DEFINE('ROUT4()')                                                       PRE00560
        DEFINE('ROUT5()BOOL,VAR1')                                              PRE00570
        DEFINE('ROUT6()BOOL,VAR1')                                              PRE00580
        DEFINE('ROUT7()BOOL,VAR1,VAR2')                                         PRE00590
        DEFINE('ROUT8()VAR1')                                                   PRE00600
        DEFINE('ROUT9()VAR0,VAR1,IND')                                          PRE00610
        DEFINE('ROUT10()VAR1')                                                  PRE00620
        DEFINE('ROUT11()VAR1,VAR2,QUITLAB,QUITFLAG')                            PRE00630
        DEFINE('ROUT12()BOOL,VAR1,QUITLAB,QUITFLAG')                            PRE00640
      DEFINE('ROUT13()VAR1,VAR2,IND,INC,QUITLAB,QUITFLAG,VAR3,VAR4')            PRE00650
        DEFINE('ROUT14()')                                                      PRE00660
        DEFINE('ROUT15()VAR0,VAR1,VAR3,IND')                                    PRE00670
*                                                                               PRE00680
*   DECLARE ARRAYS                                                              PRE00690
*                                                                               PRE00700
        CONTPAT = ARRAY('1:2')                                                  PRE00710
        EOSPAT = ARRAY('0:1')                                                   PRE00720
        LITPAT = ARRAY('1:3')                                                   PRE00730
        LANLABPAT = ARRAY('1:9')                                                PRE00740
        IGNPAT = ARRAY('0:2')                                                   PRE00750
        COMPAT = ARRAY('1:6')                                                   PRE00760
        NULLPAT = ARRAY('0:3')                                                  PRE00770
        GOTOPAT = ARRAY('0:5')                                                  PRE00780
        IFPAT = ARRAY('0:11')                                                   PRE00790
        MINUSPAT = ARRAY('0:9')                                                 PRE00800
        PLUSPAT = ARRAY('0:9')                                                  PRE00810
        DCLPAT = ARRAY('0:3')                                                   PRE00820
        LABELPAT = ARRAY('1:3')                                                 PRE00830
        VARPAT = ARRAY('1:3')                                                   PRE00840
        MACROTAB = TABLE()                                                      PRE00850
        SEMSTACK = ARRAY('1:10')                                                PRE00860
        VARLIST = ARRAY('1:20')                                                 PRE00870
        MPAT = ARRAY('0:20')                                                    PRE00880
        MPARM = ARRAY('0:6')                                                    PRE00890
        CPARM = ARRAY('1:6')                                                    PRE00900
*                                                                               PRE00910
*   INITIALIZE STRINGS AND PATTERNS                                             PRE00920
*                                                                               PRE00930
        MACNAMEPAT = DUPL(' ',81)                                               PRE00940
        LABDELS = ' ' | ':' | '(' | '.'                                         PRE00950
        KEYCASE = 'CASE'                                                        PRE00960
```

```
          KEYDO = 'DO'                                                          PRE00970
          KEYDOGROUP = 'DOGROUP'                                                PRE00980
          KEYIF = 'IF'                                                          PRE00990
          KEYREPEAT = 'REPEAT'                                                  PRE01000
          KEYWHILE = 'WHILE'                                                    PRE01010
          KEYELSE = 'ELSE'                                                      PRE01020
          KEYQUITLOOP = 'QUITLOOP'                                              PRE01030
          KEYENDCASE = 'ENDCASE'                                                PRE01040
          KEYENDIF = 'ENDIF'                                                    PRE01050
          KEYENDWHILE = 'ENDWHILE'                                              PRE01060
          KEYENDREPEAT = 'ENDREPEAT'                                            PRE01070
          KEYENDDO = 'ENDDO'                                                    PRE01080
          KEYENDDOGROUP = 'ENDDOGROUP'                                          PRE01090
          KEYTHEN = 'THEN'                                                      PRE01100
          KEYUNTIL = 'UNTIL'                                                    PRE01110
          KEYWORDS = KEYCASE | KEYDO | KEYDOGROUP |                             PRE01120
+                    KEYIF | KEYREPEAT | KEYWHILE | KEYELSE |                   PRE01130
+                    KEYQUITLOOP | KEYENDCASE | KEYENDIF |                      PRE01140
+                    KEYENDWHILE | KEYENDREPEAT | KEYENDDO |                    PRE01150
+                    KEYENDDOGROUP                                               PRE01160
          KEYWORD2 = KEYTHEN | KEYUNTIL                                         PRE01170
          NULL =                                                                PRE01180
          CURSTMT =                                                             PRE01190
          WORKING =                                                             PRE01200
          TOKEN =                                                               PRE01210
          NEXTCARD =                                                            PRE01220
          LANGPAT = 'FORTRAN' | 'PL/I' | 'SNOBOL'                               PRE01230
*                                                                               PRE01240
*     INITIALIZE SCALARS AND STRINGS TO BE USED AS SCALARS                      PRE01250
*                                                                               PRE01260
          CCLNUM = 80                                                           PRE01270
          VARNUM = '1'                                                          PRE01280
          NEXTVAR = 1                                                           PRE01290
          LABELNUM = '1'                                                        PRE01300
          TOP = 0                                                               PRE01310
          ENDFILE = 0                                                           PRE01320
          STARTCOL = 0                                                          PRE01330
          STRINGSEP = 0                                                         PRE01340
          GETEND = 0                                                            PRE01350
          COMMENTONLY = 0                                                       PRE01360
          STMTEOS = 1                                                           PRE01370
          EOSNEXT = 0                                                           PRE01380
          MACROSPRES = 0                                                        PRE01390
          INASTRUCT = 0                                                         PRE01400
          NO.OF.CASES = 0                                                       PRE01410
          KTRO = 0                                                              PRE01420
*                                                                               PRE01421
*********************************************************************************PRE01422
```

```
*                                                                                PRE01423
*     DETERMINE LANGUAGE BEING PROCESSED --                                      PRE01424
*         FORTRAN, PL/I, OR SNOBOL                                               PRE01425
*                                                                                PRE01426
*********************************************************************************PRE01427
*                                                                                PRE01430
*     READ THE LANGUAGE TYPE; BRANCH IF NOT VALID                                PRE01440
*                                                                                PRE01450
          LANGUAGE = TRIM(INPUT)                                                 PRE01460
          LANGUAGE LANGPAT                                  :F(LANGERR)          PRE01470
*                                                                                PRE01471
*********************************************************************************PRE01472
*                                                                                PRE01473
*     DECODE DEFINITIONS                                                         PRE01474
*                                                                                PRE01475
*********************************************************************************PRE01476
*                                                                                PRE01480
*     READ A CARD, DETERMINE IF IT IS AN OVERRIDE CARD, THE FIRST                PRE01490
*         DEFINITION, A MACRO CALLING FORM, OR THE FIRST CARD OF THE             PRE01500
*         INPUT PROGRAM                                                          PRE01510
*                                                                                PRE01520
INLOOP    CARD = INPUT                                                          PRE01530
*                                                                                PRE01540
*     DECODE AN OVERRIDE CARD; BRANCH TO READ THE NEXT CARD                      PRE01550
*                                                                                PRE01560
          CARD POS(0) 'OVERRIDE:' =                         :F(NOTOVERRIDE)      PRE01570
NEXTOVERKEY CARD BREAK(',') . OVERKEY ',' =                 :F(LASTOVERKEY)      PRE01580
          FINOUT = OVERKEY ' KEYWORD OVERRIDDEN.'                                PRE01590
          $('KEY' OVERKEY) = DUPL('X',81)                   :(NEXTOVERKEY)       PRE01600
LASTOVERKEY CARD BREAK(' ') . OVERKEY =                     :S(KEYSDONE)         PRE01610
          CARD REM . OVERKEY =                                                   PRE01620
KEYSDONE  $('KEY' OVERKEY) = DUPL('X',81)                                        PRE01630
          FINOUT = OVERKEY ' KEYWORD OVERRIDDEN.'                                PRE01640
          FINOUT = '    '                                                        PRE01650
          FINOUT = '    '                                   :(INLOOP)            PRE01660
NOTOVERRIDE CARD POS(0) '<MACRO CALL>::='                   :F(NOTMACRO)         PRE01670
*                                                                                PRE01680
*     SET THE MACROS PRESENT FLAG TO 1; CALL THE MACRO SCANNING ROUTINE;         PRE01690
*         BRANCH TO READ THE NEXT CARD                                           PRE01700
*                                                                                PRE01710
          MACROSPRES = 1                                                         PRE01720
          MACROSCAN(TRIM(CARD))                             :(INLOOP)            PRE01730
NOTMACRO  CARD POS(0) '<'                                   :F(NOTDEFNS)         PRE01740
*                                                                                PRE01750
*     CALL THE DEFINITION SCANNING ROUTINE; SET THE SEMANTIC                     PRE01760
*         DELIMITER STRING; BRANCH TO READ ANOTHER CARD                          PRE01770
*                                                                                PRE01780
          SCANDEFS(CARD)                                                         PRE01790
```

```
            SEMDEL = ' '                                                    PRE01800
            EOSPAT<0> 'COL'                          :S(INLOOP)             PRE01810
            SEMDEL = ' '    SUBSTR(EOSPAT<1>,1,1)    :(INLOOP)              PRE01820
NOTDEFNS    NEXTCARD = CARD                                                 PRE01830
*                                                                           PRE01840
*************************************************************************PRE01841
*                                                                           PRE01842
*     PARSE THE INPUT PROGRAM ON PARSE NUMBER 1                             PRE01850
*                                                                           PRE01851
*************************************************************************PRE01852
*                                                                           PRE01860
            P.NUMBER = 1                                                    PRE01870
            PARSE()                                                         PRE01880
            WORKING POS(0) ' PROG' RPOS(0)           :F(ENDPARSEERR)        PRE01890
            FINOUT = 'FIRST PARSE OF PROGRAM WAS SUCCESSFUL.'               PRE01900
            FINOUT = '    '                                                 PRE01910
            FINOUT = '    '                                                 PRE01920
*                                                                           PRE01930
*     REWIND THE TEMPORARY PUNCH FILE; ASSOCIATE AN INPUT VARIABLE          PRE01940
*          WITH THAT FILE AND OUTPUT VARIABLES WITH A SECOND TEMPORARY      PRE01950
*          'PRINT' FILE AND A SECOND TEMPORARY 'PUNCH' FILE                 PRE01960
*                                                                           PRE01970
            REWIND('TEMPCH')                                                PRE01980
            INPUT(.INPUT,'TEMPCH')                                          PRE01990
            OUTPUT(.OUTPUT,'TEMOUT2')                                       PRE02000
            OUTPUT(.PUNCH,'TEMPCH2')                                        PRE02010
*                                                                           PRE02020
*     BRANCH IF NO SECOND PARSE IS REQUIRED (NO MACROS WERE PROVIDED)       PRE02030
*                                                                           PRE02040
            EQ(MACROSPRES,0)                         :S(NOSECPARSE)         PRE02050
*                                                                           PRE02060
*     REINITIALIZE STRINGS AND SCALARS FOR SECOND PARSE                     PRE02070
*                                                                           PRE02080
            CURSTMT =                                                       PRE02090
            WORKING =                                                       PRE02100
            TOKEN =                                                         PRE02110
            CCLNUM = 80                                                     PRE02120
            TOP = 0                                                         PRE02130
            EOSNEXT = 0                                                     PRE02140
            ENDFILE = 0                                                     PRE02150
            GETEND = 0                                                      PRE02160
            COMMENTONLY = 0                                                 PRE02170
            STMTEOS = 1                                                     PRE02180
            STARTCOL = 0                                                    PRE02190
            STRINGSEP = 0                                                   PRE02200
            INASTRUCT = 0                                                   PRE02210
            NO.OF.CASES = 0                                                 PRE02220
            KTRU = 0                                                        PRE02230
```

```
*                                                                                    PRE02240
*************************************************************************************PRE02241
*                                                                                    PRE02242
*    PARSE THE RESULTS OF THE FIRST PARSE ON PARSE NUMBER 2                          PRE02250
*                                                                                    PRE02251
*************************************************************************************PRE02252
*                                                                                    PRE02260
            P.NUMBER = 2                                                            PRE02270
            NEXTCARD = INPUT                                                        PRE02280
            PARSE()                                                                 PRE02290
            WORKING POS(0) ' PROG' RPOS(0)                    :F(END2ERR)           PRE02300
            FINOUT = 'SECOND PARSE OF PROGRAM WAS SUCCESSFUL.'                      PRE02310
            FINOUT = '    '                                                         PRE02320
            FINOUT = '    '                                   :(ADDDCLS)            PRE02330
NOSECPARSE  FINOUT = 'SECOND PARSE OF PROGRAM NOT REQUIRED.'                        PRE02340
            FINOUT = '    '                                                         PRE02350
            FINOUT = '    '                                                         PRE02360
*                                                                                   PRE02370
*    REWIND THE TEMPORARY PRINT FILE; ASSOCIATE AN INPUT VARIABLE WITH              PRE02380
*        THAT FILE                                                                  PRE02390
*                                                                                   PRE02400
            REWIND('TEMOUT')                                                        PRE02410
            INPUT(.INTEMP,'TEMOUT')                                                 PRE02420
*                                                                                   PRE02430
*    COPY THE PREVIOUS TEMPORARY 'PRINT' FILE TO A SECOND                           PRE02440
*        TEMPORARY 'PRINT' FILE                                                     PRE02450
*                                                                                   PRE02460
COPYTEMP1   CARD = INTEMP                             :F(COPYTEMP2)                 PRE02470
            OUTPUT = CARD                             :(COPYTEMP1)                  PRE02480
*                                                                                   PRE02490
*    COPY THE PREVIOUS TEMPORARY PUNCH FILE TO A SECOND TEMPORARY                   PRE02500
*        'PUNCH' FILE                                                               PRE02510
*                                                                                   PRE02520
COPYTEMP2   CARD = INPUT                              :F(ADDDCLS)                   PRE02530
            PUNCH = CARD                              :(COPYTEMP2)                  PRE02540
*                                                                                   PRE02550
*    REWIND TEMPORARY FILES AND ASSOCIATE INPUT AND OUTPUT                          PRE02560
*        VARIABLES WITH TEMPORARY FILES                                             PRE02570
*                                                                                   PRE02580
ADDDCLS     REWIND('TEMPCH2')                                                       PRE02590
            REWIND('TEMOUT2')                                                       PRE02600
            REWIND('TEMPCH')                                                        PRE02610
            REWIND('TEMOUT')                                                        PRE02620
            INPUT(.INCARD,'TEMPCH2')                                                PRE02630
            INPUT(.INLINE,'TEMOUT2')                                                PRE02640
            OUTPUT(.PUNCH,'TEMPCH')                                                 PRE02650
            OUTPUT(.OUTPUT,'TEMOUT')                                                PRE02660
*                                                                                   PRE02670
```

```
*****************************************************************PRE02671
*                                                                PRE02672
*     ADD DECLARATIONS FOR CREATED VARIABLES IF A NON-BLANK DECLARATION  PRE02680
*          STATEMENT PROTOTYPE WAS PROVIDED                      PRE02690
*                                                                PRE02700
*****************************************************************PRE02701
*                                                                PRE02702
          EQ(DCLPAT<0>,1)                          :S(COPYRESTFILES) PRE02710
*                                                                PRE02720
*     READ A CARD AND ITS CORRESPONDING OUTPUT LINE              PRE02730
*                                                                PRE02740
COPY3     CARD = INCARD                            :F(NOADDDECSERR)  PRE02750
          LINE = INLINE                                          PRE02760
*                                                                PRE02770
*     IF THE CARD IS NOT THE ADD DECLARATIONS INDICATOR, COPY THIS PRE02780
*          CARD AND OUTPUT LINE AND BRANCH TO READ MORE;         PRE02790
*          IF NO ADD DECLARATIONS INDICATOR IF FOUND, TAKE THE   PRE02800
*          ERROR EXIT                                            PRE02810
*                                                                PRE02820
          CARD 'ADDED DECLARATIONS'                :S(ADDDECS)   PRE02830
          PUNCH = CARD                                           PRE02840
          OUTPUT = LINE                            :(COPY3)      PRE02850
*                                                                PRE02860
*     DECREMENT NEXTVAR, BRANCH TO COPY THE REMAINDER OF THE     PRE02870
*          FILES IF THERE ARE NO VARIABLES FOR WHICH TO ADD      PRE02880
*          DECLARATIONS                                          PRE02890
*                                                                PRE02900
ADDDECS   NEXTVAR = NEXTVAR - 1                                  PRE02910
          EQ(NEXTVAR,0)                            :S(COPYRESTFILES) PRE02920
*                                                                PRE02930
*     CREATE A STRING OF A DECLARATION STATEMENT; INSERT A COMMENT PRE02940
*          ABOUT ADDING A DECLARATION; INSERT THE DECLARATION;   PRE02950
*          IF THERE ARE MORE NEW VARIABLES, DECREMENT NEXTVAR AND PRE02960
*          BRANCH TO ADD THE NEXT DECLARATION                    PRE02970
*                                                                PRE02980
ADDNEXTDEC STRING = CRDCL(VARLIST<NEXTVAR>)                      PRE02990
          INSERTCOM('THE FOLLOWING IS AN ADDED DECLARATION')     PRE03000
          INSERT(STRING)                                         PRE03010
          NEXTVAR = GT(NEXTVAR,1) NEXTVAR - 1      :S(ADDNEXTDEC)  PRE03020
*                                                                PRE03030
*     COPY THE REMAINDER OF THE FILES                            PRE03040
*                                                                PRE03050
COPYRESTFILES PUNCH = INCARD                       :F(CKSNOBDL)  PRE03060
          OUTPUT = INLINE                          :(COPYRESTFILES) PRE03070
*                                                                PRE03080
*     REWIND THE TEMPORARY FILES AND ASSOCIATE INPUT VARIABLES   PRE03090
*          WITH TEMPORARY FILES AND OUTPUT VARIABLES WITH        PRE03100
*          REAL OUTPUT FILES                                     PRE03110
```

```
*                                                                      PRE03120
CKSNOBOL    REWIND('TEMPCH')                                           PRE03130
            REWIND('TEMOUT')                                           PRE03140
            INPUT(.INCARD,'TEMPCH')                                    PRE03150
            INPUT(.INLINE,'TEMOUT')                                    PRE03160
            OUTPUT(.PUNCH,'FINPUNCH')                                  PRE03170
            OUTPUT(.OUTPUT,'FINOUT',' ')                               PRE03180
*                                                                      PRE03190
*     BRANCH IF THE LANGUAGE BEING PROCESSED IS NOT SNOBOL             PRE03200
*                                                                      PRE03210
            LANGUAGE 'SNOBOL'                          :F(NOTSNOBOL)   PRE03220
*                                                                      PRE03230
*********************************************************************PRE03231
*                                                                      PRE03232
*     POST PROCESSING FOR SNOBOL:                                      PRE03233
*                                                                      PRE03234
*     SCAN THE FILES FOR CONDITIONALS USING ¬( FOLLOWED BY             PRE03240
*         BLANKS AND ,( -- THESE WERE ADDED BY THE PREPROCESSOR;       PRE03250
*         DELETE THE INTERVENING BLANKS AND THE COMMA TO MAKE THE      PRE03260
*         STATEMENT A LEGAL SNOBOL STATEMENT                           PRE03270
*                                                                      PRE03280
*********************************************************************PRE03281
*                                                                      PRE03282
SNOBOLTEST  CARD = INCARD                              :F(COMPLETE)    PRE03290
            LINE = INLINE                                              PRE03300
            CARD '¬(' SPAN(' ') ',(' = '¬(('                           PRE03310
            LINE '¬(' SPAN(' ') ',(' = '¬(('                           PRE03320
            PUNCH = CARD                                               PRE03330
            OUTPUT = LINE                              :(SNOBOLTEST)   PRE03340
*                                                                      PRE03350
*     BRANCH IF THE LANGUAGE BEING PROCESSED IS NOT FORTRAN            PRE03360
*                                                                      PRE03370
NOTSNOBOL   LANGUAGE 'FORTRAN'                         :F(NOTFORTRAN)  PRE03380
*                                                                      PRE03390
*********************************************************************PRE03391
*                                                                      PRE03392
*     POST PROCESSING FOR FORTRAN:                                     PRE03393
*                                                                      PRE03394
*     SCAN THE FILES FOR SEQUENCES OF A COMMENT, A CONTINUE, AND A     PRE03400
*         FORMAT STATEMETNT; RETURN THE STATEMENT NUMBER FROM THE      PRE03410
*         CONTINUE (INDICATION IS IN THE COMMENT THAT THE STATEMENT    PRE03420
*         NUMBER WAS TAKEN FROM THE FORMAT AND PLACED ON THE CONTINUE) PRE03430
*         TO THE FORMAT STATEMENT AND DELETE THE CONTINUE STATEMENT     PRE03440
*                                                                      PRE03450
*********************************************************************PRE03451
*                                                                      PRE03452
FORTTEST    CARD1 = INCARD                             :F(COMPLETE)    PRE03460
            LINE1 = INLINE                                             PRE03470
```

```
FORTTEST2   CARD1 POS(0) 'C' BREAK(' ') . STMT.NUMB      :F(NOFORTCOM)       PRE03480
            CARD2 = INCARD                                                   PRE03490
            LINE2 = INLINE                                                   PRE03500
            CARD2 POS(6) 'CONTINUE'                      :F(NOFORTCONT)      PRE03510
            CARD3 = INCARD                                                   PRE03520
            LINE3 = INLINE                                                   PRE03530
TESTCONTINUE CARD3 POS(0) '      CONTINUE'               :F(THISFORM)        PRE03540
            CARD3 = INCARD                                                   PRE03550
            LINE3 = INLINE                               :(TESTCONTINUE)     PRE03560
THISFORM    CARD3 POS(0) '       ' ARB . REP 'FORMAT' =                      PRE03570
+                 DUPL(' ',5 - SIZE(STMT.NUMB)) STMT.NUMB ' ' REP            PRE03580
+                 'FORMAT'                               :F(NOFORTFORM)      PRE03590
            LINE3 POS(0) '       ' = DUPL(' ',5 - SIZE(STMT.NUMB))           PRE03600
+                 STMT.NUMB ' '                                              PRE03610
            PUNCH = CARD3                                                    PRE03620
            OUTPUT = LINE3                               :(FORTTEST)         PRE03630
NOFORTCOM   PUNCH = CARD1                                                    PRE03640
            OUTPUT = LINE1                               :(FORTTEST)         PRE03650
NOFORTCONT  PUNCH = CARD1                                                    PRE03660
            CARD1 = CARD2                                                    PRE03670
            OUTPUT = LINE1                                                   PRE03680
            LINE1 = LINE2                                :(FORTTEST2)        PRE03690
NOFORTFORM  PUNCH = CARD1                                                    PRE03700
            PUNCH = CARD2                                                    PRE03710
            CARD1 = CARD3                                                    PRE03720
            OUTPUT = LINE1                                                   PRE03730
            OUTPUT = LINE2                                                   PRE03740
            LINE1 = LINE3                                :(FORTTEST2)        PRE03750
*                                                                            PRE03760
*     BRANCH IF THE LANGUAGE BEING PROCESSED IS NOT PL/I                     PRE03770
*                                                                            PRE03780
NOTFORTRAN LANGUAGE 'PL/I'                               :F(NOTPLI)          PRE03790
*                                                                            PRE03800
****************************************************************************PRE03801
*                                                                            PRE03802
*     POST PROCESSING FOR PL/I:                                              PRE03803
*                                                                            PRE03804
*     SCAN THE BEGINNING OF THE FILES FOR EITHER ONE OR TWO                  PRE03810
*          COMMENTS FOLLOWED BY THE SAME NUMBER OF NULL STATEMENTS AND       PRE03820
*          A PROCEDURE STATEMENT; RETURN THE LABEL WHICH WAS ORIGINALLY      PRE03830
*          TAKEN OFF THE PROCEDURE STATEMENT TO THE PROCEDURE STATEMENT      PRE03840
*          AND DELETE THE NULL STATEMENTS; COPY THE REMAINDER                PRE03850
*          OF THE FILES                                                      PRE03860
*                                                                            PRE03870
****************************************************************************PRE03871
*                                                                            PRE03872
            CARD1 = INCARD                                                   PRE03880
            LINE1 = INLINE                                                   PRE03890
```

```
            CARD1  POS(0) SPAN(' ') '/*' BREAK(' ') . STMT.NUMB                PRE03900
            CARD2  = INCARD                                                    PRE03910
            LINE2  = INLINE                                                    PRE03920
            CARD2  POS(0) SPAN(' ') '/*'                    :F(THISPROC)       PRE03930
            CARD2  = INCARD                                                    PRE03940
            LINE2  = INLINE                                                    PRE03950
            CARD2  = INLINE                                                    PRE03960
            LINE2  = INLINE                                                    PRE03970
THISPROC    CARD2  = INCARD                                                    PRE03980
            LINE2  = INLINE                                                    PRE03990
            CARD2  POS(0) SPAN(' ') . REP 'PROC' =                             PRE04000
+                  DUPL(' ',SIZE(REP) - SIZE(STMT.NUMB))                       PRE04010
+                  STMT.NUMB 'PROC'                                            PRE04020
            LINE2  POS(0) SPAN(' ') . REP 'PROC' = DUPL(' ',SIZE(REP) -        PRE04030
+                  SIZE(STMT.NUMB)) STMT.NUMB 'PROC'                           PRE04040
            PUNCH  = CARD1                                                     PRE04050
            PUNCH  = CARD2                                                     PRE04060
            OUTPUT = LINE1                                                     PRE04070
            OUTPUT = LINE2                                                     PRE04080
COPYRESTPLI PUNCH  = INCARD                     :F(COMPLETE)                   PRE04090
            OUTPUT = INLINE                     :(COPYRESTPLI)                 PRE04100
*                                                                              PRE04110
*******************************************************************************PRE04111
*                                                                              PRE04112
*     THE LANGUAGE WAS NOT FORTRAN OR SNOBOL OR PL/I, SO SIMPLY                PRE04120
*        COPY THE FILES                                                        PRE04130
*                                                                              PRE04140
*******************************************************************************PRE04141
*                                                                              PRE04142
NOTPLI      PUNCH  = INCARD                     :F(COMPLETE)                   PRE04150
            OUTPUT = INLINE                     :(NOTFORTRAN)                  PRE04160
*                                                                              PRE04170
*     OUTPUT A SUCCESSFUL COMPLETION MESSAGE AND TERMINATE                     PRE04180
*                                                                              PRE04190
COMPLETE    FINOUT = 'PREPROCESSING COMPLETED SUCCESSFULLY.'                   PRE04200
+                                               :(END)                         PRE04210
*                                                                              PRE04220
*******************************************************************************PRE04221
*                                                                              PRE04222
*     SECTION TO PRODUCE OUTPUT AFTER ERROR MESSAGES HAVE BEEN                 PRE04230
*        PRINTED FOR ERRORS DURING PROCESSING OF THE INPUT PROGRAM             PRE04240
*                                                                              PRE04250
*     COPY OUTPUT PRODUCED UP TO THE ERROR WHICH HAS BEEN ENCOUNTERED;         PRE04260
*        THE FILE OUTPUT DEPENDS ON THE PARSE NUMBER DURING WHICH              PRE04270
*        THE ERROR OCCURRED                                                    PRE04280
*                                                                              PRE04290
*******************************************************************************PRE04291
*                                                                              PRE04292
```

```
COPYTEMPOUT EQ(P.NUMBER,2)                                    :S(SECONDPARSE)    PRE04300
*                                                                                PRE04310
*     REWIND THE TEMPORARY PUNCH FILE; ASSOCIATE AN INPUT VARIABLE               PRE04320
*        WITH THAT FILE                                                          PRE04330
*                                                                                PRE04340
            REWIND('TEMPCH')                                                    PRE04350
            INPUT(.INTEMP,'TEMPCH')                           :(OUTPUTASSOC)    PRE04360
*                                                                               PRE04370
*     REWIND THE SECOND TEMPORARY 'PUNCH' FILE; ASSOCIATE AN INPUT              PRE04380
*        VARIABLE WITH THAT FILE                                                 PRE04390
*                                                                                PRE04400
SECONDPARSE REWIND('TEMPCH2')                                                   PRE04410
            INPUT(.INTEMP,'TEMPCH2')                                            PRE04420
*                                                                               PRE04430
*     ASSOCIATE AN OUTPUT VARIABLE WITH THE REAL PRINTER FILE                   PRE04440
*                                                                               PRE04450
OUTPUTASSOC OUTPUT(.OUTPUT,'FINOUT',' ')                                        PRE04460
            OUTPUT = '    '                                                     PRE04470
            OUTPUT = '    '                                                     PRE04480
            OUTPUT = "BECAUSE AN ERROR HAS BEEN ENCOUNTERED,"                   PRE04490
+                    " THE CONTENTS OF THE TEMPORARY FILE HOLDING THE "         PRE04500
+                    "'PUNCHED' OUTPUT PRODUCED SO FAR FOLLOWS:"                PRE04510
            OUTPUT = '    '                                                     PRE04520
            OUTPUT = '    '                                                     PRE04530
*                                                                               PRE04540
*     PRINT THE OUTPUT PRODUCED SO FAR AND BRANCH TO END                        PRE04550
*                                                                               PRE04560
TEMCOPY     CARD = INTEMP                                     :F(ENDTEMCOPY)    PRE04570
            OUTPUT = CARD                                     :(TEMCOPY)        PRE04580
ENDTEMCOPY  OUTPUT = '    '                                                     PRE04590
            OUTPUT = '    '                                                     PRE04600
            OUTPUT = 'PREPROCESSING COMPLETED UNSUCCESSFULLY.' :(END)           PRE04610
*                                                                               PRE04620
*********************************************************************************PRE04621
*                                                                               PRE04622
*     EXITS FOR ERRORS WHICH OCCURRED IN THE DRIVER                             PRE04630
*                                                                               PRE04640
*********************************************************************************PRE04641
*                                                                               PRE04642
LANGERR     FINOUT = 'AN INVALID LANGUAGE WAS SPECIFIED'                        PRE04650
            FINOUT = 'LANGUAGE = ' LANGUAGE                                     PRE04660
            FINOUT = 'ABNORMAL TERMINATION'                   :(END)            PRE04670
ENDPARSERR  FINOUT = 'THE FIRST PARSE OF THE INPUT PROGRAM TERMINATED '         PRE04680
+                    'ABNORMALLY'                                               PRE04690
            FINOUT = 'WORKING = ' WORKING                     :(COPYTEMPOUT)    PRE04700
END2ERR     FINOUT = 'THE SECOND PARSE OF THE INPUT PROGRAM TERMINATED '        PRE04710
+                    'ABNORMALLY'                                               PRE04720
            FINOUT = 'WORKING = ' WORKING                     :(COPYTEMPOUT)    PRE04730
```

```
NOADDDECSERR FINOUT = 'NO INDICATION WAS FOUND OF WHERE TO ADD '          PRE04740
+                     'DECLARATIONS FOR CREATED VARIABLES, BUT'           PRE04750
              FINOUT = 'A NON-BLANK DECLARATION STATEMENT PROTOTYPE '      PRE04760
+                     'WAS PROVIDED.'                    :(COPYTEMPOUT)   PRE04770
*                                                                         PRE04780
*********************************************************************** PRE04781
*                                                                         PRE04782
*     EXITS FOR ERRORS WHICH OCCURRED IN SCANNING DEFINITIONS             PRE04790
*                                                                         PRE04800
*********************************************************************** PRE04801
*                                                                         PRE04802
NULLPARMERR FINOUT = 'NOT ALL THE REQUIRED PARAMETERS WERE '              PRE04810
+                    'SPECIFIED IN THE NULL STATEMENT PROTOTYPE'          PRE04820
              FINOUT = 'ABNORMAL TERMINATION'                :(END)       PRE04830
GOPARMERR   FINOUT = 'NOT ALL THE REQUIRED PARAMETERS WERE '              PRE04840
+                    'SPECIFIED IN THE GO TO STATEMENT PROTOTYPE'         PRE04850
              FINOUT = 'ABNORMAL TERMINATION'                :(END)       PRE04860
MINPARMERR  FINOUT = 'NOT ALL THE REQUIRED PARAMETERS WERE '              PRE04870
+                    'SPECIFIED IN THE ASSIGN MINUS STATEMENT PROTOTYPE'  PRE04880
              FINOUT = 'ABNORMAL TERMINATION'                :(END)       PRE04890
PLUSPARMERR FINOUT = 'NOT ALL THE REQUIRED PARAMETERS WERE '              PRE04900
+                    'SPECIFIED IN THE ASSIGN PLUS STATEMENT PROTOTYPE'   PRE04910
              FINOUT = 'ABNORMAL TERMINATION'                :(END)       PRE04920
IFPARMERR   FINOUT = 'NOT ALL THE REQUIRED PARAMETERS WERE '              PRE04930
+                    'SPECIFIED IN THE IF THEN GO TO STATEMENT PROTOTYPE' PRE04940
              FINOUT = 'ABNORMAL TERMINATION'                :(END)       PRE04950
LABERROR    FINOUT = 'AN ERROR WAS ENCOUNTERED IN SCANNING THE UNIQUE '   PRE04960
+                    'LABEL DEFINITION'                                   PRE04970
              FINOUT = 'ABNORMAL TERMINATION'                :(END)       PRE04980
VARERROR    FINOUT = 'AN ERROR WAS ENCOUNTERED IN SCANNING THE UNIQUE '   PRE04990
+                    'IDENTIFIER DEFINITION'                              PRE05000
              FINOUT = 'ABNORMAL TERMINATION'                :(END)       PRE05010
CONTERR     FINOUT = 'AN ERROR WAS ENCOUNTERED IN SCANNING THE '          PRE05020
+                    'CONTINUATION DEFINITION'                            PRE05030
              FINOUT = 'ABNORMAL TERMINATION'                :(END)       PRE05040
LITERROR    FINOUT = 'AN ERROR WAS ENCOUNTERED IN SCANNING THE '          PRE05050
+                    'LITERAL DEFINITION'                                 PRE05060
              FINOUT = 'ABNORMAL TERMINATION'                :(END)       PRE05070
LANLABERR   FINOUT = 'AN ERROR WAS ENCOUNTERED IN SCANNING THE '          PRE05080
+                    'LANGUAGE LABEL DEFINITION'                          PRE05090
              FINOUT = 'ABNORMAL TERMINATION'                :(END)       PRE05100
GTHANDEFERR FINOUT = 'AN ERROR WAS ENCOUNTERED IN SCANNING THE '          PRE05110
+                    'GREATER THAN OPERATOR'                              PRE05120
              FINOUT = 'DEFINITION = ' OPERDEF                           PRE05130
              FINOUT = 'ABNORMAL TERMINAL'                   :(END)       PRE05140
LEQUDEFERR  FINOUT = 'AN ERROR WAS ENCOUNTERED IN SCANNING THE '          PRE05150
+                    'LESS THAN OR EQUAL OPERATOR'                        PRE05160
              FINOUT = 'DEFINITION = ' OPERDEF                           PRE05170
```

```
            FINOUT = 'ABNORMAL TERMINATION'                    :(END)          PRE05180
GEQUDEFERR  FINOUT = 'AN ERROR WAS ENCOUNTERED IN SCANNING THE '               PRE05190
+             'GREATER THAN OR EQUAL OPERATOR'                                 PRE05200
            FINOUT = 'DEFINITION = ' OPERDEF                                   PRE05210
            FINOUT = 'ABNORMAL TERMINATION'                    :(END)          PRE05220
NOTDEFERR   FINOUT = 'AN ERROR WAS ENCOUNTERED IN SCANNING THE '               PRE05230
+             'NOT OPERATOR'                                                   PRE05240
            FINOUT = 'DEFINITION = ' OPERDEF                                   PRE05250
            FINOUT = 'ABNORMAL TERMINATION'                    :(END)          PRE05260
NEQUDEFERR  FINOUT = 'AN ERROR WAS ENCOUNTERED IN SCANNING THE '               PRE05270
+             'NOT EQUAL OPERATOR'                                             PRE05280
            FINOUT = 'DEFINITION = ' OPERDEF                                   PRE05290
EOSERR      FINOUT = 'A COLUMN SPECIFICATION DID NOT CONTAIN ONLY '            PRE05300
+             'DIGITS OR EXTRANEOUS INFORMATION WAS FOUND IN '                 PRE05310
+             'SCANNING THE END OF STATEMENT DEFINITION'                       PRE05320
            FINOUT = 'ABNORMAL TERMINATION'                    :(END)          PRE05330
IGNERR      FINOUT = 'A COLUMN SPECIFICATION DID NOT CONTAIN '                 PRE05340
+             'ONLY DIGITS IN THE IGNORE COLUMNS DEFINITION'                   PRE05350
            FINOUT = 'ABNORMAL TERMINATION'                    :(END)          PRE05360
COMERROR    FINOUT = 'AN ERROR WAS ENCOUNTERED IN SCANNING THE '               PRE05370
+             'COMMENT DEFINITION'                                             PRE05380
            FINOUT = 'ABNORMAL TERMINATION'                    :(END)          PRE05390
MACDEFERR   FINOUT = 'THE <MACRO DEFINITION> CARD FOR MACRO ' NAME             PRE05400
+             ' IS MISSING OR INCORRECT'                                       PRE05410
            FINOUT = 'ABNORMAL TERMINATION'                    :(END)          PRE05420
MACCALLERR  FINOUT = 'AN ERROR WAS ENCOUNTERED IN A MACRO CALLING FORM '       PRE05430
+             'DEFINITION'                                                     PRE05440
            FINOUT = 'MACRO NAME = ' NAME                                      PRE05450
            FINOUT = 'ABNORMAL TERMINATION'                    :(END)          PRE05460
MACPERR     FINOUT = 'MORE THAN 6 PARAMETERS WERE SPECIFIED FOR MACRO '        PRE05470
+             NAME                                                             PRE05480
            FINOUT = 'ABNORMAL TERMINATION'                    :(END)          PRE05490
*                                                                              PRE05500
*******************************************************************************PRE05501
*                                                                              PRE05502
*     EXITS FOR ERRORS WHICH OCCURRED IN THE PARSER                            PRE05510
*                                                                              PRE05520
*******************************************************************************PRE05521
*                                                                              PRE05522
PSCANERR    FINOUT = 'SCAN RETURNED TO THE PARSER WITH FAILURE '               PRE05530
+             'WHEN A RETURNED VALUE WAS REQUIRED'                             PRE05540
            FINOUT = 'CURSTMT = ' CURSTMT                                      PRE05550
            FINOUT = 'WORKING = ' WORKING                      :(COPYTEMPOUT)  PRE05560
PUNIDERR    FINOUT = 'PARSER IS UNABLE TO RECOGNIZE THE SEQUENCE '             PRE05570
+             'OF TOKENS IN THE WORKING STRING'                                PRE05580
            FINOUT = 'CURSTMT = ' CURSTMT                                      PRE05590
            FINOUT = 'WORKING = ' WORKING                      :(COPYTEMPOUT)  PRE05600
PSTRERR     FINOUT = ' FAILURE TO PARSE STRUC TO STRLIST REDUCTION'            PRE05610
```

```
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05620
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE05630
PEDGRPERR   FINOUT = 'FAILURE TO PARSE ENDDOGROUP'                           PRE05640
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05650
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE05660
PEDOERR     FINOUT = 'FAILURE TO PARSE ENDDO'                                PRE05670
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05680
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE05690
PIFTHERR    FINOUT = 'FAILURE TO PARSE IF THEN CLAUSE'                       PRE05700
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05710
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE05720
PEWHERR     FINOUT = 'FAILURE TO PARSE ENDWHILE'                             PRE05730
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05740
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE05750
PECASEERR   FINOUT = 'FAILURE TO PARSE ENDCASE'                              PRE05760
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05770
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE05780
PQUERR      FINOUT = 'FAILURE TO PARSE QUITLOOP'                             PRE05790
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05800
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE05810
PWHERR      FINOUT = 'FAILURE TO PARSE WHILE'                                PRE05820
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05830
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE05840
PREPUERR    FINOUT = 'FAILURE TO PARSE REPUNTIL'                             PRE05850
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05860
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE05870
PREPERR     FINOUT = 'FAILURE TO PARSE REPEAT'                               PRE05880
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05890
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE05900
PIFERR      FINOUT = 'FAILURE TO PARSE IF'                                   PRE05910
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05920
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE05930
PDGRPERR    FINOUT = 'FAILURE TO PARSE DOGROUP'                              PRE05940
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05950
PDOERR      FINOUT = 'FAILURE TO PARSE DO'                                   PRE05960
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE05970
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE05980
PCASEERR    FINOUT = 'FAILURE TO PARSE CASE'                                 PRE05990
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE06000
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE06010
PEREPERR    FINOUT = 'FAILURE TO PARSE ENDREPEAT'                            PRE06020
            FINOUT = 'CURSTMT = ' CURSTMT                                    PRE06030
            FINOUT = 'WORKING = ' WORKING          :(COPYTEMPOUT)            PRE06040
*                                                                            PRE06050
*************************************************************************** PRE06051
*                                                                            PRE06052
*     EXITS FOR ERRORS WHICH OCCURRED IN THE SCANNER                         PRE06060
*                                                                            PRE06070
```

```
*************************************************************************PRE06071
*                                                                       PRE06072
COMOPENERR FINOUT = 'STRING SPECIFIED AS COMMENT OPENER IS '            PRE06080
+                   'LONGER THAN 2 CHARACTERS'          :(COPYTEMPOUT)  PRE06090
COMENDERR  FINOUT = 'STRING SPECIFIED AS COMMENT ENDER IS'              PRE06100
+                   ' LONGER THAN 2 CHARACTERS'         :(COPYTEMPOUT)  PRE06110
LITOPENERR FINOUT = 'STRING SPECIFIED AS LITERAL OPENER IS '            PRE06120
+                   'LONGER THAN 2 CHARACTERS'          :(COPYTEMPOUT)  PRE06130
LITENDERR  FINOUT = 'STRING SPECIFIED AS LITERAL ENDER IS '             PRE06140
+                   'LONGER THAN 2 CHARACTERS'          :(COPYTEMPOUT)  PRE06150
EOSDEFNERR FINOUT = 'STRING SPECIFIED AS END OF STATEMENT INDICATOR '    PRE06160
+                   'IS LONGER THAN 2 CHARACTERS'       :(COPYTEMPOUT)  PRE06170
LANLABDEFNERR FINOUT = 'LANGUAGE LABEL PATTERN DOES NOT SPECIFY '        PRE06180
+                   'EITHER SINGLE COLUMN OR SEQUENCE OF COLUMNS '       PRE06190
+                   'FOR LABEL BEGINNING'               :(COPYTEMPOUT)  PRE06200
ERREND     FINOUT = 'AN END OF INPUT WAS FOUND WITHOUT AN '             PRE06210
+                   'END OF STATEMENT DURING SCANNING'  :(COPYTEMPOUT)  PRE06220
*                                                                       PRE06230
*************************************************************************PRE06231
*                                                                       PRE06232
*     EXITS FOR ERRORS WHICH OCCURRED IN THE SEMANTIC ROUTINES          PRE06240
*                                                                       PRE06250
*************************************************************************PRE06251
*                                                                       PRE06252
ELSEERROR  FINOUT = 'ERROR IN SEMANTIC PROCESSING OF ELSE'              PRE06260
           FINOUT = 'INSTR = ' INSTR                :(COPYTEMPOUT)      PRE06270
CASEERROR  FINOUT = 'ERROR IN SEMANTIC PROCESSING OF CASE'              PRE06280
           FINOUT = 'INSTR = ' INSTR                :(COPYTEMPOUT)      PRE06290
DOERROR    FINOUT = 'ERROR IN SEMANTIC PROCESSING OF DO'                PRE06300
           FINOUT = 'INSTR = ' INSTR                :(COPYTEMPOUT)      PRE06310
DOGRPERROR FINOUT = 'ERROR IN SEMANTIC PROCESSING OF DOGROUP'           PRE06320
           FINOUT = 'INSTR = ' INSTR                :(COPYTEMPOUT)      PRE06330
IFERROR    FINOUT = 'ERROR IN SEMANTIC PROCESSING OF IF'                PRE06340
           FINOUT = 'INSTR = ' INSTR                :(COPYTEMPOUT)      PRE06350
REPERROR   FINOUT = 'ERROR IN SEMANTIC PROCESSING OF REPEAT'            PRE06360
           FINOUT = 'INSTR = ' INSTR                :(COPYTEMPOUT)      PRE06370
WHERROR    FINOUT = 'ERROR IN SEMANTIC PROCESSING OF WHILE'             PRE06380
           FINOUT = 'INSTR = ' INSTR                :(COPYTEMPOUT)      PRE06390
QUITERROR  FINOUT = 'ERROR IN SEMANTIC PROCESSING OF QUITLOOP'          PRE06400
           FINOUT = 'INSTR = ' INSTR                :(COPYTEMPOUT)      PRE06410
ECASEERROR FINOUT = 'ERROR IN SEMANTIC PROCESSING OF ENDCASE'           PRE06420
           FINOUT = 'INSTR = ' INSTR                :(COPYTEMPOUT)      PRE06430
EIFERROR   FINOUT = 'ERROR IN SEMANTIC PROCESSING OF ENDIF'             PRE06440
           FINOUT = 'INSTR = ' INSTR                :(COPYTEMPOUT)      PRE06450
EWHILEERROR FINOUT = 'ERROR IN SEMANTIC PROCESSING OF ENDWHILE'         PRE06460
           FINOUT = 'INSTR = ' INSTR                :(COPYTEMPOUT)      PRE06470
EREPERROR  FINOUT = 'ERROR IN SEMANTIC PROCESSING OF ENDREPEAT'         PRE06480
           FINOUT = 'INSTR = ' INSTR                :(COPYTEMPOUT)      PRE06490
```

```
EDOERROR    FINOUT = 'ERROR IN SEMANTIC PROCESSING OF ENDDO'                    PRE06500
            FINOUT = 'INSTR = ' INSTR                    :(COPYTEMPOUT)         PRE06510
EDOGRPERROR FINOUT = 'ERROR IN SEMANTIC PROCESSING OF ENDDOGROUP'               PRE06520
            FINOUT = 'INSTR = ' INSTR                    :(COPYTEMPOUT)         PRE06530
MACCALLPERR FINOUT = 'MORE THAN SIX ARGUMENTS WERE SUPPLIED IN A CALL '         PRE06540
+                    'TO THE MACRO ' SELMAC              :(COPYTEMPOUT)         PRE06550
ROUTOIERR   FINOUT = 'THE STMTLEN VARIABLE EQUALS 0 IN THE MACRO '              PRE06560
+                    'DEFINITION FOR ' SELMAC            :(COPYTEMPOUT)         PRE06570
MACARGERR   FINOUT = 'A PARAMETER FOUND IN THE MACRO DFINITION ARRAY '          PRE06580
+                    'FOR ' SELMAC ' IS NOT IN THE PARAMETER ARRAY FOR '        PRE06590
+                    'THAT MACRO'                        :(COPYTEMPOUT)         PRE06600
MACBLKERR   FINOUT = 'NOT ENOUGH BLANKS LEFT AFTER A PARAMETER '                PRE06610
+                    'NAME IN THE DEFINITION FOR THE MACRO ' SELMAC             PRE06620
+                                                        :(COPYTEMPOUT)         PRE06630
TOOFULLSTK  FINOUT = 'ATTEMPT TO PUSH ONTO A FULL SEMANTIC STACK'               PRE06640
+                                                        :(COPYTEMPOUT)         PRE06650
EMPTYSTK    FINOUT = 'ATTEMPT TO POP FROM AN EMPTY SEMANTIC STACK'              PRE06660
+                                                        :(COPYTEMPOUT)         PRE06670
*                                                                              PRE06680
*****************************************************************************PRE06681
*                                                                              PRE06682
*     EXITS FOR ERRORS WHICH OCCURRED IN THE CREATION ROUTINES                 PRE06690
*                                                                              PRE06700
*****************************************************************************PRE06701
*                                                                              PRE06702
CRMINERR    FINOUT = 'AN ERROR WAS ENCOUNTERED DURING AN ATTEMPT TO '          PRE06710
+                    'CREATE AN ASSIGN MINUS STATEMENT' :(COPYTEMPOUT)         PRE06720
CRDCLERR    FINOUT = 'AN ERROR WAS ENCOUNTERED DURING AN ATTEMPT TO '          PRE06730
+                    'CREATE A DECLARATION STATEMENT'   :(COPYTEMPOUT)         PRE06740
CRIFERR     FINOUT = 'AN ERROR WAS ENCOUNTERED DURING AN ATTEMPT TO '          PRE06750
+                    'CREATE AN IF THEN GO TO STATEMENT'                       PRE06760
+                                                       :(COPYTEMPOUT)         PRE06770
CRNULLERR   FINOUT = 'AN ERROR WAS ENCOUNTERED DURING AN ATTEMPT TO '          PRE06780
+                    'CREATE A NULL STATEMENT'          :(COPYTEMPOUT)         PRE06790
CRPLERR     FINOUT = 'AN ERROR WAS ENCOUNTERED DURING AN ATTEMPT TO '          PRE06800
+                    'CREATE AN ASSIGN PLUS STATEMENT'  :(COPYTEMPOUT)         PRE06810
CRGOTOERR   FINOUT = 'AN ERROR WAS ENCOUNTERED DURING AN ATTEMPT TO '          PRE06820
+                    'CREATE A GO TO STATEMENT'         :(COPYTEMPOUT)         PRE06830
DCLBLKERR   FINOUT = 'NOT ENOUGH BLANKS LEFT AFTER A PARAMETER NAME '          PRE06840
+                    'IN THE DECLARATION PROTOTYPE'                            PRE06850
            FINOUT = 'VARNAME = ' VARNAME              :(COPYTEMPOUT)         PRE06860
GOTOBLKERR  FINOUT = 'NOT ENOUGH BLANKS LEFT AFTER A PARAMETER NAME '          PRE06870
+                    'IN THE GO TO STATEMENT PROTOTYPE'                        PRE06880
            FINOUT = 'CURSTMT = ' CURSTMT              :(COPYTEMPOUT)         PRE06890
PLUSBLKERR  FINOUT = 'NOT ENOUGH BLANKS LEFT AFTER A PARAMETER NAME '          PRE06900
+                    'IN THE ASSIGN PLUS PROTOTYPE'                            PRE06910
            FINOUT = 'CURSTMT = ' CURSTMT              :(COPYTEMPOUT)         PRE06920
MINBLKERR   FINOUT = 'NOT ENOUGH BLANKS LEFT AFTER A PARAMETER NAME '          PRE06930
```

```
+                    'IN THE ASSIGN MINUS PROTOTYPE'                                PRE06940
          FINOUT = 'CURSTMT = ' CURSTMT                    :(COPYTEMPOUT)          PRE06950
NULLBLKERR FINOUT = 'NOT ENOUGH BLANKS LEFT AFTER A PARAMETER NAME '               PRE06960
+                    'IN THE NULL STATEMENT PROTOTYPE'                              PRE06970
          FINOUT = 'CURSTMT = ' CURSTMT                    :(COPYTEMPOUT)          PRE06980
IFBLKERR  FINOUT = 'NOT ENOUGH BLANKS LEFT AFTER A PARAMETER NAME '                PRE06990
+                    'IN THE IF THEN GO TO STATEMENT PROTOTYPE'                     PRE07000
          FINOUT = 'CURSTMT = ' CURSTMT                    :(COPYTEMPOUT)          PRE07010
TOOMANYLABELS FINOUT = 'ATTEMPT TO CREATE MORE UNIQUE LABELS THAN '                PRE07020
+                    'THE DEFINITION ALLOWS'                :(COPYTEMPOUT)          PRE07030
TOOMANYVARS FINOUT = 'ATTEMPT TO CREATE MORE UNIQUE IDENTIFIERS THAN '             PRE07040
+                    'THE DEFINITION ALLOWS'                :(COPYTEMPOUT)          PRE07050
BEGSCAN   OLDSTMTEOS = STMTEOS                                                     PRE07060
*                                                                                  PRE07070
*********************************************************************************PRE07071
*                                                                                  PRE07072
*     ROUTINE TO SCAN INPUT FOR TOKENS RECOGNIZABLE BY THE PARSER                  PRE07080
*                                                                                  PRE07090
*     NO PARAMETERS                                                                PRE07100
*                                                                                  PRE07110
*     A CARD IMAGE MAY BE PRESENT IN NEXTCARD OR NOT WHEN THIS ROUTINE             PRE07120
*        IS INITIALLY CALLED                                                       PRE07130
*                                                                                  PRE07140
*********************************************************************************PRE07141
*                                                                                  PRE07150
*     IF AN EOS SHOULD BE SIGNALLED IMMEDIATELY, BRANCH TO THAT LABEL              PRE07160
*                                                                                  PRE07170
*     SAVE FLAG INDICATING WHETHER OR NOT AN EOS WAS FOUND IN THE LAST             PRE07180
*        STATEMENT; SET EOS FLAG FOR THIS STATEMENT TO 0                           PRE07190
*                                                                                  PRE07200
          STMTEOS = 0                                                              PRE07210
          EQ(ECSNEXT,1)                                :S(FOSIMMED)                PRE07220
*                                                                                  PRE07230
*     GET THE NEXT USEABLE CHARACTER AND ITS COLUMN NUMBER; IF GETCHAR             PRE07240
*        RETURNS WITH FAILURE, BRANCH                                              PRE07250
*                                                                                  PRE07260
BEGSCANI  GETCHAR('INCHAR','CHARCOL')                  :F(ENDOFINPUT)              PRE07270
*                                                                                  PRE07280
*     IF THIS IS THE FIRST CHARACTER OF A TOKEN OR THE FIRST NON-BLANK             PRE07290
*        CHARACTER OF A TOKEN, SET THE STARTCOL VARIABLE                           PRE07300
*                                                                                  PRE07310
          STARTCOL = EQ(SIZE(TOKEN),0) CHARCOL                                     PRE07320
*                                                                                  PRE07330
*     IF THIS IS THE FIRST USEABLE COLUMN ON A CARD, CHECK TO                      PRE07340
*        SEE IF THE TOKEN IS CURRENTLY ALL BLANKS LEFT OVER FROM A                 PRE07350
*        PREVIOUS CARD WHICH SHOULD BE DISCARDED                                   PRE07360
*                                                                                  PRE07370
OLDSTART  NE(IGNPAT<1>,1)                              :S(CKATCOLONE)              PRE07380
```

```
            IGNPAT<0> 'ONE'                                    :F(LOOKATSEQEND)   PRE07390
            EQ(CHARCOL,2)                                      :F(LEAVETOKEN)     PRE07400
LOOKATSEQEND EQ(CHARCOL,IGNPAT<2> + 1)      :F(LEAVETOKEN)S(CKBLANKTOKEN)PRE07410
CKATCOLONE  EQ(CHARCOL,1)                                      :F(LEAVETOKEN)     PRE07420
CKBLANKTOKEN TOKEN POS(0) SPAN(' ') RPOS(0)                    :S(TESTOLDEOS)     PRE07430
*                                                                                 PRE07440
*     IF THE TOKEN IS OF 0 LENGTH AND A COMMENT WAS NOT ALL THAT                  PRE07450
*        APPEARED ON THE LAST CARD, LEAVE THE TOKEN AND CURSTMT INTACT            PRE07460
            EQ(COMMENTONLY,1)                                  :F(LEAVETOKEN)     PRE07470
            INSERTCOM(CURSTMT)                                                    PRE07480
*                                                                                 PRE07490
*     THE CURSTMT AND TOKEN ARE NULLIFIED IF AN EOS WAS FOUND BEFORE              PRE07500
*        THIS CALL OF SCAN                                                        PRE07510
*                                                                                 PRE07520
TESTOLDEOS EQ(OLDSTMTEOS,1)                                    :F(LEAVETOKEN)     PRE07530
            CURSTMT =                                                             PRE07540
            TOKEN =                                                               PRE07550
*                                                                                 PRE07560
*     ADD THE CHARACTER TO THE TOKEN STRING                                       PRE07570
*                                                                                 PRE07580
LEAVETOKEN  TOKEN = TOKEN INCHAR                                                  PRE07590
            LABSTART = 0                                                          PRE07600
            COMMENTONLY = 0                                                       PRE07610
*                                                                                 PRE07620
*********************************************************************************PRE07621
*                                                                                 PRE07622
*     LOOK FOR BEGINNING OF A COMMENT                                             PRE07623
*                                                                                 PRE07624
*********************************************************************************PRE07625
*                                                                                 PRE07626
*     BRANCH TO APPROPRIATE SECTION DEPENDING ON OPENING                          PRE07630
*        FOR COMMENT SPECIFIED IN DEFINITION                                      PRE07640
*                                                                                 PRE07650
            COMPAT<1> '10'                                 :S(COMSYM)             PRE07660
*                                                                                 PRE07670
*     COLUMN NUMBER PART OR ALL OF COMMENT OPENING                                PRE07680
*                                                                                 PRE07690
            EQ(CHARCOL,COMPAT<4>)                          :F(NOTCOM)             PRE07700
*                                                                                 PRE07710
*     COMMENT FOUND IF COLUMN ONLY STARTS COMMENT                                 PRE07720
*                                                                                 PRE07730
            COMPAT<1> '11'                       :F(COMFOUND)S(COMCOLSYM)         PRE07740
*                                                                                 PRE07750
*     SYMBOL ONLY STARTS COMMENT                                                  PRE07760
*                                                                                 PRE07770
COMSYM      COMPAT<3> POS(0) INCHAR                            :F(NOTCOM)         PRE07780
*                                                                                 PRE07790
*     COMMENT FOUND IF ONLY ONE SYMBOL REQUIRED                                   PRE07800
```

```
*                                                                                    PRE07810
        EQ(SIZE(COMPAT<3>),1)                                   :S(COMFOUND)         PRE07820
*                                                                                    PRE07830
*   ERROR IF COMMENT STARTING CHARACTER STRING IS                                    PRE07840
*       LONGER THAN 2 CHARACTERS                                                     PRE07850
*                                                                                    PRE07860
        EQ(SIZE(COMPAT<3>),2)                                   :F(COMOPENERR)       PRE07870
*                                                                                    PRE07880
*   SAVE THE CURRENT CHARACTER AND COLUMN                                            PRE07890
*                                                                                    PRE07900
        OLDCOL = CHARCOL                                                             PRE07910
        OLDCHAR = INCHAR                                                             PRE07920
*                                                                                    PRE07930
*   GET THE NEXT USEABLE CHARACTER AND ITS COLUMN                                    PRE07940
*                                                                                    PRE07950
        GETCHAR('INCHAR','CHARCOL')                                                  PRE07960
*                                                                                    PRE07970
*   BRANCH IF THE 2 CHARACTERS DO NOT OPEN A COMMENT;                                PRE07980
*       OTHERWISE ADD THE NEW CHARACTER TO THE TOKEN STRING                          PRE07990
*                                                                                    PRE08000
        COMPAT<3> OLDCHAR INCHAR                                :F(NOTCOM2CHARS)     PRE08010
        TOKEN = TOKEN INCHAR                                    :(COMFOUND)          PRE08020
*                                                                                    PRE08030
*   COLUMN NUMBER AND SYMBOL START COMMENT                                           PRE08040
*                                                                                    PRE08050
CCMCOLSYM  COMPAT<3> POS(0) INCHAR                              :F(NOTCOM)           PRE08060
*                                                                                    PRE08070
*   COMMENT FOUND IF ONLY ONE SYMBOL REQUIRED                                        PRE08080
*                                                                                    PRE08090
        EQ(SIZE(COMPAT<3>),1)                                   :S(COMFOUND)         PRE08100
*                                                                                    PRE08110
*   ERROR IF COMMENT STARTING STRING LONGER THAN 2 CHARACTERS                        PRE08120
*                                                                                    PRE08130
        EQ(SIZE(COMPAT<3>),2)                                   :F(COMOPENERR)       PRE08140
*                                                                                    PRE08150
*   SAVE THE CURRENT CHARACTER AND CCLUMN                                            PRE08160
*                                                                                    PRE08170
        OLDCOL = CHARCOL                                                             PRE08180
        OLDCHAR = INCHAR                                                             PRE08190
*                                                                                    PRE08200
*   GET THE NEXT USEABLE CHARACTER AND ITS COLUMN                                    PRE08210
*                                                                                    PRE08220
        GETCHAR('INCHAR','CHARCOL')                             :F(NOTCOM2CHARS)     PRE08230
*                                                                                    PRE08240
*   BRANCH IF THE 2 CHARACTERS DO NOT OPEN A COMMENT;                                PRE08250
*       OTHERWISE ADD THE NEW CHARACTER TO THE TOKEN STRING                          PRE08260
*                                                                                    PRE08270
        COMPAT<3> OLDCHAR INCHAR                                :F(NOTCOM2CHARS)     PRE08280
```

```
            TOKEN = TOKEN INCHAR                                :(COMFOUND)          PRE08290
*                                                                                    PRE08300
*     BACKUP BY RESTORING THE OLD CHARACTTER AND OLD COLUMN NUMBER;                  PRE08310
*         BRANCH TO SEARCH FOR SOMETHING ELSE                                        PRE08320
*                                                                                    PRE08330
NOTCOM2CHARS INCHAR = OLDCHAR                                                        PRE08340
            CHARCOL = OLDCOL                                                         PRE08350
            COLNUM = OLDCOL                                 :(NOTCOM)            PRE08360
*                                                                                    PRE08370
*********************************************************************************** PRE08371
*                                                                                    PRE08372
*     LOOK FOR THE END OF THE COMMENT                                                PRE08373
*                                                                                    PRE08374
*********************************************************************************** PRE08375
*                                                                                    PRE08376
*     COMMENT HAS BEEN FOUND; RESET THE STMTEOS FLAG TO WHATEVER IF WAS              PRE08380
*         BEFORE THIS CALL OF SCAN; SET THE COMMENTONLY FLAG IF NOTHING              PRE08390
*         ELSE HAS BEEN ENCOUNTERED IN THIS STATEMENT; SEARCH FOR THE ENDPRE08400
*         OF THE COMMENT                                                           PRE08410
*                                                                                    PRE08420
COMFOUND    STMTEOS = OLDSTMTEOS                                                     PRE08430
            NE(SIZE(CURSTMT),0)                             :S(SKIPNOSTMT)       PRE08440
            COMMENTONLY = 1                                                          PRE08450
*                                                                                    PRE08460
*     GET THE NEXT USEABLE CHARACTER AND ITS COLUMN NUMBER; ADD THE                  PRE08470
*         CHARACTER TO THE TOKEN STRING                                            PRE08480
*                                                                                    PRE08490
SKIPNOSTMT GETCHAR('INCHAR','CHARCOL')                      :F(ENDOFINPUT)       PRE08500
            TOKEN = TOKEN INCHAR                                                     PRE08510
*                                                                                    PRE08520
*     BRANCH TO APPROPRIATE SECTION DEPENDING ON CLOSING FOR COMMENT                 PRE08530
*         SPECIFIED IN DEFINITION                                                  PRE08540
*                                                                                    PRE08550
COMFOUNDNOCHAR COMPAT<2> '10'                               :S(COMENDSTR)        PRE08560
*                                                                                    PRE08570
*     COLUMN NUMBER PART OR ALL OF COMMENT CLOSING                                   PRE08580
*                                                                                    PRE08590
            EQ(CHARCOL,COMPAT<6>)                           :F(COMFOUND)         PRE08600
*                                                                                    PRE08610
*     COMMENT ENDS IF ONLY COLUMN ENDS COMMENT                                       PRE08620
*                                                                                    PRE08630
            COMPAT<2> '11'                                  :F(ENDCOM)           PRE08640
*                                                                                    PRE08650
*     COLUMN NUMBER AND SYMBOL END COMMENT                                           PRE08660
*                                                                                    PRE08670
COMENDCOLSYM COMPAT<5> POS(0) INCHAR                        :F(NOTCOMEND)        PRE08680
*                                                                                    PRE08690
*     COMMENT ENDS IF ONLY ONE SYMBOL REQUIRED                                       PRE08700
```

```
*                                                                       PRE08710
        EQ(SIZE(COMPAT<5>),1)                          :S(ENDCOM)       PRE08720
*                                                                       PRE08730
*     ERROR IF COMMENT CLOSING STRING IS LONGER THAN 2 CHARS.           PRE08740
*                                                                       PRE08750
        EQ(SIZE(COMPAT<5>),2)                          :F(COMENDERR)    PRE08760
*                                                                       PRE08770
*     SAVE THE CURRENT CHARACTER AND COLUMN                             PRE08780
*                                                                       PRE08790
        OLDCOL = CHARCOL                                                PRE08800
        OLDCHAR = INCHAR                                                PRE08810
*                                                                       PRE08820
*     GET THE NEXT USEABLE CHARACTER AND ITS COLUMN NUMBER;             PRE08830
*         ADD THE CHARACTER TO THE TOKEN STRING                         PRE08840
*                                                                       PRE08850
        GETCHAR('INCHAR','CHARCOL')                    :F(ENDOFINPUT)   PRE08860
        TOKEN = TOKEN INCHAR                                            PRE08870
*                                                                       PRE08880
*     COMMENT ENDS IF THE 2 CHARACTERS CLOSE A COMMENT;                 PRE08890
*         OTHERWISE KEEP SEARCHING FOR THE END                          PRE08900
*                                                                       PRE08910
        COMPAT<5> OLDCHAR INCHAR     :S(ENDCOM)F(COMFOUNDNOCHAR)        PRE08920
*                                                                       PRE08930
*     SYMBOL ONLY ENDS COMMENT, BRANCH IF THIS CHARACTER MAY            PRE08940
*         INDICATE THE END OF THE COMMENT                               PRE08950
*                                                                       PRE08960
COMENDSTR  COMPAT<5> POS(0) INCHAR                     :S(FIRSTCHAREND) PRE08970
*                                                                       PRE08980
*     GET THE NEXT USEABLE CHARACTER AND ITS COLUMN NUMBER;             PRE08990
*         ADD THE CHARACTER TO THE TOKEN STRING; KEEP LOOKING           PRE09000
*         FOR THE CLOSING SYMBOL                                        PRE09010
*                                                                       PRE09020
        GETCHAR('INCHAR','CHARCOL')                    :F(ENDOFINPUT)   PRE09030
        TOKEN = TOKEN INCHAR                           :(COMENDSTR)     PRE09040
*                                                                       PRE09050
*     ONE SYMBOL HAS BEEN MATCHED; END OF COMMENT HAS BEEN FOUND        PRE09060
*         IF ONLY ONE CHARACTER IS NEEDED                               PRE09070
*                                                                       PRE09080
FIRSTCHAREND EQ(SIZE(COMPAT<5>),1)                     :S(ENDCOM)       PRE09090
*                                                                       PRE09100
*     ERROR IF COMMENT CLOSING STRING LONGER THAN 2 CHARS.              PRE09110
*                                                                       PRE09120
        EQ(SIZE(COMPAT<5>),2)                          :F(COMENDERR)    PRE09130
*                                                                       PRE09140
*     SAVE THE CURRENT CHARACTER AND COLUMN                             PRE09150
*                                                                       PRE09160
        OLDCOL = CHARCOL                                                PRE09170
        OLDCHAR = INCHAR                                                PRE09180
```

```
*                                                                          PRE09190
*     GET THE NEXT USEABLE CHARACTER AND ITS COLUMN;                       PRE09200
*          ADD THE CHARACTER TO THE TOKEN STRING                           PRE09210
*                                                                          PRE09220
          GETCHAR('INCHAR','CHARCOL')                :F(ENDOFINPUT)      PRE09230
          TOKEN = TOKEN INCHAR                                             PRE09240
*                                                                          PRE09250
*     END OF COMMENT FOUND IF THE 2 CHARACTERS CLOSE A COMMENT;            PRE09260
*          OTHERWISE KEEP SEARCHING                                        PRE09270
*                                                                          PRE09280
          COMPAT<5> OLDCHAR INCHAR                   :F(COMENDSTR)       PRE09290
*                                                                          PRE09300
*     END OF COMMENT FOUND                                                 PRE09310
*                                                                          PRE09320
ENDCOM    CURSTMT = CURSTMT TOKEN                                          PRE09330
          TOKEN =                                    :(BEGSCAN1)         PRE09340
*                                                                          PRE09350
*************************************************************************PRE09351
*                                                                          PRE09352
*     NOT A COMMENT; LOOK FOR A LITERAL OPENING                            PRE09360
*                                                                          PRE09370
*************************************************************************PRE09371
*                                                                          PRE09372
NOTCOM    LITPAT<1> POS(0) INCHAR                    :F(NOTLIT)          PRE09380
*                                                                          PRE09390
*     LITERAL FOUND IF THE SINGLE SYMBOL STARTS A LITERAL                  PRE09400
*                                                                          PRE09410
          EQ(SIZE(LITPAT<1>),1)                      :S(LITFOUND)        PRE09420
*                                                                          PRE09430
*     ERROR IF LITERAL OPENING STRING LONGER THAN 2 CHARACTERS             PRE09440
*                                                                          PRE09450
          EQ(SIZE(LITPAT<2>),2)                      :F(LITOPENERR)      PRE09460
*                                                                          PRE09470
*     SAVE THE CURRENT CHARACTER AND ITS COLUMN NUMBER                     PRE09480
*                                                                          PRE09490
          OLDCOL = CHARCOL                                                 PRE09500
          OLDCHAR = INCHAR                                                 PRE09510
*                                                                          PRE09520
*     GET THE NEXT USEABLE CHARACTER AND ITS COLUMN NUMBER                 PRE09530
          GETCHAR('INCHAR','CHARCOL')                :F(NOTLIT2CHARS)    PRE09540
*                                                                          PRE09550
*     BRANCH IF THE 2 CHARACTERS DO NOT START A LITERAL;                   PRE09560
*                                                                          PRE09570
*          OTHERWISE ADD THE CHARACTER TO THE TOKEN STRING                 PRE09580
*                                                                          PRE09590
          LITPAT<1> OLDCHAR INCHAR                   :F(NOTLIT2CHARS)    PRE09600
          TOKEN = TOKEN INCHAR                       :(LITFOUND)         PRE09610
*                                                                          PRE09620
```

```
*     BACKUP BY RESTORING OLD CHARACTER AND COLUMN NUMBER;                    PRE09630
*         BRANCH TO SEARCH FOR SOMETHING ELSE                                PRE09640
*                                                                            PRE09650
NOTLIT2CHARS INCHAR = OLDCHAR                                                PRE09660
             CHARCOL = OLDCOL                                                PRE09670
             CCLNUM = OLDCOL                                :(NOTLIT)        PRE09680
*                                                                            PRE09690
*************************************************************************** PRE09691
*                                                                            PRE09692
*     LOOK FOR THE END OF THE LITERAL                                        PRE09693
*                                                                            PRE09694
*************************************************************************** PRE09695
*                                                                            PRE09696
*     LITERAL FOUND; GET THE NEXT USEABLE CHARACTER AND ITS                  PRE09700
*         COLUMN NUMBER; ADD THE CHARACTER TO THE TOKEN STRING               PRE09710
*                                                                            PRE09720
LITFOUND     GETCHAR('INCHAR','CHARCOL')                :F(ENDOFINPUT)     PRE09730
             TOKEN = TOKEN INCHAR                                           PRE09740
*                                                                            PRE09750
*     IF CHARACTER DOES NOT CLOSE A LITERAL, KEEP SEARCHING                  PRE09760
*                                                                            PRE09770
             LITPAT<2> POS(0) INCHAR                    :F(LITFOUND)       PRE09780
*                                                                            PRE09790
*     BRANCH TO LOOK FOR SPECIAL SEQUENCE IF THE CHARACTER MATCHES           PRE09800
*         THE CLOSING SINGLE CHARACTER                                       PRE09810
*                                                                            PRE09820
             EQ(SIZE(LITPAT<2>),1)                      :S(MATCHLITEND1)   PRE09830
*                                                                            PRE09840
*     ERROR IF LITERAL CLOSING STRING IS LONGER THAN 2 CHARACTERS            PRE09850
*                                                                            PRE09860
             EQ(SIZE(LITPAT<2>),2)                      :F(LITENDERR)      PRE09870
*                                                                            PRE09880
*     SAVE THE CURRENT CHARACTER AND ITS COLUMN NUMBER                       PRE09890
*                                                                            PRE09900
             OLDCOL = CHARCOL                                               PRE09910
             OLDCHAR = INCHAR                                               PRE09920
*                                                                            PRE09930
*     GET THE NEXT USEABLE CHARACTER AND ITS COLUMN NUMBER;                  PRE09940
*         ADD THE CHARACTER TO THE TOKEN STRING                              PRE09950
*                                                                            PPE09960
             GETCHAR('INCHAR','CHARCOL')                :F(ENDOFINPUT)     PRE09970
             TOKEN = TOKEN INCHAR                                           PRE09980
*                                                                            PRE09990
*     LITERAL FOUND IF THE 2 CHARACTERS CLOSE A LITERAL;                     PRE10000
*         OTHERWISE KEEP SEARCHING FOR END                                   PRE10010
*                                                                            PRE10020
             LITPAT<2> OLDCHAR INCHAR    :F(LITFOUND)S(ENDLIT)              PRE10030
*                                                                            PRE10040
```

```
*     ONE CHARACTER HAS BEEN MATCHED; LITERAL END FOUND IF NO SPECIAL     PRE10050
*         SEQUENCE SPECIFIED IN DEFINITION                               PRE10060
*                                                                        PRE10070
MATCHLITEND1 EQ(SIZE(LITPAT<3>),0)                    :S(ENDLIT)       PRE10080
*                                                                        PRE10090
*     SAVE THE CURRENT CHARACTER AND COLUMN NUMBER                       PRE10100
*                                                                        PRE10110
          OLDCOL = CHARCOL                                              PRE10120
          OLDCHAR = INCHAR                                              PRE10130
*                                                                        PRE10140
*     GET THE NEXT USEABLE CHARACTER AND ITS COLUMN NUMBER               PRE10150
*                                                                        PRE10160
          GETCHAR('INCHAR','CHARCOL')                 :F(ENDLIT1)      PRE10170
*                                                                        PRE10180
*     END OF LITERAL FOUND IF THIS IS NOT THE SPECIAL SEQUENCE           PRE10190
*                                                                        PRE10200
          LITPAT<3> OLDCHAR INCHAR                    :F(ENDLIT1)      PRE10210
*                                                                        PRE10220
*     ADD THE CHARACTER TO THE TOKEN STRING; SKIP THIS CHARACTER         PRE10230
*         AND KEEP SEARCHING FOR THE LITERAL END                         PRE10240
*                                                                        PRE10250
          TOKEN = TOKEN INCHAR                        :(LITFOUND)      PRE10260
*                                                                        PRE10270
*     BACKUP BY RESTORING THE OLD CHARACTER AND ITS COLUMN NUMBER        PRE10280
*                                                                        PRE10290
ENDLIT1   INCHAR = OLDCHAR                                              PRE10300
          CHARCOL = OLDCOL                                              PRE10310
          CCLNUM = OLDCOL                                               PRE10320
*                                                                        PRE10330
*     A LITERAL IS AN UNIDENTIFIED OBJECT FOR THE PARSER, SO BRANCH TO   PRE10340
*         THAT SECTION                                                   PRE10350
*                                                                        PRE10360
ENDLIT                                                :(UNIDENT)       PRE10370
*                                                                        PRE10380
*************************************************************************PRE10381
*                                                                        PRE10382
*     NOT A LITERAL; SEARCH FOR AN END OF STATEMENT                      PRE10390
*                                                                        PRE10400
*************************************************************************PRE10401
*                                                                        PRE10402
NOTLIT    EOSPAT<0> 'COL'                             :S(EOSCOL)       PRE10410
*                                                                        PRE10420
*     SYMBOL SIGNIFIES END OF STATEMENT                                  PRE10430
*                                                                        PRE10440
          EOSPAT<1> POS(0) INCHAR                     :F(NOTEOSSCAN)   PRE10450
*                                                                        PRE10460
*     END OF STATEMENT FOUND IF SINGLE SYMBOL ENDS STATEMENT             PRE10470
*                                                                        PRE10480
```

```
            EQ(SIZE(EOSPAT<1>),1)                                :S(EOSFOUND)      PRE10490
*                                                                                  PRE10500
*     ERROR IF END OF STATEMENT STRING IS LONGER THAN 2 CHARACTERS                 PRE10510
*                                                                                  PRE10520
            EQ(SIZE(EOSPAT<1>),2)                                :(EOSDEFNERR)     PRE10530
*                                                                                  PRE10540
*     SAVE THE CURRENT CHARACTER AND COLUMN NUMBER                                 PRE10550
*                                                                                  PRE10560
            OLDCOL = CHARCOL                                                       PRE10570
            OLDCHAR = INCHAR                                                       PRE10580
*                                                                                  PRE10590
*     GET THE NEXT USEABLE CHARACTER AND ITS COLUMN NUMBER                         PRE10600
*                                                                                  PRE10610
            GETCHAR('INCHAR','CHARCOL')                          :F(NOTEOS2CHARS)  PRE10620
*                                                                                  PRE10630
*     BRANCH IF THE 2 CHARACTERS DO NOT MARK END OF STATEMENT;                     PRE10640
*         OTHERWISE ADD THE CHARACTER TO THE TOKEN STRING                          PRE10650
*                                                                                  PRE10660
            EOSPAT<1> OLDCHAR INCHAR                             :F(NOTEOS2CHARS)  PRE10670
            TOKEN = TOKEN INCHAR                                 :(EOSFOUND)       PRE10680
*                                                                                  PRE10690
*     BACKUP BY RESTORING THE OLD CHARACTER AND COLUMN NUMBER;                     PRE10700
*     BRANCH TO SEARCH FOR SOMETHING ELSE                                          PRE10710
*                                                                                  PRE10720
NOTEOS2CHARS INCHAR = OLDCHAR                                                      PRE10730
            CHARCOL = OLDCOL                                                       PRE10740
            CCLNUM = OLDCOL                                      :(NOTEOSSCAN)     PRE10750
*                                                                                  PRE10760
*                                                                                  PRE10770
*     END OF STATEMENT MARKED BY COLUMN                                            PRE10780
*                                                                                  PRE10790
*     END OF STATEMENT FOUND IF COLUMN NUMBER IS THAT SPECIFIED                    PRE10800
*         IN DEFINITION; OTHERWISE GO SEARCH FOR SOMETHING ELSE                    PRE10810
*                                                                                  PRE10820
EOSCOL      EQ(COLNUM,EOSPAT<1>)          :S(EOSCOLFOUND)F(NOTEOSSCAN)             PRE10830
*                                                                                  PRE10840
*     END OF STATEMENT FOUND BY COLUMN NUMBER;                                     PRE10850
*         READ IN ANOTHER CARD AND LOOK FOR THE CONTINUATION                       PRE10860
*         SPECIFICATION                                                            PRE10870
*                                                                                  PRE10880
EOSCOLFOUND EQ(GETEND,1)                                         :S(EOSFOUND)      PRE10890
            NEXTCARD = INPUT                                     :S(READACARD)     PRE10900
            ENDFILE = 1                                          :(EOSFOUND)       PRE10910
*                                                                                  PRE10920
*     BRANCH TO SEARCH FOR SOMETHING ELSE IF IT IS FOUND THAT                      PRE10930
*         THE STATEMENT CONTINUES ON THE NEXT CARD                                 PRE10940
*                                                                                  PRE10950
READACARD   SUBSTR(NEXTCARD,CONTPAT<2>,SIZE(CONTPAT<1>))                          PRE10960
```

```
+                              CONTPAT<1>                    :S(NOTLANLABSEP)   PRE10970
*                                                                               PRE10980
*         ADD THE TOKEN STRING TO THE CURRENT STMT STRING                       PRE10990
*                                                                               PRE11000
EOSFOUND    CURSTMT = CURSTMT TOKEN                                             PRE11010
*                                                                               PRE11020
*     TEST FOR A KEYWORD ENDING DIRECTLY BEFORE THE END OF STATEMENT            PRE11030
*         SYMBOL OR A KEYWORD ENDING IN THE END OF STATEMENT COLUMN             PRE11040
*         OR IN THE COLUMN BEFORE A BLANK IN THE END OF STATEMENT COLUMN        PRE11050
*                                                                               PRE11060
            EOSPAT<0> 'COL'                                 :S(COLKEYWORD)      PRE11070
            TESTSTR = SUBSTR(TOKEN,1,SIZE(TOKEN) - SIZE(EOSPAT<1>))             PRE11080
+                                                           :(TESTKEY)          PRE11090
COLKEYWORD  TESTSTR = TOKEN                                                     PRE11100
TESTKEY     TESTSTR KEYWORDS . KEYFOUND RPOS(0)             :S(KEYEOS)          PRE11110
            TESTSTR KEYWORDS . KEYFOUND ' ' RPOS(0)         :F(EOSIMMED)        PRE11120
KEYEOS      WORKING = WORKING ' ' KEYFOUND                                      PRE11130
            TOKEN =                                                             PRE11140
*                                                                               PRE11150
*     EOSNEXT IS SET SO AN EOS WILL BE SIGNALLED IMMEDIATELY UPON THE           PRE11160
*         NEXT CALL OF SCAN; RETURN                                             PRE11170
*                                                                               PRE11180
            EOSNEXT = 1                                     :(RETURN)           PRE11190
*                                                                               PRE11200
**********************************************************************          PRE11201
*                                                                               PRE11202
*     END OF STATEMENT HAS BEEN FOUND                                           PRE11203
*                                                                               PRE11204
**********************************************************************          PRE11205
*                                                                               PRE11206
*     STMTEOS IS SET FOR THIS STATEMENT AND EOSNEXT IS SET TO 0                 PRE11210
*                                                                               PRE11220
EOSIMMED    STMTEOS = 1                                                         PRE11230
            EOSNEXT = 0                                                         PRE11240
*                                                                               PRE11250
*     ADD THE 'UNIDENTIFIED TO END OF STATEMENT' SYMBOL TO                      PRE11260
*         THE WORKING STRING FOR THE PARSER AND RETURN                          PRE11270
*                                                                               PRE11280
ADDUEOS     WORKING = WORKING ' UEOS'                                           PRE11290
            TOKEN =                                         :(RETURN)           PRE11300
*                                                                               PRE11310
**********************************************************************          PRE11311
*                                                                               PRE11312
*     NOT AN END OF STATEMENT; LOOK FOR A LANGUAGE LABEL IF AT                  PRE11320
*         BEGINNING OF STATEMENT; ANYTHING IN CURSTMT FOLLOWED                  PRE11330
*         BY A BLANK OR JUST BLANKS INDICATES A LABEL HAS BEEN PROCESSED        PRE11340
*         ALREADY                                                               PRE11350
*                                                                               PRE11360
```

```
*********************************************************************PRE11361
*                                                                    PRE11362
NOTEOSSCAN CURSTMT SPAN(' ') BREAK(' ')                :S(NOTLANLABSEP)  PRE11370
           CURSTMT BREAK(' ')                          :S(NOTLANLABSEP)  PRE11380
           LANLABPAT<2> 'STR'                          :S(STRLABSEP)     PRE11390
*                                                                    PRE11400
*     ERROR IF LANGLABEL DEFINITION DOES NOT SPECIFY EITHER          PRE11410
*         A STRING OR A COLUMN AS AN ENDING                          PRE11420
*                                                                    PRE11430
           LANLABPAT<2> 'COL'                          :F(LANLABDEFNERR) PRE11440
*                                                                    PRE11450
*     LANGLABEL ENDED BY COLUMN                                      PRE11460
*                                                                    PRE11470
*     BRANCH IF A SEQUENCE OF COLUMN NUMBERS IS NOT SPECIFIED        PRE11480
*         AS A LANGLABEL ENDER                                       PRE11490
*                                                                    PRE11500
           EQ(SIZE(LANLABPAT<7>),0)                :S(ONECOLLANLABEND)  PRE11510
*                                                                    PRE11520
*     MAY BE A LANGLABEL ENDER IF THE COLUMN NUMBER IS LESS THAN OR  PRE11530
*         EQUAL THE UPPER LIMIT OF THE SPECIFIED SEQUENCE            PRE11540
*                                                                    PRE11550
           LE(COLNUM,LANLABPAT<7>)                     :F(NOTLANLABSEP)  PRE11560
*                                                                    PRE11570
*     MAY BE A LANGLABEL ENDER IF THE COLUMN NUMBER IS GREATER THAN OR  PRE11580
*         EQUAL THE LOWER LIMIT OF THE SPECIFIED SEQUENCE; OTHERWISE IT  PRE11590
*         IS NOT A LANGLABEL ENDER                                   PRE11600
*                                                                    PRE11610
           GE(COLNUM,LANLABPAT<6>)     :S(CKLANLABSEP)F(NOTLANLABSEP)  PRE11620
*                                                                    PRE11630
*     IF LANGLABELS END IN ONLY ONE COLUMN, AND THIS                 PRE11640
*         COLUMN NUMBER IS THE RIGHT ONE, THIS MAY BE A LANGLABEL    PRE11650
*         ENDER                                                      PRE11660
*                                                                    PRE11670
ONECOLLANLABEND EQ(COLNUM,LANLABPAT<6>)  :S(CKLANLABSEP)F(NOTLANLABSEP)  PRE11680
*                                                                    PRE11690
*     A SYMBOL STRING SEPARATES A LANGUAGE LABEL FROM A              PRE11700
*         STATEMENT; BRANCH IF THIS IS NOT THE RIGHT CHARACTER       PRE11710
*         TO BEGIN A LABEL SEPARATOR                                 PRE11720
*                                                                    PRE11730
STRLABSEP  LANLABPAT<6> INCHAR                         :F(NOTLANLABSEP)  PRE11740
           SUBSTR(TOKEN,1,SIZE(TOKEN) - 1) NOTANY(' ') BREAK(' ')  PRE11750
+                SPAN(' ') NOTANY(' ')                 :S(NOTLANLABSEP)  PRE11760
*                                                                    PRE11770
*     THIS IS A LABEL SEPARATOR IF ONLY ONE SYMBOL IS REQUIRED       PRE11780
*                                                                    PRE11790
           STRINGSEP = 1                                            PRE11800
           EQ(SIZE(LANLABPAT<6>),1)                    :S(CKLANLABSEP)  PRE11810
*                                                                    PRE11820
```

```
*     ERROR IF THE LABEL SEPARATOR STRING IS LONGER THAN 2                    PRE11830
*         CHARACTERS                                                          PRE11840
*                                                                             PRE11850
          EQ(SIZE(LANLABPAT<6>),2)                        :F(LANLABDEFNERR)   PRE11860
*                                                                             PRE11870
*     SAVE THE CURRENT CHARACTER AND COLUMN NUMBER                            PRE11880
          OLDCOL = CHARCOL                                                    PRE11890
          OLDCHAR = INCHAR                                                    PRE11900
*                                                                             PRE11910
*     GET THE NEXT USEABLE CHARACTER AND ITS COLUM NUMBER                     PRE11920
*                                                                             PRE11930
          GETCHAR('INCHAR','CHARCOL')                     :F(ENDOFINPUT)      PRE11940
*                                                                             PRE11950
*     BRANCH IF THE 2 CHARACTERS ARE NOT A LANGLABEL                          PRE11960
*         SEPARATOR; OTHERWISE ADD THE CHARACTER TO THE                       PRE11970
*         TOKEN STRING AND SET THE BACKUP FLAG TO 1 TO INDICATE THAT          PRE11980
*         BACKUP MUST BE DONE LATER IF THE ITEM PRECEDING THE SEPARATOR       PRE11990
*         IS NOT A LABEL                                                      PRE12000
*                                                                             PRE12010
          LANLABPAT<6> OLDCHAR INCHAR                     :F(NOTLANLAB2CHAR)  PRE12020
          TOKEN = TOKEN INCHAR                                                PRE12030
          BACKUP = 1                                      :(CKLANLABSEP)      PRE12040
*                                                                             PRE12050
*     BACKUP BY RESTORING THE OLD CHARACTOR AND COLUMN NUMBER                 PRE12060
*                                                                             PRE12070
NOTLANLAB2CHAR INCHAR = OLDCHAR                                               PRE12080
          CHARCOL = OLDCOL                                                    PRE12090
          CCLNUM = OLDCOL                                 :(NOTLANLABSEP)     PRE12100
*                                                                             PRE12110
*     A LANGUAGE LABEL ENDER HAS BEEN FOUND; ADD THE 'LABEL' ITEM TO          PRE12120
*         THE WORKING STRING FOR THE PARSER                                   PRE12130
*                                                                             PRE12140
*                                                                             PRE12150
*     EXTRACT THE 'LABELNAME' FROM THE TOKEN STRING                           PRE12160
*                                                                             PRE12170
CKLANLABSEP TOKEN SPAN(' ')  @LABSTART REM . LABELNAME :S(CKLABELCONT)         PRE12180
          LABELNAME = TOKEN                                                   PRE12190
          LABSTART = 0                                                        PRE12200
*                                                                             PRE12210
*     COMPUTE THE COLUMN NUMBER THE LABEL ACTUALLY STARTS IN                  PRE12220
*                                                                             PRE12230
CKLABELCONT LABSTART = STARTCOL + LABSTART                                     PRE12240
*                                                                             PRE12250
*     BRANCH IF THE LABEL MUST START WITHIN A SEQUENCE OF COLUMNS             PRE12260
*                                                                             PRE12270
          NE(LANLABPAT<1>,1)                              :S(SEQUFCOLS)       PRE12280
*                                                                             PRE12290
*      IF THE LABEL STARTS IN THE CORRECT SINGLE COLUMN, BRANCH BECAUSE       PRE12300
```

```
*         A LABEL HAS BEEN FOUND; OTHERWISE, BRANCH BECAUSE THIS IS       PRE12310
*         NOT A LABEL                                                     PRE12320
*                                                                         PRE12330
           EQ(LANLABPAT<4>,LABSTART)                                      PRE12340
+                                        :S(FOUNDLABEL)F(NOLABEL)         PRE12350
*                                                                         PRE12360
*     ERROR IF THE LANGUAGE LABEL DEFINITION SPECIFIES OTHER THAN         PRE12370
*         A SINGLE COLUMN OR A SEQUENCE OF COLUMNS (1 OR 2 IN             PRE12380
*         LANLABPAT<1>) FOR A LABEL BEGINNING                             PRE12390
*                                                                         PRE12400
SEQOFCOLS  EQ(LANLABPAT<1>,2)                          :F(LANLABDEFNERR)  PRE12410
*                                                                         PRE12420
*     NOT A LABEL IF THE LABEL STARTS IN A COLUMN WHICH IS GREATER THAN   PRE12430
*         THE UPPER BOUND OF THE SEQUENCE OR LESS THAN THE LOWER BOUND    PRE12440
*                                                                         PRE12450
           GT(LABSTART,LANLABPAT<5>)                   :S(NOLABEL)       PRE12460
           LT(LABSTART,LANLABPAT<4>)                   :S(NOLABEL)       PRE12470
*                                                                         PRE12480
*     A LABEL HAS BEEN FOUND IF A STRING TERMINATED THE LABEL             PRE12490
*                                                                         PRE12500
           LANLABPAT<2> 'STR' RPOS(0)                  :S(FOUNDLABEL)    PRE12510
*                                                                         PRE12520
*     CHECK THE FIRST CHARACTER OF THE LABEL TO SEE IF IT                 PRE12530
*         IS VALID ACCORDING TO THE LANGLABEL DEFINITION; IF NOT,         PRE12540
*         THIS IS NOT A LABEL                                             PRE12550
*                                                                         PRE12560
           SUBSTR(LABELNAME,1,1) ANY(LANLABPAT<8>)     :F(NOLABEL)       PRE12570
*                                                                         PRE12580
*     CHECK THE REMAINING CHARACTERS OF THE LABEL TO SEE IF THEY          PRE12590
*         ARE VALID ACCORDING TO THE LANGLABEL DEFINTION; IF NOT,         PRE12600
*         THIS IS NOT A LABEL                                             PRE12610
*                                                                         PRE12620
           EQ(SIZE(LABELNAME),1)                       :S(FOUNDLABEL)    PRE12630
           LABELNAME LEN(1) SPAN(LANLABPAT<9>) RPOS(0)                   PRE12640
+                                        :F(NOLABEL)S(FOUNDLABEL)         PRE12650
*                                                                         PRE12660
*     THIS IS NOT A LABEL; IF NECESSARY, BACKUP SHOULD BE SET TO 0 AND    PRE12670
*         A BRANCH TAKEN TO A LABEL WHERE BACKUP WILL TAKE PLACE          PRE12680
*                                                                         PRE12690
NOLABEL    NE(BACKUP,1)                                :S(NOTLANLABSEP)  PRE12700
           BACKUP = 0                                  :(NOTLANLAB2CHAR) PRE12710
FOUNDLABEL EQ(STRINGSEP,1)                             :S(LABELCOMMENT)  PRE12720
*                                                                         PRE12730
*     IF COLUMN WAS LABEL INDICATOR AND NO CHARACTER EXISTS BEYOND        PRE12740
*         THE COLUMN (GETCHAR RETURNS WITH FAILURE), IT IS NOT A LABEL    PRE12750
*                                                                         PRE12760
           OLDCOL = CHARCOL                                              PRE12770
           OLDCHAR = INCHAR                                              PRE12780
```

```
            GETCHAR('INCHAR','CHARCOL')                          :F(BACKUPNOLAB)   PRE12790
*                                                                                  PRE12800
*     IF THE NEXT CHARACTER IS A LABEL DELIMITER (BLANK OR SPECIAL                 PRE12810
*        CHARACTER), THIS IS A LABEL                                               PRE12820
*                                                                                  PRE12830
            INCHAR LABDELS                                       :S(LABELYES)      PRE12840
*                                                                                  PRE12850
*     BACKUP BY RESTORING THE OLD CHARACTER AND ITS COLUMN NUMBER;                 PRE12860
*        BRANCH BASED ON THE FINDING OF A LABEL OR NOT                             PRE12870
*                                                                                  PRE12880
BACKUPNOLAB INCHAR = OLDCHAR                                                      PRE12890
            CHARCOL = OLDCOL                                                      PRE12900
            COLNUM = OLDCOL                                      :(NOTLANLABSEP)  PRE12910
LABELYES    INCHAR = OLDCHAR                                                      PRE12920
            CHARCOL = OLDCOL                                                      PRE12930
            COLNUM = OLDCOL                                      :(LABELCOMMENT)  PRE12940
*                                                                                  PRE12950
*******************************************************************************  PRE12951
*                                                                                  PRE12952
*     A LABEL HAS BEEN FOUND                                                       PRE12953
*                                                                                  PRE12954
*******************************************************************************  PRE12955
*                                                                                  PRE12956
*     INSERT A COMMENT THAT THE LABEL NAME HAS BEEN PLACED ON                      PRE12960
*        A NULL STATEMENT                                                          PRE12970
*                                                                                  PRE12980
LABELCOMMENT INSERTCOM(LABELNAME ' HAS BEEN MADE THE LABEL ON A NULL '            PRE12990
+                                'STATEMENT.')                                     PRE13000
            STRINGSEP = 0                                                         PRE13010
*                                                                                  PRE13020
*     PLACE ENOUGH BLANKS IN THE CURRENT STMT STRING TO                            PRE13030
*        TAKE THE PLACE OF THE LABEL JUST FOUND                                    PRE13040
*                                                                                  PRE13050
            CURSTMT = CURSTMT DUPL(' ',SIZE(TOKEN))                               PRE13060
            WORKING = WORKING ' LABEL'                                            PRE13070
            TOKEN =                                              :(RETURN)        PRE13080
*                                                                                  PRE13090
*******************************************************************************  PRE13091
*                                                                                  PRE13092
*     NOT A LANGUAGE LABEL ENDER; IS IT A BLANK?                                   PRE13100
*                                                                                  PRE13110
*******************************************************************************  PRE13111
*                                                                                  PRE13112
*     IF NOT A BLANK CHARACTER, GO BACK TO THE BEGINNING OF THE                    PRE13120
*        SCAN ROUTINE                                                              PRE13130
*                                                                                  PRE13140
NOTLANLABSEP INCHAR ' '                          :S(KEYWORDSRCH)F(BEGSCAN1)       PRE13150
*                                                                                  PRE13160
```

```
*     LOOK FOR A KEYWORD APPEARING IMMEDIATELY BEFORE THIS                    PRE13170
*          BLANK; BRANCH TO THE UNIDENTIFIED SECTION IF A                     PRE13180
*          KEYWORD IS NOT FOUND                                               PRE13190
*                                                                             PRE13200
KEYWORDSRCH TOKEN KEYWORDS . KEYFOUND ' ' RPOS(0)                            PRE13210
+                                                  :S(ISKEYWORD)             PRE13220
+            TOKEN KEYWORD2 . KEYFOUND ' ' RPOS(0) :F(UNIDENT)               PRE13230
*                                                                             PRE13240
*************************************************************************PRE16241
*                                                                             PRE13242
*     A KEYWORD HAS BEEN FOUND                                               PRE13243
*                                                                             PRE13244
*************************************************************************PRE13245
*                                                                             PRE13246
*     IF THE KEYWORD MATCHED IS 'THEN', ADD 'UTHEN' TO THE WORKING           PRE13250
*          STRING FOR                                                         PRE13260
*          THE PARSER; BRANCH TO ADD THE TOKEN STRING TO THE CURRENT          PRE13270
*          STATEMENT STRING                                                   PRE13280
*                                                                             PRE13290
ISKEYWORD  KEYFOUND KEYTHEN                        :F(NOTATHEN)             PRE13300
           WORKING = WORKING ' UTHEN'              :(ADDTOCURSTMT)          PRE13310
*                                                                             PRE13320
*     THE KEYWORD MATCHED IS NOT 'THEN'; ADD THE KEYWORD                     PRE13330
*          TO THE WORKING STRING FOR THE PARSER                               PRE13340
*                                                                             PRE13350
NOTATHEN   WORKING = WORKING ' ' KEYFOUND                                    PRE13360
*                                                                             PRE13370
*     ADD THE TOKEN STRING TO THE CURRENT STATEMENT STRING AND RETURN        PRE13380
*                                                                             PRE13390
ADDTOCURSTMT CURSTMT = CURSTMT TOKEN                                         PRE13400
           TOKEN =                                 :(RETURN)                 PRE13410
*                                                                             PRE13420
*************************************************************************PRE13421
*                                                                             PRE13422
*     UNIDENTIFIED-TO-END-OF-STATEMENT (UEOS) HAS BEEN FOUND                 PRE13423
*                                                                             PRE13424
*************************************************************************PRE13425
*                                                                             PRE13426
*     THE ITEM LOCATED IS UNIDENTIFIED; GO BACK TO THE BEGINNING OF          PRE13430
*          THE SCANNING ROUTINE TO FIND MORE OF THE TOKEN SO                  PRE13440
*          IT MIGHT BE IDENTIFIED                                             PRE13450
*                                                                             PRE13460
UNIDENT                                            :(BEGSCAN1)               PRE13470
*                                                                             PRE13480
*************************************************************************PRE13481
*                                                                             PRE13482
*     CHECK FOR ERROR AT END OF FILE                                         PRE13483
*                                                                             PRE13484
```

```
*********************************************************************PRE13485
*                                                                    PRE13486
*     END OF FILE WAS FOUND WHEN TRYING TO GET A CHARACTER AT THE    PRE13490
*        BEGINNING OF THE SCAN ROUTINE (NOT WHEN LOOKING FOR A       PRE13500
*     SECOND CHARACTER OF A STRING); IF THE TOKEN IS ALL BLANK OR NULL,PRE13510
*        TEST IF AN EOS HAS BEEN FOUND BEFORE THIS CALL TO SCAN      PRE13520
*                                                                    PRE13530
ENDOFINPUT TOKEN POS(0) SPAN(' ') RPOS(0)          :S(TESTEOS)      PRE13540
           EQ(SIZE(TOKEN),0)                       :F(ERREND)       PRE13550
TESTEOS    EQ(OLDSTMTEOS,1)                        :F(ERREND)       PRE13560
*                                                                    PRE13570
*     AN EOS HAD BEEN FOUND; ADD END-OF-FILE TO WORKING STRING FOR   PRE13580
*        PARSER AND RETURN WITH FAILURE                              PRE13590
*                                                                    PRE13600
ADDEOF     WORKING = WORKING ' EOF'                :(FRETURN)       PRE13610
GETCHAR    COLNUM = COLNUM + 1                                      PRE13620
*                                                                    PRE13630
*********************************************************************PRE13631
*                                                                    PRE13632
*     ROUTINE TO GET THE NEXT CHARACTER FROM A NON-IGNORED COLUMN;   PRE13633
*        RETURN THE CHARACTER AND ITS COLUMN NUMBER                  PRE13634
*                                                                    PRE13635
*     TWO PARAMETERS -- CHAR AND COL, THE NAMES (IN STRING FORM)     PRE13636
*        OF THE VARIABLES TO HOLD THE RETURN VALUES                  PRE13637
*                                                                    PRE13638
*********************************************************************PRE13639
*                                                                    PRE13640
*     INCREMENT COLUMN NUMBER MODULO 80                              PRE13641
*                                                                    PRE13650
          LE(COLNUM,80)                            :S(NOINPUT)      PRE13660
          COLNUM = 1                                                PRE13670
*                                                                    PRE13680
*     READ A NEW CARD IF ONE HAS NOT BEEN SCANNED FOR CONTINUATION   PRE13690
*        OTHERWISE USE THE ONE THAT WAS SCANNED                      PRE13700
*                                                                    PRE13710
          EQ(SIZE(NEXTCARD),0)                     :F(COPYNEXTCARD) PRE13720
*                                                                    PRE13730
*     IF THE SCANNER HAS REACHED END OF FILE PREVIOUSLY OR THIS      PRE13740
*        ROUTINE HAS REACHED END OF FILE PREVIOUSLY, RETURN WITH FAILUREPRE13750
*                                                                    PRE13760
          EQ(ENDFILE,1)                            :S(FRETURN)      PRE13770
          EQ(GETEND,1)                             :S(FRETURN)      PRE13780
*                                                                    PRE13790
*     READ THE CARD AND BRANCH                                       PRE13800
*                                                                    PRE13810
          CARDIMAGE = INPUT                        :S(NOINPUT)      PRE13820
*                                                                    PRE13830
*     SET THE FLAG IF JUST NOW REACHED END OF FILE AND               PRE13840
```

```
*          RETURN WITH FAILURE                                                  PRE13850
*                                                                               PRE13860
            GETEND = 1                                    :(FRETURN)            PRE13870
COPYNEXTCARD CARDIMAGE = NEXTCARD                                               PRE13880
            NEXTCARD =                                                          PRE13890
*                                                                               PRE13900
*     BRANCH TO GET THE NEXT CHARACTER IF THIS IS AN IGNORED                    PRE13910
*          COLUMN                                                               PRE13920
*                                                                               PRE13930
NOINPUT     EQ(COLNUM,IGNPAT<1>)                          :S(GETCHAR)           PRE13940
            IGNPAT<0> 'SEQ'                               :F(NOTIGNORED)        PRE13950
            LE(COLNUM,IGNPAT<2>)                          :F(NOTIGNORED)        PRE13960
            GE(COLNUM,IGNPAT<1>)                          :S(GETCHAR)           PRE13970
*                                                                               PRE13980
*     RETURN ONE CHARACTER AND ITS COLUMN NUMBER                                PRE13990
*                                                                               PRE14000
NOTIGNORED CARDIMAGE TAB(COLNUM - 1) LEN(1) . $CHAR                             PRE14010
            $COL = COLNUM                                 :(RETURN)             PRE14020
SCANEOSDEF EOSDEF "'" ARB . EOSPAT<1> "'" =               :S(SYMSTR)            PRE14030
*                                                                               PRE14040
*****************************************************************************PRE14041
*                                                                               PRE14042
*     ROUTINE TO SCAN DEFINITION OF END OF STATEMENT                            PRE14050
*                                                                               PRE14060
*     ONE PARAMETER -- EOSDEF                                                   PRE14070
*                                                                               PRE14080
*****************************************************************************PRE14081
*                                                                               PRE14082
            EOSPAT<1> = EOSDEF                                                  PRE14090
*                                                                               PRE14100
*     TEST FOR COLUMN SPECIFICATION CONTAINING ONLY DIGITS                      PRE14110
*                                                                               PRE14120
            EOSPAT<1> SPAN('0123456789') RPOS(0)          :F(EOSERR)            PRE14130
*                                                                               PRE14140
*     SPECIFICATION INDICATES A COLUMN FOR END OF STATEMENT                     PRE14150
*                                                                               PRE14160
            EOSPAT<0> = 'COL'                             :(ENDEOSSCAN)         PRE14170
SYMSTR      EQ(SIZE(EOSDEF),0)                            :F(EOSERR)            PRE14180
*                                                                               PRE14190
*     SPECIFICATION INDICATES A SYMBOL STRING FOR END OF STATEMENT              PRE14200
*                                                                               PRE14210
            EOSPAT<0> = 'STR'                                                   PRE14220
ENDEOSSCAN SCANEOSDEF =                                   :(RETURN)             PRE14230
SCANIGNCOLDEF IGNCOLDEF ARB . IGNPAT<1> '-' ARB . IGNPAT<2> RPOS(0) =           PRE14240
*                                                         :F(ONEIGN)            PRE14250
*                                                                               PRE14260
*****************************************************************************PRE14261
*                                                                               PRE14262
```

```
*     ROUTINE TO SCAN DEFINITION OF IGNORE COLUMNS                              PRE14270
*                                                                               PRE14280
*     ONE PARAMETER -- IGNCOLDEF                                                PRE14290
*                                                                               PRE14300
*     TEST FOR COLUMN SPECIFICATIONS CONTAINING ONLY DIGITS                     PRE14310
*                                                                               PRE14320
*****************************************************************************PRE14321
*                                                                               PRE14322
          IGNPAT<1> SPAN('0123456789') RPOS(0)          :F(IGNERR)              PRE14330
          IGNPAT<2> SPAN('0123456789') RPOS(0)          :F(IGNERR)              PRE14340
*                                                                               PRE14350
*     SPECIFICATION INDICATES A SEQUENCE OF COLUMNS TO BE IGNORED               PRE14360
*                                                                               PRE14370
          IGNPAT<0> = 'SEQ'                             :(ENDIGNCOLSCAN)        PRE14380
ONEIGN    IGNPAT<1> = IGNCOLDEF                                                 PRE14390
*                                                                               PRE14400
*     TEST FOR COLUMN SPECIFICATION CONTAINING ONLY DIGITS                      PRE14410
*                                                                               PRE14420
          IGNPAT<1> SPAN('0123456789') RPOS(0)          :F(IGNERR)              PRE14430
*                                                                               PRE14440
*     SPECIFICATION INDICATES A SINGLE COLUMN TO BE IGNORED                     PRE14450
*                                                                               PRE14460
          IGNPAT<0> = 'ONE'                                                     PRE14470
ENDIGNCOLSCAN SCANIGNCOLDEF =                           :(RETURN)               PRE14480
SCANCOMDEF                                                                      PRE14490
*                                                                               PRE14500
*****************************************************************************PRE14501
*                                                                               PRE14502
*     ROUTINE TO SCAN DEFINITION OF COMMENT                                     PRE14510
*                                                                               PRE14520
*     ONE PARAMETER -- COMDEF                                                   PRE14530
*                                                                               PRE14540
*****************************************************************************PRE14541
*                                                                               PRE14542
          COMDEF "'" BREAK("'") . COMPAT<3> "'" SPAN(' ') =                     PRE14550
+                                                       :F(OPENCOL)             PRE14560
*                                                                               PRE14570
*     SPECIFICATION INDICATES A SYMBOL FOR COMMENT OPENER                       PRE14580
*                                                                               PRE14590
          I = '10'                                                              PRE14600
          COMDEF '+' SPAN(' ') BREAK(' ') . COMPAT<4> SPAN(' ') =               PRE14610
+                                                       :F(NOOPENCOL)           PRE14620
*                                                                               PRE14630
*     TEST FOR COLUMN SPECIFICATION CONTAINING ONLY DIGITS                      PRE14640
*                                                                               PRE14650
          COMPAT<4> SPAN('0123456789') RPOS(0)          :F(COMERROR)            PRE14660
*                                                                               PRE14670
*     SPECIFICATION INDICATES A SYMBOL AND A COLUMN NUMBER FOR OPENER           PRE14680
```

```
*                                                                       PRE14690
            I = '11'                                    :(COMENDER)     PRE14700
NOOPENCOL   COMPAT<4> =                                 :(COMENDER)     PRE14710
OPENCOL     COMPAT<3> =                                                 PRE14720
*                                                                       PRE14730
*     SPECIFICATION INDICATES ONLY A COLUMN NUMBER FOR COMMENT OPENER   PRE14740
*                                                                       PRE14750
            I = '01'                                                    PRE14760
            COMDEF BREAK(' ') . COMPAT<4> SPAN(' ') =   :F(COMERROR)    PRE14770
*                                                                       PRE14780
*     TEST FOR COLUMN SPECIFICATION CONTAINING ONLY DIGITS              PRE14790
*                                                                       PRE14800
            COMPAT<4> SPAN('0123456789') RPOS(0)        :F(COMERROR)    PRE14810
*                                                                       PRE14820
*     SAVE COMMENT OPENER 'DESCRIPTOR'                                  PRE14830
*                                                                       PRE14840
COMENDER    COMPAT<1> = I                                               PRE14850
            I = '10'                                                    PRE14860
            COMDEF "'" BREAK("'") . COMPAT<5> "'" =     :F(ENDCOL)      PRE14870
            COMDEF SPAN(' ') =                          :F(NOCLOSECOL)  PRE14880
*                                                                       PRE14890
*     SPECIFICATION INDICATES A SYMBOL FOR COMMENT ENDER                PRE14900
*                                                                       PRE14910
            COMDEF '+' SPAN(' ') REM . COMPAT<6> =      :F(NOCLOSECOL)  PRE14920
*                                                                       PRE14930
*     TEST FOR COLUMN SPECIFICATION CONTAINING ONLY DIGITS              PRE14940
*                                                                       PRE14950
            COMPAT<6> SPAN('0123456789') RPOS(0)        :F(COMERROR)    PRE14960
*                                                                       PRE14970
*     SPECIFICATION INDICATES A SYMBOL AND A COLUMN NUMBER FOR ENDER    PRE14980
*                                                                       PRE14990
            I = '11'                                    :(ENDCOMSCAN)   PRE15000
NOCLOSECOL  COMPAT<6> =                                 :(ENDCOMSCAN)   PRE15010
ENDCOL      COMPAT<5> =                                                 PRE15020
*                                                                       PRE15030
*     SPECIFICATION INDICATES ONLY A COLUMN NUMBER FOR COMMENT CLOSER   PRE15040
*                                                                       PRE15050
            I = '01'                                                    PRE15060
            COMDEF REM . COMPAT<6> =                    :F(COMERROR)    PRE15070
*                                                                       PRE15080
*     TEST FOR COLUMN SPECIFICATION CONTAINING ONLY DIGITS              PRE15090
*                                                                       PRE15100
            COMPAT<6> SPAN('0123456789') RPOS(0)        :F(COMERROR)    PRE15110
*                                                                       PRE15120
*     SAVE COMMENT ENDER 'DESCRIPTOR'                                   PRE15130
*                                                                       PRE15140
ENDCOMSCAN  COMPAT<2> = I                                               PRE15150
            SCANCOMDEF =                                :(RETURN)       PRE15160
```

```
SCANCONTDEF CONTDEF BREAK(' ') . CONTPAT<1> SPAN(' ') =                    PRE15170
+                                                    :F(NOCONT)            PRE15180
*                                                                          PRE15190
*     ROUTINE TO SCAN DEFINITION OF CONTINUATION                           PRE15200
*                                                                          PRE15210
*     ONE PARAMETER -- CONTDEF                                             PRE15220
*                                                                          PRE15230
          CONTPAT<2> = CONTDEF                                             PRE15240
          CONTPAT<2> SPAN('0123456789') RPOS(0)      :F(CONTERR)           PRE15250
          CONTDEF =                                  :(ENDCONTSCAN)        PRE15260
NOCONT    CONTPAT<1> = CONTDEF                                             PRE15270
          CONTDEF 'NONE' RPOS(0)                     :F(CONTERR)           PRE15280
ENDCONTSCAN SCANCONTDEF =                            :(RETURN)             PRE15290
SCANLITDEF LITDEF BREAK(' ') . LITPAT<1> SPAN(' ') =  :F(LITERROR)         PRE15300
*                                                                          PRE15310
**********************************************************************    PRE15311
*                                                                          PRE15312
*     ROUTINE TO SCAN DEFINITION OF LITERAL                                PRE15320
*                                                                          PRE15330
*     ONE PARAMETER -- LITDEF                                              PRE15340
*                                                                          PRE15350
**********************************************************************    PRE15351
*                                                                          PRE15352
          LITDEF BREAK(' ') . LITPAT<2> SPAN(' ') =  :F(NOSPECIAL)         PRE15360
          LITDEF REM . LITPAT<3> =                   :(ENDLITSCAN)         PRE15370
NOSPECIAL LITDEF REM . LITPAT<2>                                           PRE15380
          LITPAT<3> =                                                      PRE15390
ENDLITSCAN SCANLITDEF =                              :(RETURN)             PRE15400
SCANLANLABDEF                                                              PRE15410
*                                                                          PRE15420
**********************************************************************    PRE15421
*                                                                          PRE15422
*     ROUTINE TO SCAN DEFINITION OF LANGUAGE LABEL                         PRE15430
*                                                                          PRE15440
*     ONE PARAMETER -- LANLABDEF                                           PRE15450
*                                                                          PRE15460
**********************************************************************    PRE15461
*                                                                          PRE15462
          LANLABDEF BREAK(' ') . SPEC =              :F(LANLABERR)         PRE15470
          SPEC BREAK('-') . LANLABPAT<4> '-' REM . LANLABPAT<5> =          PRE15480
+                                                    :F(ONECOLOPEN)        PRE15490
*                                                                          PRE15500
*     SPECIFICATION INDICATES A SEQUENCE OF COLUMNS TO START LABEL         PRE15510
*                                                                          PRE15520
*     TEST FOR COLUMN SPECIFICATIONS CONTAINING ONLY DIGITS                PRE15530
*                                                                          PRE15540
          LANLABPAT<4> SPAN('0123456789') RPOS(0)    :F(LANLABERR)         PRE15550
          LANLABPAT<5> SPAN('0123456789') RPOS(0)    :F(LANLABERR)         PRE15560
```

```
            LANLABPAT<1> = 2                                    :(CLOSELANLAB)      PRE15570
*                                                                                   PRE15580
*     TEST FOR COLUMN SPECIFICATION CONTAINING ONLY DIGITS                          PRE15590
*                                                                                   PRE15600
ONECOLOPEN  SPEC SPAN('0123456789') RPOS(0)                     :F(LANLABERR)       PRE15610
*                                                                                   PRE15620
*     SPECIFICATION INDICATES A COLUMN TO START LABEL                               PRE15630
*                                                                                   PRE15640
            LANLABPAT<1> = 1                                                        PRE15650
            LANLABPAT<4> = SPEC                                                     PRE15660
            LANLABPAT<5> =                                                          PRE15670
CLOSELANLAB LANLABDEF SPAN(' ') BREAK(' ') . SPEC =             :F(LANLABERR)       PRE15680
            SPEC "'" BREAK("'") . LANLABPAT<6> "'" =            :F(COLCLOSE)        PRE15690
*                                                                                   PRE15700
*     SPECIFICATION INDICATES A SYMBOL STRING TO TERMINATE LABEL                    PRE15710
*                                                                                   PRE15720
            LANLABPAT<2> = 'STR'                                                    PRE15730
            LANLABPAT<7> =                                      :(OPENCHARS)        PRE15740
COLCLOSE    LANLABPAT<2> = 'COL'                                                    PRE15750
            SPEC BREAK('-') . LANLABPAT<6> '-' REM . LANLABPAT<7> =                 PRE15760
+                                                               :F(ONECOLCLOSE)     PRE15770
*                                                                                   PRE15780
*     SPECIFICATION INDICATES A SEQUENCE OF COLUMNS TO TERMINATE A LABEL            PRE15790
*                                                                                   PRE15800
*     TEST FOR COLUMN SPECIFICATIONS CONTAINING ONLY DIGITS                         PRE15810
*                                                                                   PRE15820
            LANLABPAT<6> SPAN('0123456789') RPOS(0)             :F(LANLABERR)       PRE15830
            LANLABPAT<7> SPAN('0123456789') RPOS(0)             :F(LANLABERR)       PRE15840
            LANLABPAT<3> = 2                                    :(OPENCHARS)        PRE15850
*                                                                                   PRE15860
*     TEST FOR COLUMN SPECIFICATION CONTAINING ONLY DIGITS                          PRE15870
*                                                                                   PRE15880
ONECOLCLOSE SPEC SPAN('0123456789') RPOS(0)                     :F(LANLABERR)       PRE15890
*                                                                                   PRE15900
*     SPECIFICATION INDICATES A COLUMN TO TERMINATE LABEL                           PRE15910
*                                                                                   PRE15920
            LANLABPAT<3> = 1                                                        PRE15930
            LANLABPAT<6> = SPEC                                                     PRE15940
            LANLABPAT<7> =                                                          PRE15950
*                                                                                   PRE15960
*     DETERMINE SET OF OPENING CHARACTERS                                           PRE15970
*                                                                                   PRE15980
OPENCHARS   LANLABDEF SPAN(' ') "'" BREAK("'") . LANLABPAT<8> "'" =                 PRE15990
+                                                               :F(LANLABERR)       PRE16000
*                                                                                   PRE16010
*     DETERMINE SET OF REMAINING CHARACTERS                                         PRE16020
*                                                                                   PRE16030
            LANLABDEF SPAN(' ') "'" BREAK("'") . LANLABPAT<9>                       PRE16040
```

```
+                   "!" RPOS(0) =          :F(LANLABERR)S(RETURN)                PRE16050
STARTP          SCAN()                                        :F(PSCANERR)        PRE16060
*                                                                                 PRE16070
*********************************************************************************PRE16071
*                                                                                 PRE16072
*     STRUCTURE PARSER -- NC PARAMETERS                                           PRE16080
*                                                                                 PRE16090
*     THE WORKING STRING ACTS AS A SYNTACTIC STACK                                PRE16100
*                                                                                 PRE16110
*********************************************************************************PRE16111
*                                                                                 PRE16112
*     TRY TO MATCH A LABEL                                                        PRE16120
*                                                                                 PRE16130
START           WORKING 'LABEL' RPOS(0)                       :F(NXT.1)           PRE16140
*                                                                                 PRE16150
*     CALL SEMANTIC ROUTINE 0, SCAN, AND RETURN TO START                          PRE16160
*                                                                                 PRE16170
                ROUT0()                                                           PRE16180
*                                                                                 PRE16190
*     SCAN AND BRANCH                                                             PRE16200
*                                                                                 PRE16210
                SCAN()                                  :S(START)F(PSCANERR)      PRE16220
*                                                                                 PRE16230
*     TRY TO MATCH AN ENDCASE                                                     PRE16240
*                                                                                 PRE16250
NXT.1           WORKING 'ENDCASE' RPOS(0)                     :F(NXT.2)           PRE16260
*                                                                                 PRE16270
*     SCAN AND BRANCH                                                             PRE16280
*                                                                                 PRE16290
                SCAN()                          :S(ECASESTMT)F(PSCANERR)          PRE16300
*                                                                                 PRE16310
*     TRY TO MATCH AN ENDIF                                                       PRE16320
*                                                                                 PRE16330
NXT.2           WORKING 'ENDIF' RPOS(0)                       :F(NXT.3)           PRE16340
*                                                                                 PRE16350
*     SCAN AND BRANCH                                                             PRE16360
*                                                                                 PRE16370
                SCAN()                                  :S(EIFSTMT)F(PSCANERR)    PRE16380
*                                                                                 PRE16390
*     TRY TO MATCH AN ENDWHILE                                                    PRE16400
*                                                                                 PRE16410
NXT.3           WORKING 'ENDWHILE' RPOS(0)                    :F(NXT.4)           PRE16420
*                                                                                 PRE16430
*     SCAN AND BRANCH                                                             PRE16440
*                                                                                 PRE16450
                SCAN()                          :S(EWHILESTMT)F(PSCANERR)         PRE16460
*                                                                                 PRE16470
*     TRY TO MATCH AN ENDREPEAT                                                   PRE16480
```

```
*                                                                    PRE16490
NXT.4           WORKING 'ENDREPEAT' RPOS(0)                :F(NXT.5) PRE16500
*                                                                    PRE16510
*     SCAN AND BRANCH                                                PRE16520
*                                                                    PRE16530
                SCAN()                    :S(EREPEATSTMT)F(PSCANERR) PRE16540
*                                                                    PRE16550
*     TRY TO MATCH AN ENDDO                                          PRE16560
*                                                                    PRE16570
NXT.5           WORKING 'ENDDO' RPOS(0)                    :F(NXT.6) PRE16580
*                                                                    PRE16590
*     SCAN AND BRANCH                                                PRE16600
*                                                                    PRE16610
                SCAN()                          :S(EDOSTMT)F(PSCANERR) PRE16620
*                                                                    PRE16630
*     TRY TO MATCH AN ENDDOGROUP                                     PRE16640
*                                                                    PRE16650
NXT.6           WORKING 'ENDDOGROUP' RPOS(0)               :F(NXT.7) PRE16660
*                                                                    PRE16670
*     SCAN AND BRANCH                                                PRE16680
*                                                                    PRE16690
                SCAN()                    :S(EDOGRPSTMT)F(PSCANERR) PRE16700
*                                                                    PRE16710
*     TRY TO MATCH AN ELSE                                           PRE16720
*                                                                    PRE16730
NXT.7           WORKING 'ELSE' RPOS(0)                     :F(NXT.8) PRE16740
*                                                                    PRE16750
*     CALL SEMANTIC ROUTINE 1, SCAN AND RETURN TO START              PRE16760
*                                                                    PRE16770
                ROUT1()                                              PRE16780
                SCAN()                          :S(START)F(PSCANERR) PRE16790
*                                                                    PRE16800
*     TRY TO MATCH A QUITLOOP                                        PRE16810
*                                                                    PRE16820
NXT.8           WORKING 'QUITLOOP' RPOS(0)                 :F(NXT.9) PRE16830
*                                                                    PRE16840
*     SCAN AND BRANCH                                                PRE16850
*                                                                    PRE16860
                SCAN()                    :S(QUITSTMT)F(PSCANERR)    PRE16870
*                                                                    PRE16880
*     TRY TO MATCH A CASE                                            PRE16890
*                                                                    PRE16900
NXT.9           WORKING 'CASE' RPOS(0)                    :F(NXT.10) PRE16910
*                                                                    PRE16920
*     SCAN AND BRANCH                                                PRE16930
*                                                                    PRE16940
                SCAN()                          :S(CASESTMT)F(PSCANERR)PRE16950
*                                                                    PRE16960
```

```
*     TRY TO MATCH A DO                                                    PRE16970
*                                                                          PRE16980
NXT.10          WORKING 'DO' RPOS(0)                    :F(NXT.11)         PRE16990
*                                                                          PRE17000
*     SCAN AND BRANCH                                                      PRE17010
*                                                                          PRE17020
                SCAN()                              :S(DOSTMT)F(PSCANERR)  PRE17030
*                                                                          PRE17040
*     TRY TO MATCH A DOGROUP                                               PRE17050
*                                                                          PRE17060
NXT.11          WORKING 'DOGROUP' RPOS(0)               :F(NXT.12)         PRE17070
*                                                                          PRE17080
*     SCAN AND BRANCH                                                      PRE17090
*                                                                          PRE17100
                SCAN()                        :S(DOGRPSTMT)F(PSCANERR)     PRE17110
*                                                                          PRE17120
*     TRY TO MATCH AN IF                                                   PRE17130
*                                                                          PRE17140
NXT.12          WORKING 'IF' RPOS(0)                    :F(NXT.13)         PRE17150
*                                                                          PRE17160
*     SCAN AND BRANCH                                                      PRE17170
*                                                                          PRE17180
                SCAN()                              :S(IFSTMT)F(PSCANERR)  PRE17190
*                                                                          PRE17200
*     TRY TO MATCH A REPEAT                                                PRE17210
*                                                                          PRE17220
NXT.13          WORKING 'REPEAT' RPOS(0)                :F(NXT.14)         PRE17230
*                                                                          PRE17240
*     SCAN AND BRANCH                                                      PRE17250
*                                                                          PRE17260
                SCAN()                        :S(REPEATSTMT)F(PSCANERR)    PRE17270
*                                                                          PRE17280
*     TRY TO MATCH A WHILE                                                 PRE17290
*                                                                          PRE17300
NXT.14          WORKING 'WHILE' RPOS(0)                 :F(NXT.15)         PRE17310
*                                                                          PRE17320
*     SCAN AND BRANCH                                                      PRE17330
*                                                                          PRE17340
                SCAN()                        :S(WHILESTMT)F(PSCANERR)     PRE17350
*                                                                          PRE17360
*     TRY TO MATCH A STRUCTURE LIST AND END OF FILE; IF MATCHED,           PRE17370
*        RETURN TO CALLING ROUTINE                                         PRE17380
*                                                                          PRE17390
NXT.15          WORKING 'STRLIST EOF' = 'PROG'          :S(RETURN)         PRE17400
*                                                                          PRE17410
*     TRY TO MATCH AN UNIDENTIFIED TO END OF STATEMENT; IF NOT             PRE17420
*        MATCHED TAKE ERROR EXIT                                           PRE17430
*                                                                          PRE17440
```

```
                WORKING 'UEOS' RPOS(0) = 'STRUC'          :F(PUNIDERR)      PRE17450
*                                                                           PRE17460
*     CALL SEMANTIC ROUTINE 01 AND BRANCH TO STRUCTURE SECTION              PRE17470
*                                                                           PRE17480
                ROUT01()                                  :(STRUCTURE)      PRE17490
*                                                                           PRE17500
*     CASE HAS BEEN RECOGNIZED, REDUCE TO CASESTMT; CALL SEMANTIC           PRE17510
*          ROUTINE 2, SCAN AND RETURN TO START                              PRE17520
*                                                                           PRE17530
CASESTMT        WORKING 'CASE UEOS' RPOS(0) = 'CASESTMT' :F(PCASEERR)       PRE17540
                ROUT2()                                                     PRE17550
                SCAN()                                  :S(START)F(RETURN)  PRE17560
*                                                                           PRE17570
*     DO HAS BEEN RECOGNIZED, REDUCE TO DOSTMT; CALL SEMANTIC               PRE17580
*          ROUTINE 3, SCAN AND RETURN TO START                              PRE17590
*                                                                           PRE17600
DOSTMT          WORKING 'DO UEOS' RPOS(0) = 'DOSTMT'     :F(PDOERR)         PRE17610
                ROUT3()                                                     PRE17620
                SCAN()                                  :S(START)F(RETURN)  PRE17630
*                                                                           PRE17640
*     DOGROUP HAS BEEN RECOGNIZED, REDUCE TO DOGROUPSTMT; CALL SEMANTIC     PRE17650
*          ROUTINE 4, SCAN AND RETURN TO START                              PRE17660
*                                                                           PRE17670
DOGRPSTMT       WORKING 'DOGROUP UEOS' RPOS(0) =                            PRE17680
+                    'DOGROUPSTMT'                        :F(PDGRPERR)       PRE17690
                ROUT4()                                                     PRE17700
                SCAN()                                  :S(START)F(RETURN)  PRE17710
*                                                                           PRE17720
*     IF HAS BEEN RECOGNIZED, REDUCE TO IFTHEN; CALL SEMANTIC ROUTINE       PRE17730
*          5, SCAN AND RETURN TO START                                      PRE17740
*                                                                           PRE17750
IFSTMT          WORKING 'IF UTHEN' RPOS(0) = 'IFTHEN'    :F(PIFERR)         PRE17760
                ROUT5()                                                     PRE17770
                SCAN()                                  :S(START)F(PSCANERR) PRE17780
*                                                                           PRE17790
*     REPEAT HAS BEEN RECOGNIZED, REDUCE TO REPUNTIL; SCAN                  PRE17800
*                                                                           PRE17810
REPEATSTMT      WORKING 'REPEAT UNTIL' RPOS(0) =                            PRE17820
+                    'REPUNTIL'                           :F(PREPERR)        PRE17830
                SCAN()                                   :F(PSCANERR)       PRE17840
*                                                                           PRE17850
*     REPUNTIL HAS BEEN RECOGNIZED, REDUCE TO REPEATSTMT; CALL SEMANTIC     PRE17860
*          ROUTINE 6, SCAN AND RETURN TO START                              PRE17870
*                                                                           PRE17880
REPSTMT         WORKING 'REPUNTIL UEOS' RPOS(0) =                           PRE17890
+                    'REPEATSTMT'                         :F(PREPUERR)       PRE17900
                ROUT6()                                                     PRE17910
                SCAN()                                  :S(START)F(RETURN)  PRE17920
```

```
*                                                                       PRE17930
*    WHILE HAS BEEN RECOGNIZED, REDUCE TO WHILESTMT; CALL SEMANTIC      PRE17940
*        ROUTINE 7, SCAN AND RETURN TO START                            PRE17950
*                                                                       PRE17960
WHILESTMT       WORKING 'WHILE UEOS' RPOS(0) =                          PRE17970
+                       'WHILESTMT'                     :F(PWHERR)      PRE17980
                ROUT7()                                                 PRE17990
                SCAN()                                :S(START)F(RETURN)PRE18000
*                                                                       PRE18010
*    QUITLOOP HAS BEEN RECOGNIZED, REDUCE TO STRUCTURE; CALL SEMANTIC   PRE18020
*        ROUTINE 8 AND BRANCH TO STRUCTURE SECTION                      PRE18030
*                                                                       PRE18040
QUITSTMT        WORKING 'QUITLOOP UEOS' RPOS(0) =                       PRE18050
+                       'STRUC'                         :F(PQUERR)      PRE18060
                ROUT8()                                 :(STRUCTURE)    PRE18070
*                                                                       PRE18080
*    REDUCTION OF CASE-ENDCASE TO STRUCTURE; CALL SEMANTIC              PRE18090
*        ROUTINE 9 AND BRANCH TO STRUCTURE SECTION                      PRE18100
*                                                                       PRE18110
ECASESTMT       WORKING 'CASESTMT STRLIST ENDCASE UEOS'                 PRE18120
+                       RPOS(0) = 'STRUC'               :F(PECASEERR)   PRE18130
                ROUT9()                                 :(STRUCTURE)    PRE18140
*                                                                       PRE18150
*    REDUCTION OF IF-THEN-ENDIF TO STRUCTURE; CALL SEMANTIC ROUTINE     PRE18160
*        10 AND BRANCH TO STRUCTURE SECTION                             PRE18170
*                                                                       PRE18180
EIFSTMT         WORKING 'IFTHCL ENDIF UEOS' RPOS(0) =                   PRE18190
+                       'STRUC'                         :F(NXT.16)      PRE18200
                ROUT10()                                :(STRUCTURE)    PRE18210
*                                                                       PRE18220
*    REDUCTION OF IF-THEN-ELSE-ENDIF TC STRUCTURE; CALL SEMANTIC ROUTINEPRE18230
*        10 AND BRANCH TO STRUCTURE SECION                              PRE18240
*                                                                       PRE18250
NXT.16          WORKING 'IFTHELSE ENDIF UEOS' RPOS(0) =                 PRE18260
+                       'STRUC'                         :F(PIFTHERR)    PRE18270
                ROUT10()                                :(STRUCTURE)    PRE18280
*                                                                       PRE18290
*    REDUCTION OF WHILE-ENDWHILE TO STRUCTURE; CALL SEMANTIC            PRE18300
*        ROUTINE 11 AND BRANCH TO STRUCTURE SECTION                     PRE18310
*                                                                       PRE18320
EWHILESTMT      WORKING 'WHILESTMT STRLIST ENDWHILE UEOS'               PRE18330
+                       RPOS(0) = 'STRUC'               :F(PEWHERR)     PRE18340
                ROUT11()                                :(STRUCTURE)    PRE18350
*                                                                       PRE18360
*    REDUCTION OF REPEAT-ENDREPEAT TO STRUCTURE; CALL SEMANTIC          PRE18370
*        ROUTINE 12 AND BRANCH TO STRUCTURE SECTION                     PRE18380
*                                                                       PRE18390
EREPEATSTMT WORKING 'REPEATSTMT STRLIST ENDREPEAT UEOS' RPOS(0) =       PRE18400
```

```
+                                   'STRUC'                          :F(PEREPERR)
                    ROUT12()                                        :(STRUCTURE)
*
*       REDUCTION OF DO-ENDDO TO STRUCTURE; CALL SEMANTIC ROUTINE
*           13 AND BRANCH TO STRUCTURE SECTION
*           RETURN TO START
*
EDOSTMT             WORKING 'DOSTMT STRLIST ENDDO UEOS'
+                       RPOS(0) = 'STRUC'                           :F(PEDOERR)
                    ROUT13()                                        :(STRUCTURE)
*
*       REDUCTION OF DOGROUP-ENDDOGROUP TO STRUCTURE; CALL SEMANTIC
*           ROUTINE 14 AND BRANCH TO STRUCTURE SECTION
*
EDOGRPSTMT          WORKING 'DOGROUPSTMT STRLIST ENDDOGROUP UEOS'
+                       RPOS(0) = 'STRUC'                           :F(PEDGRPERR)
                    ROUT14()                                        :(STRUCTURE)
*
*       LABEL AND A STRUCTURE REDUCE TO STRUCTURE; RETURN TO STRUCTURE
*           SECTION
*
STRUCTURE           WORKING 'LABEL STRUC' RPOS(0) =
+                       'STRUC'                                     :S(STRUCTURE)
*
*       ELSE AND STRUCTURE REDUCE TO ELSESTRUCTURE; RETURN TO STRUCTRUE
*           SECTION
*
                    WORKING 'ELSE STRUC' RPOS(0) = 'ELSESTRUC'
+                                                                   :S(STRUCTURE)
*
*       LABEL AND ELSE STRUCTURE REDUCE TO ELSESTRUCTURE; RETURN
*           TO STRUCTURE SECTION
*
                    WORKING 'LABEL ELSESTRUC' RPOS(0) =
+                       'ELSESTRUC'                                 :S(STRUCTURE)
*
*       IFTHEN CLAUSE AND ELSESTRUC REDUCE TO IFTHELSE; SCAN AND
*           RETURN TO START
*
                    WORKING 'IFTHCL ELSESTRUC' RPOS(0) =
+                       'IFTHELSE'                                  :F(NXT.17)
                    SCAN()                                      :S(START)F(RETURN)
*
*       IF-THEN AND STRUCTURE REDUCE TO IF THEN CLAUSE; SCAN AND
*           RETURN TO START
*
NXT.17              WORKING 'IFTHEN STRUC' RPOS(0) =
+                       'IFTHCL'                                    :F(NXT.18)
```

```
                SCAN()                                          :S(START)F(RETURN)      PRE18890
*                                                                                       PRE18900
*     STRLIST AND STRUCTURE REDUCE TO STRLIST; CALL SEMANTIC ROUTINE                    PRE18910
*          15, SCAN AND RETURN TO START                                                 PRE18920
*                                                                                       PRE18930
NXT.18          WORKING 'STRLIST STRUC' RPOS(0) =                                       PRE18940
+                    'STRLIST'                                  :F(NXT.19)              PRE18950
                RCUT15()                                                                PRE18960
                SCAN()                                         :(START)                 PRE18970
*                                                                                       PRE18980
*     STRUC REDUCES TO STRLIST; CALL SEMANTIC ROUTINE 15, SCAN                          PRE18990
*          AND RETURN TO START                                                          PRE19000
*                                                                                       PRE19010
NXT.19          WORKING 'STRUC' RPOS(0) = 'STRLIST'            :F(PSTRERR)              PRE19020
                ROUT15()                                                                PRE19030
                SCAN()                                         :(START)                 PRE19040
MACROSCAN  MACCALL POS(0) '<MACRO CALL>:::=' =                 :F(MACCALLERR)           PRE19050
*                                                                                       PRE19060
*********************************************************************************     PRE19061
*                                                                                       PRE19062
*     ROUTINE TO SCAN A MACRO DEFINITION                                                PPE19070
*                                                                                       PRE19080
*     ONE PARAMETER -- MACCALL, THE INPUT CARD (OR FIRST OF TWO)                        PRE19090
*          SPECIFYING A MACRO CALLING FORM                                              PRE19100
*                                                                                       PRE19110
*********************************************************************************     PRE19111
*                                                                                       PRE19112
*     BREAK OUT THE MACRO NAME                                                          PRE19120
*                                                                                       PRE19130
          MACCALL BREAK('(') . NAME '(' =                      :F(MACCALLERR)           PRE19140
*                                                                                       PRE19150
*     BREAK OUT THE PARAMETER LIST, READING IN ANOTHER CARD IF NECESSARY                PRE19160
*                                                                                       PRE19170
          MACCALL BREAK(')') . PLIST ')' RPOS(0)               :S(GETMPARMS)            PRE19180
          MACCALL = MACCALL TRIM(INPUT)                                                 PPE19190
          MACCALL BREAK(')') . PLIST ')' RPOS(0)               :F(MACCALLERR)           PRE19200
*                                                                                       PRE19210
*     BREAK OUT THE INDIVIDUAL PARAMETERS; PLACE THE NUMBER OF                          PRE19220
*          PARAMETERS IN MPARM<0> AND THE PARAMETERS IN ELEMENTS 1-6                    PPE19230
*          OF MPARM                                                                     PPE19240
*                                                                                       PRE19250
GETMPARMS   I = 0                                                                       PPE19260
MACPARMS    I = I + 1                                                                   PPE19270
          PLIST BREAK(',') . MPARM<I> ',' =                    :S(MACPARMS)             PRE19280
          PLIST REM . MPARM<I>                                 :F(MACPERR)              PRE19290
          MPARM<0> = I                                                                  PPE19300
          MACDEF = TRIM(INPUT)                                                          PPE19310
          MACDEF '<MACRO DEFINITION>:::=' RPOS(0)              :F(MACDEFERR)            PRE19320
```

```
*                                                                    PRE19330
*    READ THE MACRO DEFINITION AND PLACE IT IN MACREPSTR             PRE19340
*                                                                    PRE19350
          MACREPSTR =                                               PRE19360
MACREP    MACDEF = INPUT                                            PRE19370
          MACDEF POS(0) 'END MACRO DEFINITION' SPAN(' ') RPOS(0)    PRE19380
+                                                  :S(ENDMACDEF)    PRE19390
          MACREPSTR = MACREPSTR MACDEF             :(MACREP)        PRE19400
*                                                                    PRE19410
*    ADD THE MACRONAME TO THE MACRCNAME PATTERN                      PRE19420
ENDMACDEF MACNAMEPAT = MACNAMEPAT | NAME                            PRE19430
*                                                                    PRE19440
*    START THE PARAMETER PATTERN WITH A SERIES OF 81 BLANKS          PRE19450
          PARM = DUPL(' ',81)                                       PRE19460
*                                                                    PRE19470
*    ADD THE PARAMETERS TO THE PARAMETER PATTERN                     PRE19480
PARMPAT   PARM = PARM | MPARM<I>                                    PRE19490
          I = GT(I,1) I - 1                        :S(PARMPAT)      PRE19500
*                                                                    PRE19510
*    BREAK THE MACRO DEFINITION INTO COMPONENTS AND PLACE THEM IN    PRE19520
*        ELEMENTS 1-20 OF MPAT                                       PRE19530
*                                                                    PRE19540
          I = -1                                                    PRE19550
MACDEFLOOP I = I + 2                                                PRE19560
          MACREPSTR ARB . COPY PARM . PARAM =      :F(MACDEFEND)    PRE19570
          MACREPSTR POS(0) SPAN(' ') . BLKS =                       PRE19580
          MPAT<I> = COPY                                            PRE19590
          MPAT<I + 1> = PARAM BLKS                                  PRE19600
          BLKS =                                                    PRE19610
          GT(SIZE(MACREPSTR),0)       :S(MACDEFLOOP)F(MACDEFLEN)    PRE19620
MACDEFEND MPAT<I> = MACREPSTR                                       PRE19630
*                                                                    PRE19640
*    PLACE THE NUMBER OF COMPONENTS IN MPAT<0>                       PRE19650
          MPAT<0> = I                              :(RETMACDEF)     PRE19660
MACDEFLEN MPAT<0> = I + 1                                           PRE19670
*                                                                    PRE19680
*    STORE THE MPAT ARRAY IN ELEMENT 'MACRONAME' OF TABLE MACROTAB   PRE19690
*                                                                    PRE19700
*    STORE THE MPARM ARRAY IN ELEMENT 'MACRONAME P' OF TABLE MACROTAB PRE19710
*        AND RETURN TO CALLING ROUTINE                               PRE19720
*                                                                    PRE19730
RETMACDEF MACROTAB<NAME> = COPY(MPAT)                               PRE19740
          MACROTAB<NAME 'P'> = COPY(MPARM)                          PRE19750
          FINOUT = NAME ' MACRO DEFINITION PROCESSED.'              PRE19760
          FINOUT = '    '                                           PRE19770
          FINOUT = '    '                          :(RETURN)        PRE19780
SCANDEFS  &ANCHOR = 1                                               PRE19790
*                                                                    PRE19800
```

```
*****************************************************************PRE19801
*                                                                PRE19802
*     ROUTINE TO SCAN DEFINITIONS AND CALL APPROPRIATE           PRE19810
*         INDIVIDUAL SCANNING ROUTINE                            PRE19820
*                                                                PRE19830
*     ONE PARAMETER -- CARD;  CARD IS THE FIRST CARD OF THE      PRE19840
*         FIRST DEFINITION                                       PRE19850
*                                                                PRE19860
*****************************************************************PRE19861
*                                                                PRE19862
          I = 1                                                 PRE19870
          DEFNAME = 'NULL STATEMENT' | 'GO TO STATEMENT' |      PRE19880
+                   'IF THEN GO TO STATEMENT' | 'ASSIGN MINUS STATEMENT'  PRE19890
+                   | 'ASSIGN PLUS STATEMENT' | 'DECLARATION STATEMENT'  PRE19900
+                   | 'LITERAL' | 'COMMENT' | 'UNIQUE LABEL' |  PRE19910
+                   'UNIQUE IDENTIFIER' | 'GREATER THAN' | 'NOT' |  PRE19920
+                   'LESS THAN OR EQUAL' | 'GREATER THAN OR EQUAL' |  PRE19930
+                   'NOT EQUAL' | 'CONTINUATION' | 'END OF STATEMENT' |  PRE19940
+                   'IGNORE COLUMNS' | 'LANGLABEL'               PRE19950
*                                                                PRE19960
*     DETERMINE DEFINITION NAME                                  PRE19970
*                                                                PRE19980
NEXTDEF   CARD '<' DEFNAME . SELECT '>:=' =          :F(ILLEGALDEF)  PRE19990
          SELECT ARB . STMTTYPE ' STATEMENT' RPOS(0) :F(NOTASTMT)  PRE20000
*                                                                PRE20010
*     BRANCH IF A STATEMENT DEFINITION                           PRE20020
*                                                                PRE20030
          STMTTYPE 'END OF' RPOS(0)                :F(NOTEOS)    PRE20040
*                                                                PRE20050
*     IF END OF STATEMENT DEFINITION, TRIM TRAILING BLANKS       PRE20060
*         AND BRANCH                                             PRE20070
*                                                                PRE20080
          CARD = TRIM(CARD)                        :(EOSSCAN)    PRE20090
*                                                                PRE20100
*     STATEMENT PROTOTYPES REQUIRE TWO INPUT CARDS, SO READ      PRE20110
*         ANOTHER CARD AND TRIM THE TRAILING BLANKS FROM THE     PRE20120
*         ENTIRE DEFINITION                                      PRE20130
*                                                                PRE20140
NOTEOS    CARD = TRIM(CARD INPUT)                                PRE20150
*                                                                PRE20160
*     BRANCH TO THE APPROPRIATE LABEL                            PRE20170
*                                                                PRE20180
          STMTTYPE 'NULL' RPOS(0)                  :S(NULLSCAN)  PRE20190
          STMTTYPE 'GO TO' RPOS(0)                 :S(GOTOSCAN)  PRE20200
          STMTTYPE 'IF THEN GO TO' RPOS(0)         :S(IFSCAN)    PRE20210
          STMTTYPE 'ASSIGN MINUS' RPOS(0)          :S(MINUSSCAN) PRE20220
          STMTTYPE 'ASSIGN PLUS' RPOS(0)           :S(PLUSSCAN)  PRE20230
*                                                                PRE20240
```

```
*     ERROR IF NOT A VALID STATEMENT TYPE IN THE DEFINITION NAME          PRE20250
*                                                                         PRE20260
          STMTTYPE 'DECLARATION' RPOS(0)    :S(DCLSCAN)F(ILLEGALDEF)      PRE20270
*                                                                         PRE20280
*     IF LANGUAGE LABEL DEFINITION, BRANCH                                PRE20290
*                                                                         PRE20300
NOTASTMT  SELECT 'LANGLABEL' RPOS(0)                  :S(LANGLABELSCAN)   PRE20310
*                                                                         PRE20320
*     ALL OTHER DEFINITIONS REQUIRE ONLY ONE CARD, SO TRIM THE            PRE20330
*        TRAILING BLANKS FROM THIS CARD                                   PRE20340
*                                                                         PRE20350
          CARD = TRIM(CARD)                                              PRE20360
*                                                                         PRE20370
*     BRANCH APPROPRIATELY BASED ON DEFINITION NAME                       PRE20380
*                                                                         PRE20390
          SELECT BREAK(' ')                        :F($(SELECT 'SCAN'))   PRE20400
          SELECT 'UNIQUE ' REM . UNITYPE           :S($(UNITYPE 'SCAN'))  PRE20410
          SELECT 'IGNORE COLUMNS' RPOS(0)             :S(IGNSCAN)         PRE20420
          SELECT 'NOT EQUAL' RPOS(0)                  :S(NEQUSCAN)        PRE20430
          SELECT 'GREATER THAN' RPOS(0)               :S(GTHANSCAN)       PRE20440
          SELECT 'GREATER THAN OR EQUAL' RPOS(0)      :S(GEQUSCAN)        PRE20450
          SELECT 'LESS THAN OR EQUAL' RPOS(0)                             PRE20460
+                                    :S(LEQUSCAN)F(ILLEGALDEF)            PRE20470
*                                                                         PRE20480
*     ERROR MESSAGE FOR INVALID DEFINITION NAME OR                        PRE20490
*        EXTRANEOUS CARD                                                  PRE20500
*                                                                         PRE20510
ILLEGALDEF OUTPUT = 'CARD WITH INVALID DEFINITION NAME OR CARD NOT '      PRE20520
+                   'CONTAINING A DEFINITION ENCOUNTERED -> '             PRE20530
+                   CARD                                                  PRE20540
*                                                                         PRE20550
*     RETURN IF THIS SHOULD HAVE BEEN THE LAST DEFINITION                 PRE20560
*                                                                         PRE20570
          EQ(I,19)                                    :F(OUTSKIPMSG)      PRE20580
          &ANCHOR = 0                                 :(RETURN)           PRE20590
OUTSKIPMSG OUTPUT = 'SKIPPING TO NEXT DEFINITION'                         PRE20600
*                                                                         PRE20610
*     INCREMENT DEFINITION COUNTER AND LOCATE NEXT CARD                   PRE20620
*        CONTAINING A DEFINITION NAME IN ANGLE BRACKETS                   PRE20630
*                                                                         PRE20640
          I = I + 1                                                       PRE20650
SKIPCARD  CARD = INPUT                                                    PRE20660
          CARD '<'                          :S(NEXTDEF)F(SKIPCARD)        PRE20670
*                                                                         PRE20680
*     DETERMINE IF THE ENTIRE OPENING CHARACTER SET IS CONTAINED ON       PRE20690
*        THE FIRST CARD;  IF SO, BRANCH TO LASTSET;  IF NOT, READ         PRE20700
*        IN ANOTHER CARD AND TRIM THE TRAILING BLANKS FROM THE ENTIRE     PRE20710
*        DEFINITION                                                       PRE20720
```

```
*                                                                       PRE20730
LANGLABELSCAN CARD BREAK(' ') ' ' BREAK(' ') ' ' REM . REST             PRE20740
            REST SPAN(' ') =                                            PRE20750
            REST "'" BREAK("'") "'" =                  :S(LASTSET)      PRE20760
NEEDONE     CARD = TRIM(CARD INPUT)                                     PRE20770
*                                                                       PRE20780
*     CALL THE LANGUAGE LABEL DEFINITION SCANNER                        PRE20790
*                                                                       PRE20800
CALLSCAN    SCANLANLABDEF(CARD)                        :(NEXTIN)        PRE20810
*                                                                       PRE20820
*     DETERMINE IF THE ENTIRE REMAINING CHARACTER SET IS CONTAINED      PRE20830
*          ON THE FIRST CARD;  IF NOT, BRANCH TO READ IN ANOTHER CARD   PRE20840
*          AND TRIM THE TRAILING BLANKS;  IF SO, TRIM THE TRAILING      PRE20850
*          BLANKS AND BRANCH TO CALL THE SCANNING ROUTINE               PRE20860
*                                                                       PRE20870
LASTSET     REST SPAN(' ') =                                            PRE20880
            REST "'" BREAK("'") "'"                    :F(NEEDONE)      PRE20890
            CARD = TRIM(CARD)                          :(CALLSCAN)      PRE20900
NOTSCAN     SCANOPERDEF('NOT',CARD)                    :(NEXTIN)        PRE20910
CONTINUATIONSCAN SCANCONTDEF(CARD)                     :(NEXTIN)        PRE20920
COMMENTSCAN SCANCOMDEF(CARD)                           :(NEXTIN)        PRE20930
LITERALSCAN SCANLITDEF(CARD)                           :(NEXTIN)        PRE20940
NULLSCAN    SCANNULLDEF(CARD)                          :(NEXTIN)        PRE20950
GOTOSCAN    SCANGOTODEF(CARD)                          :(NEXTIN)        PRE20960
IFSCAN      SCANIFDEF(CARD)                            :(NEXTIN)        PRE20970
MINUSSCAN   SCANMINUSDEF(CARD)                         :(NEXTIN)        PRE20980
PLUSSCAN    SCANPLUSDEF(CARD)                          :(NEXTIN)        PRE20990
DCLSCAN     SCANDCLDEF(CARD)                           :(NEXTIN)        PRE21000
LABELSCAN   SCANLABELDEF(CARD)                         :(NEXTIN)        PRE21010
IDENTIFIERSCAN SCANVARDEF(CARD)                        :(NEXTIN)        PRE21020
EOSSCAN     SCANEOSDEF(CARD)                           :(NEXTIN)        PRE21030
IGNSCAN     SCANIGNCOLDEF(CARD)                        :(NEXTIN)        PRE21040
NEQUSCAN    SCANOPERDEF('NEQU',CARD)                   :(NEXTIN)        PRE21050
GTHANSCAN   SCANOPERDEF('GTHAN',CARD)                  :(NEXTIN)        PRE21060
GEQUSCAN    SCANOPERDEF('GEQU',CARD)                   :(NEXTIN)        PRE21070
LEQUSCAN    SCANOPERDEF('LEQU',CARD)                   :(NEXTIN)        PRE21080
*                                                                       PRE21090
*     RETURN IF THIS WAS THE LAST DEFINITION                            PRE21100
*                                                                       PRE21110
NEXTIN      EQ(I,19)                                   :F(INCRDEFNCTR)  PRE21120
            &ANCHOR = 0                                :(RETURN)        PRE21130
*                                                                       PRE21140
*     INCREMENT DEFINITION COUNTER AND READ ANOTHER CARD                PRE21150
*                                                                       PRE21160
INCRDEFNCTR I = I + 1                                                   PRE21170
            CARD = INPUT                               :(NEXTDEF)       PRE21180
SCANOPERDEF OPERDEF "'" ARB . $TYPEDEF "'" RPOS(0) =                    PRE21190
+                                          :F($(TYPEDEF 'DEFERR'))      PRE21200
```

```
            SCANOPERDEF =                                   :(RETURN)          PRE21210
SCANVARDEF                                                                     PRE21220
*                                                                              PRE21230
*****************************************************************************PRE21231
*                                                                              PRE21232
*     ROUTINE TO SCAN DEFINITION OF UNIQUE VARIABLE                            PRE21240
*                                                                              PRE21250
*     ONE PARAMETER -- VARDEF                                                  PRE21260
*                                                                              PRE21270
*****************************************************************************PRE21271
*                                                                              PRE21272
            VARDEF BREAK(' ') . VARPAT<1> SPAN(' ') =   :F(VARERROR)           PRE21280
            VARDEF BREAK(' ') . VARPAT<2> =             :F(NOVAREND)           PRE21290
            VARDEF SPAN(' ') =                                                 PRE21300
            VARPAT<3> = VARDEF                                                 PRE21310
            VARDEF =                                    :(TESTVARCOL)          PRE21320
NOVAREND    VARPAT<3> =                                                        PRE21330
            VARPAT<2> = VARDEF                                                 PRE21340
            VARDEF =                                                           PRE21350
*                                                                              PRE21360
*     TEST FOR COLUMN SPECIFICATION CONTAINING ONLY DIGITS                     PRE21370
*                                                                              PRE21380
TESTVARCOL  VARPAT<2> SPAN('0123456789') RPOS(0)    :F(VARERROR)           PRE21390
            SCANVARDEF =                            :(RETURN)              PRE21400
SCANNULLDEF                                                                    PRE21410
*                                                                              PRE21420
*****************************************************************************PRE21421
*                                                                              PRE21422
*     ROUTINE TO SCAN DEFINITION OF NULL STATEMENT                             PRE21430
*                                                                              PRE21440
*     ONE PARAMETER -- NULLDEF                                                 PRE21450
*                                                                              PRE21460
*****************************************************************************PRE21461
*                                                                              PRE21462
            I = -1                                                             PRE21470
            J = 0                                                              PRE21480
NULLDEFLOOP I = I + 2                                                          PRE21490
            NULLDEF ARB . COPY 'LAB' =                  :F(NULLDEFEND)         PRE21500
            NULLDEF POS(0) SPAN(' ') . BLKS =                                  PRE21510
            J = J + 1                                                          PRE21520
            NULLPAT<I> = COPY                                                  PRE21530
*                                                                              PRE21540
*     SAVE PARAMETER NAME AND BLANKS FOLLOWING IT                              PRE21550
*                                                                              PRE21560
*                                                                              PRE21570
            NULLPAT<I + 1> = 'LAB' BLKS                                        PRE21580
            BLKS =                                                             PRE21590
            GT(SIZE(NULLDEF),0)                                                PRE21600
```

```
                                                :S(NULLDEFLOOP)F(NULLDEFLEN)      PRE21610
NULLDEFEND NULLPAT<I> = NULLDEF                                                   PRE21620
           NULLPAT<0> = I                                                         PRE21630
           NULLDEF =                                    :(ENDNULLDEF)         PRE21640
NULLDEFLEN NULLPAT<0> = I + 1                                                     PRE21650
ENDNULLDEF SCANNULLDEF =                                                          PRE21660
*                                                                                 PRE21670
*     TEST FOR ALL PARAMETERS SPECIFIED                                           PRE21680
*                                                                                 PRE21690
           EQ(J,1)                                                                PRE21700
+                                               :S(RETURN)F(NULLPARMERR)          PRE21710
SCANGOTODEF                                                                       PRE21720
*                                                                                 PRE21730
*********************************************************************************PRE21731
*                                                                                 PRE21732
*     ROUTINE TO SCAN DEFINITION OF GO TO STATEMENT                               PRE21740
*                                                                                 PRE21750
*     ONE PARAMETER -- GOTODEF                                                    PRE21760
*                                                                                 PRE21770
*********************************************************************************PRE21771
*                                                                                 PRE21772
           PARM = 'LAB' | 'DES'                                                   PRE21780
           I = -1                                                                 PRE21790
           J = 0                                                                  PRE21800
GOTODEFLOOP I = I + 2                                                             PRE21810
           GOTODEF ARB . COPY PARM . PARAM =            :F(GOTODEFEND)            PRE21820
           GOTODEF POS(0) SPAN(' ') . BLKS =                                      PRE21830
           J = J + 1                                                              PRE21840
           GOTOPAT<I> = COPY                                                      PRE21850
*                                                                                 PRE21860
*     SAVE PARAMETER NAME AND BLANKS FOLLOWING IT                                 PRE21870
*                                                                                 PRE21880
           GOTOPAT<I + 1> = PARAM BLKS                                            PRE21890
           BLKS =                                                                 PRE21900
           GT(SIZE(GOTODEF),0)                                                    PRE21910
+                                               :S(GOTODEFLOOP)F(GOTODEFLEN)      PRE21920
GOTODEFEND GOTOPAT<I> = GOTODEF                                                   PRE21930
           GOTOPAT<0> = I                                                         PRE21940
           GOTODEF =                                    :(ENDGOTODEF)         PRE21950
GOTODEFLEN GOTOPAT<0> = I + 1                                                     PRE21960
ENDGOTODEF SCANGOTODEF =                                                          PRE21970
*                                                                                 PRE21980
*     TEST FOR ALL PARAMETERS SPECIFIED                                           PRE21990
*                                                                                 PRE22000
           EQ(J,2)                                                                PRE22010
+                                               :S(RETURN)F(GOPARMERR)            PRE22020
SCANIFDEF                                                                         PRE22030
*                                                                                 PRE22040
```

```
*****************************************************************PRE22041
*                                                                PRE22042
*     ROUTINE TO SCAN DEFINITION OF IF THEN GO TO STATEMENT      PRE22050
*                                                                PRE22060
*     ONE PARAMETER -- IFDEF                                     PRE22070
*                                                                PRE22080
*****************************************************************PRE22081
*                                                                PRE22082
            PARM = 'LAB' | 'VAL1' | 'RL' | 'VAL2' | 'DES'       PRE22090
            I = -1                                              PRE22100
            J = 0                                               PRE22110
IFDEFLOOP   I = I + 2                                           PRE22120
            IFDEF ARB . COPY PARM . PARAM =          :F(IFDEFEND)PRE22130
            IFDEF POS(0) SPAN(' ') . BLKS =                     PRE22140
            J = J + 1                                           PRE22150
            IFPAT<I> = COPY                                     PRE22160
*                                                               PRE22170
*     SAVE PARAMETER NAME AND BLANKS FOLLOWING IT               PRE22180
*                                                               PRE22190
            IFPAT<I + 1> = PARAM BLKS                           PRE22200
            BLKS =                                              PRE22210
            GT(SIZE(IFDEF),0)                                   PRE22220
+                                    :S(IFDEFLOOP)F(IFDEFLEN)   PRE22230
IFDEFEND    IFPAT<I> = IFDEF                                    PRE22240
            IFPAT<0> = I                                        PRE22250
            IFDEF =                              :(ENDIFDEF)    PRE22260
IFDEFLEN    IFPAT<0> = I + 1                                    PRE22270
ENDIFDEF    SCANIFDEF =                                         PRE22280
*                                                               PRE22290
*     TEST FOR ALL PARAMETERS SPECIFIED                         PRE22300
*                                                               PRE22310
            EQ(J,5)                                             PRE22320
+                                    :S(RETURN)F(IFPARMERR)     PRE22330
SCANDCLDEF                                                      PRE22340
*                                                               PRE22350
*****************************************************************PRE22351
*                                                                PRE22352
*     ROUTINE TO SCAN DEFINITION OF DECLARATION STATEMENT        PRE22360
*                                                                PRE22370
*     ONE PARAMETER -- DCLDEF                                    PRE22380
*                                                                PRE22390
*****************************************************************PRE22391
*                                                                PRE22392
            I = -1                                              PRE22400
            J = 0                                               PRE22410
DCLDEFLOOP  I = I + 2                                           PRE22420
            DCLDEF ARB . COPY 'VARNAME' =            :F(DCLDEFEND)PRE22430
            DCLDEF POS(0) SPAN(' ') . BLKS =                    PRE22440
```

```
            J = J + 1                                                          PRE22450
            DCLPAT<I> = COPY                                                   PRE22460
*                                                                              PRE22470
*     SAVE PARAMETER NAME AND BLANKS FOLLOWING IT                              PRE22480
*                                                                              PRE22490
            DCLPAT<I + 1> = 'VARNAME' BLKS                                     PRE22500
            BLKS =                                                             PRE22510
            GT(SIZE(DCLDEF),0)                                                 PRE22520
+                                        :S(DCLDEFLOOP)F(DCLDEFLEN)            PRE22530
DCLDEFEND   DCLPAT<I> = DCLDEF                                                 PRE22540
            DCLPAT<0> = I                                                      PRE22550
            DCLDEF =                                 :(ENDDCLDEF)              PRE22560
DCLDEFLEN   DCLPAT<0> = I + 1                                                  PRE22570
ENDDCLDEF   SCANDCLDEF =                             :(RETURN)                 PRE22580
SCANMINUSDEF                                                                   PRE22590
*                                                                              PRE22600
*******************************************************************************PRE22601
*                                                                              PRE22602
*     ROUTINE TO SCAN DEFINITION OF ASSIGN MINUS STATEMENT                     PRE22610
*                                                                              PRE22620
*     ONE PARAMETER -- MINUSDEF                                                PRE22630
*                                                                              PRE22640
*******************************************************************************PRE22641
*                                                                              PRE22642
            PARM = 'LAB' | 'VAR' | 'VAL1' | 'VAL2'                             PRE22650
            I = -1                                                             PRE22660
            J = 0                                                              PRE22670
MINUSDEFLOOP I = I + 2                                                         PRE22680
            MINUSDEF ARB . COPY PARM . PARAM =           :F(MINUSDEFEND)       PRE22690
            MINUSDEF POS(0) SPAN(' ') . BLKS =                                 PRE22700
            J = J + 1                                                          PRE22710
            MINUSPAT<I> = COPY                                                 PRE22720
*                                                                              PRE22730
*     SAVE PARAMETER NAME AND BLANKS FOLLOWING IT                              PRE22740
*                                                                              PRE22750
            MINUSPAT<I + 1> = PARAM BLKS                                       PRE22760
            BLKS =                                                             PRE22770
            GT(SIZE(MINUSDEF),0)                                               PRE22780
+                                        :S(MINUSDEFLOOP)F(MINUSDEFLEN)        PRE22790
MINUSDEFEND MINUSPAT<I> = MINUSDEF                                             PRE22800
            MINUSPAT<0> = I                                                    PRE22810
            MINUSDEF =                               :(ENDMINUSDEF)            PRE22820
MINUSDEFLEN MINUSPAT<0> = I + 1                                                PRE22830
ENDMINUSDEF SCANMINUSDEF =                                                     PRE22840
*                                                                              PRE22850
*     TEST FOR ALL PARAMETERS SPECIFIED                                        PRE22860
*                                                                              PRE22870
            EQ(J,4)                                                            PRE22880
```

```
+                                                  :S(RETURN)F(MINPARMERR)          PRE22890
SCANPLUSDEF                                                                         PRE22900
*                                                                                   PRE22910
*******************************************************************************PRE22911
*                                                                                   PRE22912
*     ROUTINE TO SCAN DEFINITION OF ASSIGN PLUS STATEMENT                           PRE22920
*                                                                                   PRE22930
*     ONE PARAMETER -- PLUSDEF                                                      PRE22940
*                                                                                   PRE22950
*******************************************************************************PRE22951
*                                                                                   PRE22952
          PARM = 'LAB' | 'VAR' | 'VAL1' | 'VAL2'                                    PRE22960
          I = -1                                                                    PRE22970
          J = 0                                                                     PRE22980
PLUSDEFLOOP I = I + 2                                                               PRE22990
          PLUSDEF ARB . COPY PARM . PARAM =              :F(PLUSDEFEND)             PRE23000
          PLUSDEF POS(0) SPAN(' ') . BLKS =                                         PRE23010
          J = J + 1                                                                 PRE23020
          PLUSPAT<I> = COPY                                                         PRE23030
*                                                                                   PRE23040
*     SAVE PARAMETER NAME AND BLANKS FOLLOWING IT                                   PRE23050
*                                                                                   PRE23060
          PLUSPAT<I + 1> = PARAM BLKS                                               PRE23070
          BLKS =                                                                    PRE23080
          GT(SIZE(PLUSDEF),0)                                                       PRE23090
+                                            :S(PLUSDEFLOOP)F(PLUSDEFLEN)           PRE23100
PLUSDEFEND PLUSPAT<I> = PLUSDEF                                                     PRE23110
          PLUSPAT<0> = I                                                            PRE23120
          PLUSDEF =                                        :(ENDPLUSDEF)            PRE23130
PLUSDEFLEN PLUSPAT<0> = I + 1                                                       PRE23140
ENDPLUSDEF SCANPLUSDEF =                                                            PRE23150
*                                                                                   PRE23160
*     TEST FOR ALL PARAMETERS SPECIFIED                                             PRE23170
*                                                                                   PRE23180
          EQ(J,4)                                                                   PRE23190
+                                            :S(RETURN)F(PLUSPARMERR)               PRE23200
SCANLABELDEF                                                                        PRE23210
*                                                                                   PRE23220
*******************************************************************************PRE23221
*                                                                                   PRE23222
*     ROUTINE TO SCAN DEFINITION OF UNIQUE LABEL                                    PRE23230
*                                                                                   PRE23240
*     ONE PARAMETER -- LABELDEF                                                     PRE23250
*                                                                                   PRE23260
*******************************************************************************PRE23261
*                                                                                   PRE23262
          LABELDEF BREAK(' ') . LABELPAT<1> SPAN(' ') =                             PRE23270
+                                                        :F(LABERROR)               PRE23280
```

```
            LABELDEF BREAK(' ') . LABELPAT<2> =          :F(NOLABEND)        PRE23290
            LABELDEF SPAN(' ') =                                             PRE23300
            LABELPAT<3> = LABELDEF                                           PRE23310
            LABELDEF =                                   :(TESTLABCOL)       PRE23320
NOLABEND    LABELPAT<3> =                                                    PRE23330
            LABELPAT<2> = LABELDEF                                           PRE23340
            LABELDEF =                                                       PRE23350
*                                                                            PRE23360
*     TEST FOR COLUMN SPECIFICATION CONTAINING ONLY DIGITS                   PRE23370
*                                                                            PRE23380
TESTLABCOL  LABELPAT<2> SPAN('0123456789') RPOS(0)   :F(LABERROR)            PRE23390
            SCANLABELDEF =                           :(RETURN)               PRE23400
ROUTO          LANLABPAT<2> 'COL'                    :S(OUTPUTLABCOM)        PRE23410
               LABELNAME = SUBSTR(LABELNAME,1,SIZE(LABELNAME) - 1)           PRE23420
OUTPUTLABCOM   STRING = CRNULL(LABELNAME)                                    PRE23430
*                                                                            PRE23440
*************************************************************************PRE23441
*                                                                            PRE23442
*     ROUTINE FOR LABEL PROCESSING -- NO PARAMETERS                          PRE23450
*                                                                            PRE23460
*************************************************************************PRE23461
*                                                                            PRE23462
*     INSERT A NULL STATEMENT USING THE LABELNAME AND RETURN                 PRE23470
*                                                                            PRE23480
               INSERT(STRING)                        :(RETURN)               PRE23490
ROUTO1      &ANCHOR = 1                                                      PRE23500
*                                                                            PRE23510
*************************************************************************PRE23511
*                                                                            PRE23512
*     ROUTINE FOR UNIDENTIFIED-TO-END-OF-STATEMENT AND MACRO CALL            PRE23520
*         PROCESSING -- NO PARAMETERS                                        PRE23530
*                                                                            PRE23540
*************************************************************************PRE23541
*                                                                            PRE23542
*     DETERMINE IF THIS UEOS WAS A MACRO CALL                                PRE23550
*                                                                            PRE23560
            CURSTMT SPAN(' ') MACNAMEPAT . SELMAC '('                        PRE23570
+                             BREAK(')') . PARMLIST ')'                      PRE23580
+                                                    :S(MACROCALL)            PRE23590
            CURSTMT MACNAMEPAT . SELMAC '(' BREAK(')') . PARMLIST            PRE23600
+                           ')'                      :F(COPYUEOS)            PRE23610
*                                                                            PRE23620
MACROCALL   I = 0                                                            PRE23630
*                                                                            PRE23640
*************************************************************************PRE23641
*                                                                            PRE23642
*     PROCESS A MACRO CALL                                                   PRE23643
*                                                                            PRE23644
```

```
*************************************************************************PRE23645
*                                                                       PRE23646
*     BREAK OUT THE ACTUAL ARGUMENTS                                    PRE23650
*                                                                       PRE23660
BREAKPARMS I = I + 1                                                    PRE23670
           PARMLIST BREAK(',') . CPARM<I> ',' =          :S(BREAKPARMS)   PRE23680
           PARMLIST REM . CPARM<I>                       :F(MACCALLPERR)  PRE23690
           ACTARG = I                                                   PRE23700
*                                                                       PRE23710
*     RETRIEVE THE MACRO PATTERN AND NUMBER OF COMPONENTS INTO MPAT     PRE23720
*                                                                       PRE23730
*     RETRIEVE THE MACRO PARAMETERS AND NUMBER OF PARAMETERS INTO MPARM PRE23740
*                                                                       PRE23750
           MPAT = COPY(MACROTAB<SELMAC>)                                PRE23760
           MPARM = COPY(MACROTAB<SELMAC 'P'>)                           PRE23770
           I = MPARM<0>                                                 PRE23780
*                                                                       PRE23790
*     START THE PARAMETER PATTERN WITH A SERIES OF 81 BLANKS            PRE23800
*                                                                       PRE23810
           PARM = DUPL(' ',81)                                          PRE23820
*                                                                       PRE23830
*     ADD THE PARAMETERS TO THE PARAMETER PATTERN                       PRE23840
*                                                                       PRE23850
MACPARMPAT PARM = PARM | MPARM<I>                                       PRE23860
           I = GT(I,1) I - 1                             :S(MACPARMPAT)   PRE23870
           MACSTR =                                                     PRE23880
*                                                                       PRE23890
*     COPY THE STATEMENT COMPONENTS AND REPLACE THE PARAMETERS          PRE23900
*         WITH THE ARGUMENTS                                            PRE23910
*                                                                       PRE23920
           I = 0                                                        PPE23930
           STMTLEN = MPAT<0>                                            PRE23940
           LE(STMTLEN,0)                                 :S(ROUTOLERR)  PRE23950
MACLOOP    LT(I,STMTLEN)                                 :F(MACDONE)    PRE23960
           I = I + 1                                                    PRE23970
           MPAT<I> PARM . SELECTPARM SPAN(' ') RPOS(0)                  PRE23980
+                                                        :S(COPYMACPARM)  PRE23990
           MPAT<I> PARM . SELECTPARM RPOS(0)             :F(COPYMAC)    PRE24000
           J = 0                                                        PRE24010
INCR.J     J = LT(J,ACTARG) J + 1                        :F(MACARGERR)  PRE24020
           SELECTPARM MPARM<J>                           :F(INCR.J)     PRE24030
           MACSTR = MACSTR CPARM<J>                      :(MACLOOP)     PRE24040
COPYMACPARM J = 0                                                       PRE24050
INCR2.J    J = LT(J,ACTARG) J + 1                        :F(MACARGERR)  PRE24060
           SELECTPARM MPARM<J>                           :F(INCR2.J)    PRE24070
           LT(SIZE(MPAT<I>) - SIZE(CPARM<J>),0)          :S(MACBLKERR)  PRE24080
           MACSTR = MACSTR CPARM<J> DUPL(' ',SIZE(MPAT<I>) -            PRE24090
+                          SIZE(CPARM<J>))                :(MACLOOP)     PRE24100
```

```
COPYMAC     MACSTR = MACSTR MPAT<I>                                  :(MACLOOP)          PRE24110
*                                                                                        PRE24120
*************************************************************************************PRE24121
*                                                                                        PRE24122
*     THIS IS AN UNIDENTIFIED-TO-END-OF-STATEMENT                                      PRE24123
*                                                                                        PRE24124
*************************************************************************************PRE24125
*                                                                                        PRE24126
*     NOT A MACRO CALL; INSERT THE CURSTMT; NULLIFY CURSTMT AND RETURN                 PRE24130
*                                                                                        PRE24140
COPYUEOS    INSERTSTMT(CURSTMT)                                                        PRE24150
            &ANCHOR = 0                                                                PRE24160
            CURSTMT =                                            :(RETURN)             PRE24170
MACDONE     INSERTCOM(CURSTMT)                                                         PRE24180
                                                                                       PRE24190
*     IS A MACRO CALL; INSERT THE CURSTMT AS A COMMENT; NULLIFY                        PRE24200
*          THE CURSTMT; INSERT THE MACRO REPLACEMENT TEXT AS ONE OR                    PRE24210
*          MORE STATEMENTS AND RETURN                                                  PRE24220
*                                                                                      PRE24230
              CURSTMT =                                                                PRE24240
              INSERT(MACSTR)                                                           PRE24250
              &ANCHOR = 0                                                              PRE24260
              MACSTR =                                           :(RETURN)             PRE24270
ROUT1             &ANCHOR = 1                                                          PRE24280
*                                                                                      PRE24290
*************************************************************************************PRE24291
*                                                                                      PRE24292
*     ROUTINE FOR ELSE PROCESSING                                                      PRE24300
*                                                                                      PRE24310
*************************************************************************************PRE24311
*                                                                                      PRE24312
*     ERROR IF NOT IN CORRECT FORM FOR ELSE                                            PRE24320
*                                                                                      PRE24330
                  INSTR = CURSTMT                                                      PRE24340
                  INSTR SPAN(' ') =                                                    PRE24350
                  INSTR 'ELSE' =                                   :F(ELSEERROR)       PRE24360
                  &ANCHOR = 0                                                          PRE24370
                  CURSTMT 'ELSE' = '     '                                             PRE24380
                  &ANCHOR = 1                                                          PRE24390
*                                                                                      PRE24400
*     POP THE LABEL TO BE BRANCHED TO ON FALSE FROM THE SEMANTIC STACK;                PRE24410
*          CREATE THE LABEL TO BE PLACED AT THE END OF THE IF STRUCTURE;               PRE24420
*          CREATE A STRING OF A GOTO STATEMENT AND A NULL STATEMENT                    PRE24430
*                                                                                      PRE24440
GEN1              POP.SEM(1,'VAR1')                                                    PRE24450
                  VAR2 = UNILABEL()                                                    PRE24460
                  STRING = CRGOTO(UNILABEL(),VAR2)                                     PRE24470
+                       CRNULL(VAR1)                                                   PRE24480
```

```
                INSERTCOM('ELSE KEYWORD PROCESSED.')                                PRE24490
*                                                                                   PRE24500
*       INSERT THE STRING                                                           PRE24510
*                                                                                   PRE24520
                INSERT(STRING)                                                      PRE24530
*                                                                                   PRE24540
*       PUSH THE LABEL CREATED IN THIS ROUTINE ONTO THE SEMANTIC STACK              PRE24550
*                                                                                   PRE24560
                PUSH.SEM(1,VAR2)                                                    PRE24570
                &ANCHOR = 0                                 :(RETURN)               PRE24580
ROUT2           &ANCHOR = 1                                                         PRE24590
*                                                                                   PRE24600
*********************************************************************************PRE24601
*                                                                                   PRE24602
*       ROUTINE FOR CASE STATEMENT PROCESSING                                       PRE24610
*                                                                                   PRE24620
*********************************************************************************PRE24621
*                                                                                   PRE24622
*       ERROR IF NOT IN CORRECT FORM FOR CASE                                       PRE24630
*                                                                                   PRE24640
                INSTR = CURSTMT                                                     PRE24650
                INSTR SPAN(' ') =                                                   PRE24660
*                                                                                   PRE24670
*       BREAK OUT THE INDEX OF THE CASE                                             PRE24680
*                                                                                   PRE24690
                INSTR 'CASE OF ' BREAK(SEMDEL) . IND =   :F(CASEERROR)              PRE24700
*                                                                                   PRE24710
*       CREATE 2 LABELS; CREATE A STRING OF AN IF STATEMENT; SET KTR TO 1           PRE24720
*                                                                                   PRE24730
GEN2            VAR0 = UNILABEL()                                                   PRE24740
                VAR1 = UNILABEL()                                                   PRE24750
                INASTRUCT = INASTRUCT + 1                                           PRE24760
                NO.OF.CASES = NO.OF.CASES + 1                                       PRE24770
                $('KTR' NO.OF.CASES) = 1                                            PRE24780
            STRING = CRIF(UNILABEL(),IND,NEQU,$('KTR' NO.OF.CASES),VAR1)            PRE24790
*                                                                                   PRE24800
*       INSERT THE CURSTMT AS A COMMENT; NULLIFY CURSTMT                            PRE24810
*                                                                                   PRE24820
                INSERTCOM(CURSTMT)                                                  PRE24830
                CURSTMT =                                                           PRE24840
*                                                                                   PRE24850
*       INSERT THE CREATED STRING; PUSH THE INDEX OF THE CASE AND THE 2             PRE24860
*           CREATED LABELS ONTO THE SEMANTIC STACK                                  PRE24870
*                                                                                   PRE24880
                INSERT(STRING)                                                      PRE24890
                PUSH.SEM(3,IND,VAR1,VAR0)                                           PRE24900
                &ANCHOR = 0                                 :(RETURN)               PRE24910
ROUT3           &ANCHOR = 1                                                         PRE24920
```

```
*                                                                                PRE24930
*********************************************************************************PRE24931
*                                                                                PRE24932
*     ROUTINE FOR DO LOOP STATEMENT PROCESSING                                   PRE24940
*                                                                                PRE24950
*********************************************************************************PRE24951
*                                                                                PRE24952
*     ALL ERRORS IF NOT IN CORRECT FORM FOR DO LOOP                              PRE24960
*                                                                                PRE24970
          INSTR = CURSTMT                                                       PRE24980
          INSTR SPAN(' ') =                                                     PRE24990
*                                                                               PRE25000
*     BREAK OUT THE INDEX, BEGINNING VALUE, ENDING VALUE, AND INCREMENT          PRE25010
*         VALUE OF THE DO LOOP                                                  PRE25020
*                                                                               PRE25030
          INSTR 'DO ' BREAK(' ') . IND ' = ' =      :F(DOERROR)                 PRE25040
          INSTR ARB . BEGIN ' TO ' =                :F(DOERROR)                 PRE25050
          INSTR ARB . END ' BY ' =                  :S(INCR)                    PRE25060
          INSTR BREAK(SEMDEL) . END =               :F(DOERROR)                 PRE25070
          INC = 1                                   :(GEN3)                     PRE25080
INCR      INSTR BREAK(SEMDEL) . INC =               :F(DOERROR)                 PRE25090
*                                                                               PRE25100
*     CREATE 1 LABEL AND 1 VARIABLE; CREATE A STRING OF 2 ASSIGN MINUS           PRE25110
*         STATEMENTS AND AN ASSIGN PLUS STATEMENT                               PRE25120
*                                                                               PRE25130
GEN3      VAR1 = UNILABEL()                                                     PRE25140
          VAR2 = UNIVAR()                                                       PRE25150
          INASTRUCT = INASTRUCT + 1                                             PRE25160
          STRING = CRMINUS(UNILABEL(),VAR2,END,INC)                             PRE25170
+                  CRMINUS(UNILABEL(),IND,BEGIN,INC)                            PRE25180
+            CRPLUS(VAR1,IND,IND,INC)                                           PRE25190
*                                                                               PRE25200
*     INSERT THE CURSTMT AS A COMMENT; NULLIFY THE CURSTMT;                     PRE25210
*         INSERT THE CREATED STRING                                             PRE25220
*                                                                               PRE25230
          INSERTCOM(CURSTMT)                                                    PRE25240
          CURSTMT =                                                             PRE25250
          INSERT(STRING)                                                        PRE25260
*                                                                               PRE25270
*     PUSH THE INCREMENT, INDEX, AND VARIABLE AND LABEL CREATED IN              PRE25280
*         THIS ROUTINE ONTO THE SEMANTIC STACK                                  PRE25290
*                                                                               PRE25300
          PUSH.SEM(4,INC,IND,VAR2,VAR1)                                         PRE25310
          &ANCHOR = 0                               :(RETURN)                   PRE25320
ROUT4     &ANCHOR = 1                                                           PRE25330
*                                                                               PRE25340
*********************************************************************************PRE25341
*                                                                                PRE25342
```

```
*     ROUTINE FOR DOGROUP STATEMENT PROCESSING                                  PRE25350
*                                                                               PRE25360
*****************************************************************************PRE25361
*                                                                               PRE25362
*     ERROR IF NOT IN CORRECT FORM FOR DOGROUP                                  PRE25370
*                                                                               PRE25380
               INSTR = CURSTMT                                                  PRE25390
               INSTR SPAN(' ') =                                                PRE25400
               INSTR 'DOGROUP' ANY(SEMDEL) =              :F(DOGRPERROR)        PRE25410
*                                                                               PRE25420
*     INSERT CURSTMT AS A COMMENT; NULLIFY CURSTMT; RETURN                      PRE25430
*                                                                               PRE25440
GEN4           INSERTCOM(CURSTMT)                                               PRE25450
               INASTRUCT = INASTRUCT + 1                                        PRE25460
               CURSTMT =                                                        PRE25470
               &ANCHOR = 0                                :(RETURN)             PRE25480
ROUT5          &ANCHOR = 1                                                      PRE25490
*                                                                               PRE25500
*****************************************************************************PRE25501
*                                                                               PRE25502
*     ROUTINE FOR IF THEN STATEMENT PROCESSING                                  PRE25510
*                                                                               PRE25520
*****************************************************************************PRE25521
*                                                                               PRE25522
*     ERROR IF NOT IN CORRECT FORM FOR IF THEN                                  PRE25530
*                                                                               PRE25540
               INSTR = CURSTMT                                                  PRE25550
               INSTR SPAN(' ') =                                                PRE25560
*                                                                               PRE25570
*     BREAK OUT THE CONDITION                                                   PRE25580
*                                                                               PRE25590
               INSTR 'IF ' ARB . BOOL ' THEN ' =          :F(IFERROR)           PRE25600
*                                                                               PRE25610
*     CREATE A LABEL FOR BRANCHING TO ON FALSE; CREATE A STRING                 PRE25620
*        OF AN IF STATEMENT                                                     PRE25630
*                                                                               PRE25640
GEN5           VAR1 = UNILABEL()                                                PRE25650
               STRING = CRIF(UNILABEL(),NULL,NOT,'(' BOOL ')',VAR1)             PRE25660
*                                                                               PRE25670
*     INSERT CURSTMT AS A COMMENT; NULLIFY CURSMT                               PRE25680
*                                                                               PRE25690
               INSERTCOM(CURSTMT)                                               PRE25700
               CURSTMT =                                                        PRE25710
*                                                                               PRE25720
*     INSERT THE STRING; PUSH THE LABEL CREATED IN THIS ROUTINE                 PRE25730
*        ONTO THE SEMANTIC STACK; RETURN                                        PRE25740
*                                                                               PRE25750
               INSERT(STRING)                                                   PRE25760
```

```
                PUSH.SEM(1,VAR1)                                                PRE25770
                &ANCHOR = 0                          :(RETURN)                  PRE25780
ROUT6           &ANCHOR = 1                                                     PRE25790
*                                                                               PRE25800
*****************************************************************************PRE25801
*                                                                               PRE25802
*     ROUTINE FOR REPEAT STATEMENT PROCESSING                                   PRE25810
*                                                                               PRE25820
*****************************************************************************PRE25821
*                                                                               PRE25822
*     ERROR IF NOT IN CORRECT FORM FOR REPEAT                                   PRE25830
*                                                                               PRE25840
                INSTR = CURSTMT                                                 PRE25850
                INSTR SPAN(' ') =                                               PRE25860
                INSTR 'REPEAT UNTIL (' =             :F(REPERROR)               PRE25870
*                                                                               PRE25880
*     BREAK OUT THE CONDITION                                                   PRE25890
*                                                                               PRE25900
                INSTR ARB . BOOL ')' ANY(SEMDEL) =   :F(REPERROR)               PRE25910
*                                                                               PRE25920
*     CREATE A LABEL FOR A NULL STATEMENT; CREATE A STRING OF A NULL            PRE25930
*         STATEMENT                                                             PRE25940
*                                                                               PRE25950
GEN6            VAR1 = UNILABEL()                                               PRE25960
                INASTRUCT = INASTRUCT + 1                                       PRE25970
                STRING = CRNULL(VAR1)                                           PRE25980
*                                                                               PRE25990
*     INSERT CURSTMT AS A COMMENT; NULLIFY CURSTMT                              PRE26000
*                                                                               PRE26010
                INSERTCOM(CURSTMT)                                              PRE26020
                CURSTMT =                                                       PRE26030
*                                                                               PRE26040
*     INSERT THE STRING; PUSH THE LABEL CREATED IN THIS ROUTINE AND THE         PRE26050
*         CONDITION ONTO THE SEMANTIC STACK; RETURN                             PRE26060
*                                                                               PRE26070
                INSERT(STRING)                                                  PRE26080
                PUSH.SEM(2,VAR1,BOOL)                                           PRE26090
                &ANCHOR = 0                          :(RETURN)                  PRE26100
ROUT7           &ANCHOR = 1                                                     PRE26110
*                                                                               PRE26120
*****************************************************************************PRE26121
*                                                                               PRE26122
*     ROUTINE FOR WHILE STATEMENT PROCESSING                                    PRE26130
*                                                                               PRE26140
*****************************************************************************PRE26141
*                                                                               PRE26142
*     ERROR IF NOT IN CORRECT FORM FOR WHILE                                    PRE26150
*                                                                               PRE26160
```

```
                INSTR = CURSTMT                                                 PRE26170
                INSTR SPAN(' ') =                                               PRE26180
                INSTR 'WHILE (' =                       :F(WHERROR)             PRE26190
*                                                                               PRE26200
*       BREAK OUT THE CONDITION                                                 PRE26210
*                                                                               PRE26220
                INSTR ARB . BOOL ')' ANY(SEMDEL) =      :F(WHERROR)             PRE26230
*                                                                               PRE26240
*       CREATE 2 LABELS; CREATE A STRING OF AN IF STATEMENT                     PRE26250
*                                                                               PRE26260
GEN7            VAR1 = UNILABEL()                                               PRE26270
                VAR2 = UNILABEL()                                               PRE26280
                INASTRUCT = INASTRUCT + 1                                       PRE26290
                STRING = CRIF(VAR1,NULL,NOT,'(' BOOL ')',VAR2)                  PRE26300
*                                                                               PRE26310
*       INSERT CURSTMT AS A COMMENT; NULLIFY CURSTMT                            PRE26320
*                                                                               PRE26330
                INSERTCOM(CURSTMT)                                              PRE26340
                CURSTMT =                                                       PRE26350
*                                                                               PRE26360
*       INSERT THE STRING; PUSH THE 2 LABELS CREATED IN THIS                    PRE26370
*           ROUTINE ONTO THE SEMANTIC STACK; RETURN                             PRE26380
*                                                                               PRE26390
                INSERT(STRING)                                                  PRE26400
                PUSH.SEM(2,VAR2,VAR1)                                           PRE26410
                &ANCHOR = 0                             :(RETURN)               PRE26420
ROUT8           &ANCHOR = 1                                                     PRE26430
*                                                                               PRE26440
*********************************************************************************PRE26441
*                                                                               PRE26442
*       ROUTINE FOR QUITLOOP STATEMENT PROCESSING                               PRE26450
*                                                                               PRE26460
*********************************************************************************PRE26461
*                                                                               PRE26462
*       ERROR IF NOT IN CORRECT FOR FOR QUITLOOP                                PRE26470
*                                                                               PRE26480
                INSTR = CURSTMT                                                 PRE26490
                INSTR SPAN(' ') =                                               PRE26500
                INSTR 'QUITLOOP' ANY(SEMDEL) =          :F(QUITERROR)           PRE26510
*                                                                               PRE26520
*       CREATE A LABEL FOR BRANCHING OUT OF THE LOOP; CREATE A                  PRE26530
*       STRING OF A GO TO STATEMENT                                             PRE26540
*                                                                               PRE26550
GEN8            VAR1 = UNILABEL()                                               PRE26560
                STRING = CRGOTO(UNILABEL(),VAR1)                                PRE26570
*                                                                               PRE26580
*       INSERT CURSTMT AS A COMMENT; NULLIFY CURSTMT                            PRE26590
*                                                                               PRE26600
```

```
                INSERTCOM(CURSTMT)                                              PRE26610
                CURSTMT =                                                       PRE26620
*                                                                               PRE26630
*       INSERT THE STRING; PUSH THE LABEL CREATED IN THIS ROUTINE               PRE26640
*           AND 'QUIT' ONTO THE SEMANTIC STACK; RETURN                          PRE26650
*                                                                               PRE26660
                INSERT(STRING)                                                  PRE26670
                PUSH.SEM(2,VAR1,'QUIT')                                         PRE26680
                &ANCHOR = 0                             :(RETURN)               PRE26690
ROUT9           &ANCHOR = 1                                                     PRE26700
*                                                                               PRE26710
*****************************************************************************PRE26711
*                                                                               PRE26712
*       ROUTINE FOR ENDCASE PROCESSING                                          PRE26720
*                                                                               PRE26730
*****************************************************************************PRE26731
*                                                                               PRE26732
*       ERROR IF NOT IN CORRECT FORM FOR ENDCASE                                PRE26740
*                                                                               PRE26750
                INSTR = CURSTMT                                                 PRE26760
                INSTR SPAN(' ') =                                               PRE26770
                INSTR 'ENDCASE' ANY(SEMDEL) =           :F(ECASEERROR)          PRE26780
*                                                                               PRE26790
*       POP 2 LABELS AND THE INDEX OF THE CASE OFF THE SEMANTIC STACK           PRE26800
*                                                                               PRE26810
GEN9            POP.SEM(3,'VAR0','VAR1','IND')                                  PRE26820
*                                                                               PRE26830
*       CREATE A STRING OF 2 NULL STATEMENTS; RESET KTR TO 0                    PRE26840
*                                                                               PRE26850
                STRING = CRNULL(VAR1) CRNULL(VAR0)                              PRE26860
                $('KTR' NO.OF.CASES) = 0                                        PRE26870
                NO.OF.CASES = NO.OF.CASES - 1                                   PRE26880
                INASTRUCT = INASTRUCT - 1                                       PRE26890
*                                                                               PRE26900
*       INSERT CURSTMT AS A COMMENT; NULLIFY CURSTMT                            PRE26910
*                                                                               PRE26920
                INSERTCOM(CURSTMT)                                              PRE26930
                CURSTMT =                                                       PRE26940
*                                                                               PRE26950
*       INSERT THE STRING; RETURN                                               PRE26960
*                                                                               PRE26970
                INSERT(STRING)                                                  PRE26980
                &ANCHOR = 0                             :(RETURN)               PRE26990
ROUT10          &ANCHOR = 1                                                     PRE27000
*                                                                               PRE27010
*****************************************************************************PRE27011
*                                                                               PRE27012
*       ROUTINE FOR ENDIF PROCESSING                                            PRE27020
```

```
*                                                                                PRE27030
*********************************************************************************PRE27031
*                                                                                PRE27032
*     ERROR IF NOT IN CORRECT FORM FOR ENDIF                                     PRE27040
*                                                                                PRE27050
                INSTR = CURSTMT                                                  PRE27060
                INSTR SPAN(' ') =                                                PRE27070
                INSTR 'ENDIF' ANY(SEMDEL) =                 :F(EIFERROR)         PRE27080
*                                                                                PRE27090
*     POP AN ITEM OFF THE SEMANTIC STACK; BRANCH IF IT ISN'T 'QUIT'              PRE27100
*                                                                                PRE27110
GEN10           POP.SEM(1,'VAR1')                                                PRE27120
                VAR1 'QUIT' RPOS(0)                         :F(NOTAQUIT)         PRE27130
*                                                                                PRE27140
*     THE ITEM POPPED WAS 'QUIT', SO POP 2 MORE ITEMS OFF AND PUSH THE           PRE27150
*         'QUIT' AND FOLLOWING ITEM BACK ONTO THE STACK, LEAVING A               PRE27160
*         LABEL AVAILABLE                                                        PRE27170
*                                                                                PRE27180
                POP.SEM(2,'VAR2','VAR1')                                         PRE27190
                PUSH.SEM(2,VAR2,'QUIT')                                          PRE27200
*                                                                                PRE27210
*     CREATE A STRING OF A NULL STATEMENT                                        PRE27220
*                                                                                PRE27230
NOTAQUIT        STRING = CRNULL(VAR1)                                            PRE27240
*                                                                                PRE27250
*     INSERT CURSTMT AS A COMMENT; NULLIFY CURSTMT                               PRE27260
*                                                                                PRE27270
                INSERTCOM(CURSTMT)                                               PRE27280
                CURSTMT =                                                        PRE27290
*                                                                                PRE27300
*     INSERT THE STRING; RETURN                                                  PRE27310
*                                                                                PRE27320
                INSERT(STRING)                                                   PRE27330
                &ANCHOR = 0                                 :(RETURN)            PRE27340
ROUT11          &ANCHOR = 1                                                      PRE27350
*                                                                                PRE27360
*********************************************************************************PRE27361
*                                                                                PRE27362
*     ROUTINE FOR ENDWHILE PROCESSING                                            PRE27370
*                                                                                PRE27380
*********************************************************************************PRE27381
*                                                                                PRE27382
*     ERROR IF NOT IN CORRECT FORM FOR ENDWHILE                                  PRE27390
*                                                                                PRE27400
                INSTR = CURSTMT                                                  PRE27410
                INSTR SPAN(' ') =                                                PRE27420
                INSTR 'ENDWHILE' ANY(SEMDEL) =              :F(EWHILEERROR)      PRE27430
*                                                                                PRE27440
```

```
*     SET QUITFLAG TO 0; POP 2 ITEMS OFF THE SEMANTIC STACK;                    PRE27450
*         BRANCH IF THE FIRST ITEM ISN'T 'QUIT'                                 PRE27460
*                                                                               PRE27470
GEN11           QUITFLAG = 0                                                    PRE27480
                POP.SEM(2,'VAR1','VAR2')                                        PRE27490
                INASTRUCT = INASTRUCT - 1                                       PRE27500
                VAR1 'QUIT' RPOS(0)                          :F(ELSE11)         PRE27510
*                                                                               PRE27520
*     THE FIRST ITEM WAS 'QUIT', SO ASSIGN THE SECOND ITEM TO QUITLAB           PRE27530
*         AND SET QUITFLAG TO 1                                                 PRE27540
*                                                                               PRE27550
                QUITLAB = VAR2                                                  PRE27560
                QUITFLAG = 1                                                    PRE27570
*                                                                               PRE27580
*     POP 2 LABELS OFF THE SEMANTIC STACK                                       PRE27590
*                                                                               PRE27600
                POP.SEM(2,'VAR1','VAR2')                                        PRE27610
*                                                                               PRE27620
*     CREATE A STRING OF A GO TO STATEMENT AND A NULL STATEMENT                 PRE27630
*                                                                               PRE27640
ELSE11          STRING = CRGOTO(UNILABEL(),VAR1) CRNULL(VAR2)                   PRE27650
*                                                                               PRE27660
*     INSERT CURSTMT AS A COMMENT; NULLIFY CURSTMT                              PRE27670
*                                                                               PRE27680
                INSERTCOM(CURSTMT)                                              PRE27690
                CURSTMT =                                                       PRE27700
*                                                                               PRE27710
*     INSERT THE STRING; RETURN IF QUITFLAG IS 0                                PRE27720
*                                                                               PRE27730
                INSERT(STRING)                                                  PRE27740
                NE(QUITFLAG,1)                               :F(QUITINWHILE)    PRE27750
                &ANCHOR = 0                                  :(RETURN)          PRE27760
*                                                                               PRE27770
*     QUITFLAG IS 1, CREATE A STRING OF A NULL STATEMENT; INSERT                PRE27780
*         THE STRING; RETURN                                                    PRE27790
*                                                                               PRE27800
QUITINWHILE     STRING = CRNULL(QUITLAB)                                        PRE27810
                INSERT(STRING)                                                  PRE27820
                &ANCHOR = 0                                  :(RETURN)          PRE27830
ROUT12          &ANCHOR = 1                                                     PRE27840
*                                                                               PRE27850
*****************************************************************************PRE27851
*                                                                               PRE27852
*     ROUTINE FOR ENDREPEAT PROCESSING                                          PRE27860
*                                                                               PRE27870
*****************************************************************************PRE27871
*                                                                               PRE27872
*     ERROR IF NOT IN CORRECT FORM FOR ENDREPEAT                                PRE27880
```

```
*                                                                       PRE27890
                INSTR = CURSTMT                                         PRE27900
                INSTR SPAN(' ') =                                       PRE27910
                INSTR 'ENDREPEAT' ANY(SEMDEL) =          :F(EREPERROR)  PRE27920
*                                                                       PRE27930
*     SET QUITFLAG TO 0; POP 2 ITEMS OFF THE SEMANTIC STACK; BRANCH     PRE27940
*       IF THE FIRST ITEM ISN'T 'QUIT'                                  PRE27950
*                                                                       PRE27960
GEN12           QUITFLAG = 0                                            PRE27970
                POP.SEM(2,'BOOL','VAR1')                                PRE27980
                INASTRUCT = INASTRUCT - 1                               PRE27990
                BOOL 'QUIT' RPOS(0)                      :F(ELSE12)     PRE28000
*                                                                       PRE28010
*     THE FIRST ITEM WAS 'QUIT', SO ASSIGN THE SECOND ITEM TO QUITLAB   PRE28020
*       AND SET QUITFLAG TO 1                                           PRE28030
*                                                                       PRE28040
                QUITLAB = VAR1                                          PRE28050
                QUITFLAG = 1                                            PRE28060
*                                                                       PRE28070
*     POP A CONDITION AND A LABEL OFF THE SEMANTIC STACK; CREATE A      PRE28080
*       STRING OF AN IF STATEMENT                                       PRE28090
*                                                                       PRE28100
                POP.SEM(2,'BOOL','VAR1')                                PRE28110
ELSE12          STRING = CRIF(UNILABEL(),NULL,NOT,'(' BOOL ')',VAR1)    PRE28120
*                                                                       PRE28130
*     INERT CURSTMT AS A COMMENT; NULLIFY CURSTMT                       PRE28140
*                                                                       PRE28150
                INSERTCOM(CURSTMT)                                      PRE28160
                CURSTMT =                                               PRE28170
*                                                                       PRE28180
*     INSERT THE STRING; RETURN IF QUITFLAG IS 0                        PRE28190
*                                                                       PRE28200
                INSERT(STRING)                                          PRE28210
                NE(QUITFLAG,1)                           :F(QUITINREP)  PRE28220
                &ANCHOR = 0                              :(RETURN)      PRE28230
*                                                                       PRE28240
*     QUITFLAG IS 1; CREATE A STRING OF A NULL STATEMENT; INSERT THE    PRE28250
*       STRING; RETURN                                                  PRE28260
*                                                                       PRE28270
QUITINREP       STRING = CRNULL(QUITLAB)                                PRE28280
                INSERT(STRING)                                          PRE28290
                &ANCHOR = 0                              :(RETURN)      PRE28300
ROUT13          &ANCHOR = 1                                             PRE28310
*                                                                       PRE28320
*********************************************************************** PRE28321
*                                                                       PRE28322
*     ROUTINE FOR ENDDO PROCESSING                                      PRE28330
*                                                                       PRE28340
```

```
*  ************************************************************************PRE28341
*                                                                          PRE28342
*     ERROR IF NOT IN CORRECT FORM FOR ENDDO                               PRE28350
*                                                                          PRE28360
                INSTR = CURSTMT                                           PRE28370
                INSTR SPAN(' ') =                                         PRE28380
                INSTR 'ENDDO' ANY(SEMDEL) =                :F(EDOERROR)   PRE28390
*                                                                         PRE28400
*     SET QUITFLAG TO 0; POP 4 ITEMS OFF THE SEMANTIC STACK; BRANCH IF    PRE28410
*        THE FIRST ITEM ISN'T 'QUIT'                                      PRE28420
*                                                                         PRE28430
GEN13           QUITFLAG = 0                                              PRE28440
                POP.SEM(4,'VAR1','VAR2','IND','INC')                      PRE28450
                INASTRUCT = INASTRUCT - 1                                 PRE28460
                VAR1 'QUIT' RPOS(0)                        :F(ELSE13)     PRE28470
*                                                                         PRE28480
*     THE FIRST ITEM WAS 'QUIT', SO ASSIGN THE SECOND ITEM TO             PRE28490
*        QUITLAB                                                          PRE28500
*                                                                         PRE28510
                QUITLAB = VAR2                                            PRE28520
*                                                                         PRE28530
*     ASSIGN THE THIRD AND FOURTH ITEMS TO VAR1 AND VAR2                  PRE28540
*                                                                         PRE28550
                VAR1 = IND                                                PRE28560
                VAR2 = INC                                                PRE28570
*                                                                         PRE28580
*     POP 2 MORE ITEMS OFF THE SEMANTIC STACK AND SET QUITFLAG TO 1       PRE28590
*                                                                         PRE28600
                POP.SEM(2,'IND','INC')                                    PRE28610
                QUITFLAG = 1                                              PRE28620
*                                                                         PRE28630
*     CREATE 2 LABELS; CREATE A STRING OF 2 IF STATEMENTS, A GO TO        PRE28640
*        STATEMENT, AN IF STATEMENT, AND AN ASSIGN PLUS STATEMENT         PRE28650
*                                                                         PRE28660
ELSE13          VAR3 = UNILABEL()                                         PRE28670
                VAR4 = UNILABEL()                                         PRE28680
                STRING = CRIF(UNILABEL(),INC,GTHAN,'0',VAR3)              PRE28690
+                             CRIF(UNILABEL(),IND,GEQU,VAR2,VAR1)         PRE28700
+                             CRGOTO(UNILABEL(),VAR4)                     PRE28710
+                             CRIF(VAR3,IND,LEQU,VAR2,VAR1)               PRE28720
+                             CRPLUS(VAR4,IND,IND,INC)                    PRE28730
*                                                                         PRE28740
*     INSERT CURSTMT AS A COMMENT; NULLIFY CURSTMT                        PRE28750
*                                                                         PRE28760
                INSERTCOM(CURSTMT)                                        PRE28770
                CURSTMT =                                                 PRE28780
*                                                                         PRE28790
*     INSERT STRING; RETURN IF QUITFLAG IS 0                              PRE28800
```

```
*                                                                                PRE28810
                INSERT(STRING)                                                   PRE28820
                NE(QUITFLAG,1)                          :F(QUITINDO)             PRE28830
                &ANCHOR = 0                             :(RETURN)                PRE28840
*                                                                                PRE28850
*     QUITFLAG IS 1; CREATE A STRING OF A NULL STATEMENT; INSERT THE             PRE28860
*          STRING; RETURN                                                        PRE28870
*                                                                                PRE28880
QUITINDO        STRING = CRNULL(QUITLAB)                                         PRE28890
                INSERT(STRING)                                                   PRE28900
                &ANCHOR = 0                             :(RETURN)                PRE28910
ROUT14          &ANCHOR = 1                                                      PRE28920
*                                                                                PRE28930
**********************************************************************************PRE28931
*                                                                                PRE28932
*     ROUTINE FOR ENDDOGROUP PROCESSING                                          PRE28940
*                                                                                PRE28950
**********************************************************************************PRE28951
*                                                                                PRE28952
*     ERROR IF NOT IN CORRECT FORM FOR ENDDOGROUP                                PRE28960
*                                                                                PRE28970
                INSTR = CURSTMT                                                  PRE28980
                INSTR SPAN(' ') =                                                PRE28990
                INSTR 'ENDDOGROUP' ANY(SEMDEL) =        :F(FDOGRPERROR)          PRE29000
*                                                                                PRE29010
*     INSERT CURSTMT AS A COMMENT; NULLIFY CURSTMT; RETURN                       PRE29020
*                                                                                PRE29030
                INSERTCOM(CURSTMT)                                               PRE29040
                INASTRUCT = INASTRUCT - 1                                        PRE29050
                CURSTMT =                                                        PRE29060
                &ANCHOR = 0                             :(RETURN)                PRE29070
ROUT15          &ANCHOR = 1                                                      PRE29080
*                                                                                PRE29090
**********************************************************************************PRE29091
*                                                                                PRE29092
*     ROUTINE FOR STRUC OR STRLIST STRUC TO STRLIST REDUCTION                    PRE29100
*                                                                                PRE29110
**********************************************************************************PRE29111
*                                                                                PRE29112
*     RETURN IF KTR = 0 (NOT WITHIN A CASE STRUCTURE) OR IF                      PRE29120
*          INASTRUCT > NO.OF.CASES (WITHIN A 'PRIMITIVE' STRUCTURE);             PRE29130
*          OTHERWISE, INCREMENT KTR BY 1                                         PRE29140
*                                                                                PRE29150
GEN15           GT($('KTR' NO.OF.CASES),0)              :S(INACASE)              PRE29160
                &ANCHOR = 0                             :(RETURN)                PRE29170
INACASE         GT(INASTRUCT,NO.OF.CASES)               :F(AFTERSTRUC)           PRE29180
                &ANCHOR = 0                             :(RETURN)                PRE29190
AFTERSTRUC      $('KTR' NO.OF.CASES) = $('KTR' NO.OF.CASES) + 1                  PRE29200
```

```
*                                                                       PRE29210
*     CREATE A LABEL; POP 3 ITEMS OFF THE SEMANTIC STACK                PRE29220
*                                                                       PRE29230
               VAR3 = UNILABEL()                                        PRE29240
               POP.SEM(3,'VAR0','VAR1','IND')                           PRE29250
*                                                                       PRE29260
*     CREATE A STRING OF A GO TO STATEMENT AND IF AN STATEMENT          PRE29270
*                                                                       PRE29280
               STRING = CRGOTO(UNILABEL(),VAR0)                         PRE29290
+                   CRIF(VAR1,IND,NEQU,$('KTR' NO.OF.CASES),VAR3)       PRE29300
*                                                                       PRE29310
*     INSERT A COMMENT ABOUT THIS ROUTINE'S RESULTS                     PRE29320
*                                                                       PRE29330
          INSERTCOM('NEXT TWO STATEMENTS INSERTED FOR CASE PROCESSING')PRE29340
*                                                                       PRE29350
*     INSERT THE STRING; PUSH THE INDEX OF THE CASE AND THE 2           PRE29360
*         LABELS CREATED IN THIS ROUTINE ONTO THE SEMANTIC STACK;       PRE29370
*         RETURN                                                        PRE29380
*                                                                       PRE29390
               INSERT(STRING)                                           PRE29400
               PUSH.SEM(3,IND,VAR3,VAR0)                                PRE29410
               &ANCHOR = 0                          :(RETURN)           PRE29420
INSERTCOM   I = 1                                                       PRE29430
*                                                                       PRE29440
*************************************************************************PRE29441
*                                                                       PRE29442
*     ROUTINE TO INSERT A STRING AS A COMMENT                           PRE29450
*                                                                       PRE29460
*     ONE PARAMETER -- COMSTR, THE STRING TO BE MADE INTO A             PRE29470
*         COMMENT AND INSERTED                                          PRE29480
*                                                                       PRE29490
*************************************************************************PRE29491
*                                                                       PRE29492
*     FIND THE FIRST AVAILABLE COLUMN FOR INSERTING THE COMMENT         PRE29500
*                                                                       PRE29510
               IGNPAT<0> 'SEQ'                          :S(COLSEQ)      PRE29520
               I = EQ(I,IGNPAT<1>) I + 1                :(FOUNDCOLOPEN) PRE29530
COLSEQ         GT(I,IGNPAT<2>)                          :S(FOUNDCOLOPEN)PRE29540
               LT(I,IGNPAT<1>)                          :S(FOUNDCOLOPEN)PRE29550
INCR.I         I = LE(I,IGNPAT<2>) I + 1                :S(INCR.I)      PRE29560
*                                                                       PRE29570
*     FILL THE COLUMNS PRECEDING THE COLUMN FOUND WITH BLANKS,          PRE29580
*         BRANCH BASED ON THE COMMENT DEFINITION                        PRE29590
*                                                                       PRE29600
FOUNDCOLOPEN OUTCOM = DUPL(' ',I - 1)                                   PRE29610
               COMPAT<1> '10'                           :S(ADDOPENSTR)  PRE29620
*                                                                       PRE29630
*     A COMMENT BEGINS IN A CERTAIN COLUMN, FILL WITH BLANKS TO THAT    PRE29640
```

```
*          COLUMN                                                                    PRE29650
*                                                                                    PRE29660
            I = COMPAT<4> - I                                                       PRE29670
            OUTCOM = OUTCOM DUPL(' ',I)                                             PRE29680
            CCMPAT<1> '11'                                  :F(ADDCOMSTR)           PRE29690
*                                                                                   PRE29700
*     A COMMENT BEGINS WITH A CERTAIN STRING IN THAT PARTICULAR                     PRE29710
*        COLUMN, ADD THE STRING TO THE OUTPUT COMMENT STRING                        PRE29720
*                                                                                   PRE29730
ADDOPENSTR OUTCOM = OUTCOM COMPAT<3>                                                PRE29740
*                                                                                   PRE29750
*     ADD THE PARAMETER STRING TO THE OUTPUT COMMENT STRING,                        PRE29760
*        CHECK FOR THE OUTPUT STRING BEING LONGER THAN 80 CHARACTERS                PRE29770
*                                                                                   PRE29780
ADDCOMSTR  OUTCOM = OUTCOM COMSTR                                                   PRE29790
            GT(SIZE(OUTCOM),80)                             :F(NOTRUNCATE)          PRE29800
*                                                                                   PRE29810
*     TRUNCATE TO 80 CHARACTERS                                                     PRE29820
*                                                                                   PRE29830
            OUTCOM = SUBSTR(OUTCOM,1,80)                                            PRE29840
NOTRUNCATE CCMPAT<2> '01'                                   :F(ENDCOMSTR)           PRE29850
*                                                                                   PRE29860
*     A COMMENT ENDS IN A CERTAIN COLUMN, ADD BLANKS IF THE COMMENT                 PRE29870
*        IS TOO SHORT, TRUNCATE IF IT IS TOO LONG, BRANCH TO                        PRE29880
*        THE COMMENT FINISHED LABEL                                                 PRE29890
*                                                                                   PRE29900
            GT(SIZE(OUTCOM),COMPAT<6>)                      :S(COLTRUNC)            PRE29910
            OUTCOM = OUTCOM DUPL(' ',80 - SIZE(OUTCOM))     :(COMFIN)               PRE29920
COLTRUNC    OUTCOM = SUBSTR(OUTCOM,1,CCMPAT<6>)             :(COMFIN)               PRE29930
ENDCOMSTR  COMPAT<2> '11'                                   :S(COMENDSTRCOL)        PRE29940
*                                                                                   PRE29950
*     A COMMENT ENDS WITH A CERTAIN STRING, IF THE STRING WON'T FIT                 PRE29960
*        ONTO THE OUTPUT STRING AND STILL HAVE THE STRING BE                        PRE29970
*        80 CHARACTERS OR LESS, TRUNCATE THE CURRENT OUTPUT STRING                  PRE29980
*                                                                                   PRE29990
            GT(SIZE(OUTCOM),80 - SIZE(COMPAT<5>))           :F(NOSYMTRUNC)          PRE30000
            OUTCOM = SUBSTR(OUTCOM,1,80 - SIZE(COMPAT<5>))                          PRE30010
*                                                                                   PRE30020
*     ADD THE TERMINATING STRING, SET I TO THE COLUMN NUMBER THE                    PRE30030
*        STRING MUST START IN TO END EXACTLY IN COLUMN 80, BRANCH                   PRE30040
*        ACCORDING TO THE IGNORE COLUMN DEFINITION SPECIFYING                       PRE30050
*        A SINGLE COLUMN OR A SEQUENCE OF COLUMNS                                   PRE30060
*                                                                                   PRE30070
NOSYMTRUNC OUTCOM = OUTCOM COMPAT<5>                                                PRE30080
            I = 81 - SIZE(CCMPAT<5>)                                                PRE30090
            IGNPAT<0> 'SEQ'                                 :S(COLSEQEND)           PRE30100
*                                                                                   PRE30110
*     DECREMENT I IF THE STRING WOULD END IN AN IGNORE COLUMN AS IT                 PRE30120
```

```
*          IS AND BRANCH                                                        PRE30130
*                                                                               PRE30140
            I = EQ(I,IGNPAT<1>) I - SIZE(COMPAT<5>)     :S(FOUNDCOLCLOSE)       PRE30150
*                                                                               PRE30160
*     DECREMENT I UNTIL THE STRING IS SURE TO END IN A NON-IGNORED              PRE30170
*          COLUMN                                                               PRE30180
*                                                                               PRE30190
COLSEQEND   GT(I,IGNPAT<2>)                                 :S(FOUNDCOLCLOSE)   PRE30200
            LT(I + SIZE(COMPAT<5>) - 1,IGNPAT<1>)           :S(FOUNDCOLCLOSE)   PRE30210
DECR.I      I = GE(I + SIZE(COMPAT<5>) - 1,IGNPAT<1>) I - 1                     PRE30220
+                                                           :S(DECR.I)          PRE30230
*                                                                               PRE30240
*     ADD THE COMMENT TERMINATING STRING TO THE OUTPUT STRING AND               PRE30250
*          ADD BLANKS AS NECESSARY TO FILL 80 COLUMNS                           PRE30260
*                                                                               PRE30270
FOUNDCOLCLOSE OUTCOM = SUBSTR(OUTCOM,1,I - 1) COMPAT<5>                         PRE30280
+                   DUPL(' ',81 - I - SIZE(COMPAT<5>)) :(COMFIN)                PRE30290
COMENDSTRCOL GT(SIZE(OUTCOM),COMPAT<6>)                     :S(COLSYMTRUNC)     PRE30300
            OUTCOM = OUTCOM DUPL(' ',COMPAT<6> - 1 - SIZE(OUTCOM))              PRE30310
+                                                           :(ADDCOLSYM)        PRE30320
COLSYMTRUNC OUTCOM = SUBSTR(OUTCOM,1,COMPAT<6> - 1)                             PRE30330
ADDCOLSYM   OUTCOM = OUTCOM COMPAT<5>                                           PRE30340
+                   DUPL(' ',80 - SIZE(OUTCOM) - SIZE(COMPAT<5>))               PRE30350
*                                                                               PRE30360
*     PRINT AND PUNCH THE COMMENT AND RETURN                                    PRE30370
*                                                                               PRE30380
COMFIN      OUTPUT = OUTCOM                                                     PRE30390
            PUNCH = OUTCOM                                                      PRE30400
            INSERTCOM =                                     :(RETURN)           PRE30410
INSERTSTMT  OUTSTR =                                                            PRE30420
*                                                                               PRE30430
*****************************************************************************PRE30431
*                                                                               PRE30432
*     ROUTINE TO INSERT A STRING AS A STATEMENT -- IGNORED COLUMNS              PRE30440
*          MUST BE SKIPPED                                                      PRE30450
*                                                                               PRE30460
*     ONE PARAMETER -- STR.INSRT, THE STRING TO BE INSERTED                     PRE30470
*          AS A STATEMENT                                                       PRE30480
*                                                                               PRE30490
*****************************************************************************PRE30491
*                                                                               PRE30492
            INSERTSTMT =                                                        PRE30500
            I = 0                                                               PRE30510
            CHARCTR = 0                                                         PRE30520
*                                                                               PRE30530
*     SET TYPE APPROPRIATELY BASED ON A SINGLE COLUMN OR A                      PRE30540
*          SEQUENCE OF COLUMNS TO BE IGNORED                                    PRE30550
*                                                                               PRE30560
```

```
            IGNPAT<0> 'SEQ'                                    :F(SINGLEIGN)      PRE30570
            TYPE = 'SEQ'                                       :(INCREMENT)       PRE30580
SINGLEIGN   TYPE = 'ONE'                                                          PRE30590
*                                                                                 PRE30600
*     CHARKTR COUNTS THE POSITION IN THE PARAMETER STRING                         PRE30610
*                                                                                 PRE30620
INCREMENT   CHARKTR = CHARKTR + 1                                                 PRE30630
*                                                                                 PRE30640
*     I COUNTS THE POSITION IN THE OUTPUT STRING                                  PRE30650
*                                                                                 PRE30660
*                                                                                 PRE30670
INCREMENT2  I = I + 1
            LE(CHARKTR,SIZE(STR.INSRT))                        :F(ENDSTMTOUT)     PRE30680
            LE(I,80)                                           :F(ENDCARD)        PRE30690
*                                                                                 PRE30700
*     BRANCH TO CHECK FOR IGNORING COLUMNS                                        PRE30710
*                                                                                 PRE30720
                                                               :($('CKIGN' TYPE)) PRE30730
*                                                                                 PRE30740
*     CHECK FOR IGNORING THIS COLUMN WHEN ONLY ONE COLUMN IS IGNORED              PRE30750
*                                                                                 PRE30760
CKIGNONE    NE(I,IGNPAT<1>)                          :S(GOODCOL)F(IGNOREIT)     PRE30770
*                                                                                 PRE30780
*     CHECK FOR IGNORING THIS COLUMN WHEN A SEQUENCE OF COLUMNS                   PRE30790
*          IS IGNORED                                                           PRE30800
*                                                                                 PRE30810
CKIGNSEQ    GT(I,IGNPAT<2>)                                    :S(GOODCOL)        PRE30820
            LT(I,IGNPAT<1>)                                    :S(GOODCOL)        PRE30830
*                                                                                 PRE30840
*     ADD A BLANK TO THE OUTPUT STRING IN AN IGNORED COLUMN AND                   PRE30850
*          BRANCH TO PROCESS THE NEXT CHARACTER                                 PRE30860
*                                                                                 PRE30870
IGNOREIT    OUTSTR = OUTSTR ' '                                :(INCREMENT2)      PRE30880
*                                                                                 PRE30890
*     ADD THE CURRENT CHARACTER TO THE OUTPUT STRING AND BRANCH                   PRE30900
*          TO PROCESS THE NEXT CHARACTER                                        PRE30910
*                                                                                 PRE30920
GOODCOL     OUTSTR = OUTSTR SUBSTR(STR.INSRT,CHARKTR,1)        :(INCREMENT)       PRE30930
*                                                                                 PRE30940
*     OUTPUT CARD IS FULL, PUNCH AND PRINT IT, REINITIALIZE                       PRE30950
*          COLUMN COUNTER AND OUTPUT STRING                                     PRE30960
*                                                                                 PRE30970
ENDCARD     OUTPUT = OUTSTR                                                       PRE30980
            PUNCH = OUTSTR                                                        PRE30990
            I = 1                                                                 PRE31000
            OUTSTR =                                           :($('CKIGN' TYPE)) PRE31010
*                                                                                 PRE31020
*     END OF PARAMETER STRING, ADD ENOUGH BLANKS TO FILL                          PRE31030
*          A CARD, PUNCH AND PRINT IT, AND RETURN                               PRE31040
```

```
*                                                                       PRE31050
ENDSTMTOUT  OUTSTR = OUTSTR DUPL(' ',80 - SIZE(OUTSTR))                 PRE31060
            OUTPUT = OUTSTR                                             PPE31070
            PUNCH = OUTSTR                          :(RETURN)           PRE31080
INSERT      PUNCH = INSRT.STR                                           PRE31090
*                                                                       PRE31100
*********************************************************************** PRE31101
*                                                                       PRE31102
*     ROUTINE TO INSERT A STRING WHICH IS ALREADY IN THE FORM OF        PRE31110
*          A STATEMENT                                                  PRE31120
*                                                                       PRE31130
*     ONE PARAMETER -- INSRT.STR, THE STATEMENT TO BE INSERTED          PRE31140
*                                                                       PRE31150
*********************************************************************** PRE31151
*                                                                       PRE31152
*     PRINT AND PUNCH THE STATEMENT AND RETURN                          PRE31160
*                                                                       PRE31170
MOREOUT     GT(SIZE(INSRT.STR),80)                  :S(OUT80)           PRE31180
            OUTPUT = INSRT.STR DUPL(' ',80 - SIZE(INSRT.STR))           PRE31190
+                                                   :(RETURN)           PRE31200
OUT80       OUTPUT = SUBSTR(INSRT.STR,1,80)                             PRE31210
            INSRT.STR = SUBSTR(INSRT.STR,81,SIZE(INSRT.STR) - 80)       PRE31220
+                                                   :(MOREOUT)          PRE31230
CRIF        CRIF =                                                      PRE31240
*                                                                       PRE31250
*********************************************************************** PRE31251
*                                                                       PRE31252
*     ROUTINE TO CREATE AN IF STATEMENT                                 PRE31260
*                                                                       PRE31270
*     FIVE PARAMETERS -- LAB, VAL1, RL, VAL2, AND DES                   PRE31280
*                                                                       PRE31290
*********************************************************************** PRE31291
*                                                                       PRE31292
            PARMNAME = 'LAB' | 'VAL1' | 'RL' | 'VAL2' | 'DES'           PRE31300
            I = 0                                                       PRE31310
            STMTLEN = IFPAT<0>                                          PRE31320
*                                                                       PRE31330
*     ERROR IF NUMBER OF STATEMENT COMPONENTS IS <= 0                   PRE31340
*                                                                       PRE31350
            LE(STMTLEN,0)                           :S(CRIFERR)         PRE31360
CRIFLOOP    LT(I,STMTLEN)                           :F(IFRETSEQ)        PRE31370
            I = I + 1                                                   PRE31380
            IFPAT<I> PARMNAME . SELECTPARM SPAN(' ') RPOS(0)            PRE31390
+                                                   :S(COPYIFPARM)      PRE31400
            IFPAT<I> PARMNAME . SELECTPARM RPOS(0)  :F(COPYCRIF)        PRE31410
            CRIF = CRIF $SELECTPARM                 :(CRIFLOOP)         PRE31420
*                                                                       PRE31430
*     ERROR IF NOT ENOUGH BLANKS LEFT AFTER PARAMETER NAME IN PROTOTYPE PRE31440
```

```
*                                                                       PRE31450
COPYIFPARM LT(SIZE(IFPAT<I>) - SIZE($SELECTPARM),0)    :S(IFBLKERR)     PRE31460
*                                                                       PRE31470
*     COPY PARAMETER AND ENOUGH BLANKS TO EQUAL FIELD LENGTH IN PROTOTYPEPRE31480
*                                                                       PRE31490
          CRIF = CRIF $SELECTPARM DUPL(' ',SIZE(IFPAT<I>) -             PRE31500
+                              SIZE($SELECTPARM))          :(CRIFLOOP)   PRE31510
COPYCRIF  CRIF = CRIF IFPAT<I>                            :(CRIFLOOP)   PRE31520
*                                                                       PRE31530
*     ADD BLANKS TO MAKE 80 COLUMNS, ERROR IF MORE THAN 80 COLUMNS      PRE31540
*          ALREADY                                                      PRE31550
*                                                                       PRE31560
IFRETSEQ  CRIF = CRIF DUPL(' ',80 - SIZE(CRIF)) :S(RETURN)F(CRIFERR)    PRE31570
CRDCL     CRDCL =                                                       PRE31580
*                                                                       PRE31590
*********************************************************************** PRE31591
*                                                                       PRE31592
*     ROUTINE TO CREATE DECLARATION STATEMENT                           PRE31600
*                                                                       PRE31610
*     ONE PARAMETER -- VARNAME                                          PRE31620
*                                                                       PRE31630
*********************************************************************** PRE31631
*                                                                       PRE31632
          I = 0                                                         PRE31640
          STMTLEN = DCLPAT<0>                                           PRE31650
*                                                                       PRE31660
*     ERROR IF NUMBER OF STATEMENT COMPONENTS IS <= 0                   PRE31670
*                                                                       PRE31680
          LE(STMTLEN,0)                               :S(CRDCLERR)      PRE31690
CRDCLLOOP LT(I,STMTLEN)                               :F(DCLRETSEQ)     PRE31700
          I = I + 1                                                     PRE31710
          DCLPAT<I> 'VARNAME' SPAN(' ') RPOS(0)       :S(COPYDCLPARM)   PRE31720
          DCLPAT<I> 'VARNAME' RPOS(0)                 :F(COPYCRDCL)     PRE31730
          CRDCL = CRDCL VARNAME                       :(CRDCLLOOP)      PRE31740
*                                                                       PRE31750
*     ERROR IF NOT ENOUGH BLANKS LEFT AFTER PARAMETER NAME IN PROTOTYPE PRE31760
*                                                                       PRE31770
COPYDCLPARM LT(SIZE(DCLPAT<I>) - SIZE(VARNAME),0)     :S(DCLBLKERR)     PRE31780
*                                                                       PRE31790
*     COPY PARAMETER AND ENOUGH BLANKS TO EQUAL FIELD LENGTH IN PROTOTYPEPRE31800
*                                                                       PRE31810
          CRDCL = CRDCL VARNAME DUPL(' ',SIZE(DCLPAT<I>) -              PRE31820
+                              SIZE(VARNAME))              :(CRDCLLOOP)  PRE31830
COPYCRDCL CRDCL = CRDCL DCLPAT<I>                         :(CRDCLLOOP)  PRE31840
*                                                                       PRE31850
*     ADD BLANKS TO MAKE 80 COLUMNS, ERROR IF MORE THAN 80 COLUMNS      PRE31860
*          ALREADY                                                      PRE31870
*                                                                       PRE31880
```

```
DCLRETSEQ  CRDCL = CRDCL DUPL(' ',80 - SIZE(CRDCL))                    PRE31890
+                                          :S(RETURN)F(CRDCLERR)       PRE31900
CRMINUS    CRMINUS =                                                   PRE31910
*                                                                      PRE31920
*********************************************************************PRE31921
*                                                                      PRE31922
*     ROUTINE TO CREATE AN ASSIGN MINUS STATEMENT                      PRE31930
*                                                                      PRE31940
*     FOUR PARAMETERS -- LAB, VAR, VAL1, AND VAL2                      PRE31950
*                                                                      PRE31960
*********************************************************************PRE31961
*                                                                      PRE31962
           PARMNAME = 'LAB' | 'VAR' | 'VAL1' | 'VAL2'                  PRE31970
           I = 0                                                       PRE31980
           STMTLEN = MINUSPAT<0>                                       PRE31990
*                                                                      PRE32000
*     ERROR IF NUMBER OF STATEMENT COMPONENTS IS <= 0                  PRE32010
*                                                                      PRE32020
           LE(STMTLEN,0)                          :S(CRMINERR)         PRE32030
CRMINUSLOOP LT(I,STMTLEN)                         :F(MINRETSEQ)        PRE32040
           I = I + 1                                                   PRE32050
           MINUSPAT<I> PARMNAME . SELECTPARM SPAN(' ') RPOS(0)         PRE32060
+                                              :S(COPYMINPARM)         PRE32070
           MINUSPAT<I> PARMNAME . SELECTPARM RPOS(0)  :F(COPYCRMINUS)  PRE32080
           CRMINUS = CRMINUS $SELECTPARM              :(CRMINUSLOOP)   PRE32090
*                                                                      PRE32100
*     ERROR IF NOT ENOUGH BLANKS LEFT AFTER PARAMETER NAME IN PROTOTYPE PRE32110
*                                                                      PRE32120
COPYMINPARM LT(SIZE(MINUSPAT<I>) - SIZE($SELECTPARM),0) :S(MINBLKERR)  PRE32130
*                                                                      PRE32140
*     COPY PARAMETER AND ENOUGH BLANKS TO EQUAL FIELD LENGTH IN PROTOTYPEPRE32150
*                                                                      PRE32160
           CRMINUS = CRMINUS $SELECTPARM DUPL(' ',SIZE(MINUSPAT<I>) -  PRE32170
+                                 SIZE($SELECTPARM))   :(CRMINUSLOOP)  PRE32180
COPYCRMINUS CRMINUS = CRMINUS MINUSPAT<I>               :(CRMINUSLOOP) PRE32190
*                                                                      PRE32200
*     ADD BLANKS TO MAKE 80 COLUMNS, ERROR IF MORE THAN 80 COLUMNS     PRE32210
*          ALREADY                                                     PRE32220
*                                                                      PRE32230
MINRETSEQ  CRMINUS = CRMINUS DUPL(' ',80 - SIZE(CRMINUS))              PRE32240
+                                          :S(RETURN)F(CRMINERR)       PRE32250
CRPLUS     CRPLUS =                                                    PRE32260
*                                                                      PRE32270
*********************************************************************PRE32271
*                                                                      PRE32272
*     ROUTINE TO CREATE AN ASSIGN PLUS STATEMENT                       PRE32280
*                                                                      PRE32290
*     FOUR PARAMETERS -- LAB, VAR, VAL1, AND VAL2                      PRE32300
```

```
*                                                                                PRE32310
*********************************************************************************PRE32311
*                                                                                PRE32312
          PARMNAME = 'LAB' | 'VAR' | 'VAL1' | 'VAL2'                             PRE32320
          I = 0                                                                  PRE32330
          STMTLEN = PLUSPAT<0>                                                   PRE32340
*                                                                                PRE32350
*     ERROR IF NUMBER OF STATEMENT COMPONENTS IS <= 0                            PRE32360
*                                                                                PRE32370
          LE(STMTLEN,0)                                  :S(CRPLERR)             PRE32380
CRPLUSLOOP LT(I,STMTLEN)                                 :F(PLUSRETSEQ)          PRE32390
          I = I + 1                                                              PRE32400
          PLUSPAT<I> PARMNAME . SELECTPARM SPAN(' ') RPOS(0)                     PRE32410
+                                                        :S(COPYPLUSPARM)        PRE32420
          PLUSPAT<I> PARMNAME . SELECTPARM RPOS(0)       :F(COPYCRPLUS)          PRE32430
          CRPLUS = CRPLUS $SELECTPARM                    :(CRPLUSLOOP)           PRE32440
*                                                                                PRE32450
*     ERROR IF NOT ENOUGH BLANKS LEFT AFTER PARAMETER NAME IN PROTOTYPE          PRE32460
*                                                                                PRE32470
COPYPLUSPARM LT(SIZE(PLUSPAT<I>) - SIZE($SELECTPARM),0) :S(PLUSBLKERR)           PRE32480
*                                                                                PRE32490
*     COPY PARAMETER AND ENOUGH BLANKS TO EQUAL FIELD LENGTH IN PROTOTYPE        PRE32500
*                                                                                PRE32510
          CRPLUS = CRPLUS $SELECTPARM DUPL(' ',SIZE(PLUSPAT<I>) -                PRE32520
+                                 SIZE($SELECTPARM))   :(CRPLUSLOOP)             PRE32530
COPYCRPLUS CRPLUS = CRPLUS PLUSPAT<I>                    :(CRPLUSLOOP)           PRE32540
*                                                                                PRE32550
*     ADD BLANKS TO MAKE 80 COLUMNS, ERROR IF MORE THAN 80 COLUMNS               PRE32560
*         ALREADY                                                                PRE32570
*                                                                                PRE32580
PLUSRETSEQ CRPLUS = CRPLUS DUPL(' ',80 - SIZE(CRPLUS))                           PRE32590
+                                                    :S(RETURN)F(CRPLERR)        PRE32600
CRNULL    CRNULL =                                                               PRE32610
*                                                                                PRE32620
*********************************************************************************PRE32621
*                                                                                PRE32622
*     ROUTINE TO CREATE A NULL STATEMENT                                         PRE32630
*                                                                                PRE32640
*     ONE PARAMETER -- LAB                                                       PRE32650
*                                                                                PRE32660
*********************************************************************************PRE32661
*                                                                                PRE32662
          I = 0                                                                  PRE32670
          STMTLEN = NULLPAT<0>                                                   PRE32680
*                                                                                PRE32690
*     ERROR IF NUMBER OF STATEMENT COMPONENTS IS <= 0                            PRE32700
*                                                                                PRE32710
          LE(STMTLEN,0)                                  :S(CRNULLERR)           PRE32720
```

```
CRNULLLOOP LT(I,STMTLEN)                                        :F(NULLRETSEQ)      PRE32730
           I = I + 1                                                                PRE32740
           NULLPAT<I> 'LAB' SPAN(' ') RPOS(0)                   :S(COPYNULLPARM)    PRE32750
           NULLPAT<I> 'LAB' RPOS(0)                             :F(COPYCRNULL)      PRE32760
           CRNULL = CRNULL LAB                                  :(CRNULLLOOP)       PRE32770
*                                                                                   PRE32780
*     ERROR IF NOT ENOUGH BLANKS LEFT AFTER PARAMETER NAME IN PROTOTYPE             PRE32790
*                                                                                   PRE32800
COPYNULLPARM LT(SIZE(NULLPAT<I>) - SIZE(LAB),0)                 :S(NULLBLKERR)      PRE32810
*                                                                                   PRE32820
*     COPY PARAMETER AND ENOUGH BLANKS TO EQUAL FIELD LENGTH IN PROTOTYPE           PRE32830
*                                                                                   PRE32840
           CRNULL = CRNULL LAB DUPL(' ',SIZE(NULLPAT<I>) - SIZE(LAB))               PRE32850
+                                                               :(CRNULLLOOP)       PRE32860
COPYCRNULL CRNULL = CRNULL NULLPAT<I>                           :(CRNULLLOOP)       PRE32870
*                                                                                   PRE32880
*     ADD BLANKS TO MAKE 80 COLUMNS, ERROR IF MORE THAN 80 COLUMNS                  PRE32890
*       ALREADY                                                                     PRE32900
*                                                                                   PRE32910
NULLRETSEQ CRNULL = CRNULL DUPL(' ',80 - SIZE(CRNULL))                              PRE32920
+                                                   :S(RETURN)F(CRNULLERR)          PRE32930
CRGOTO     CRGOTO =                                                                 PRE32940
*                                                                                   PRE32950
*********************************************************************************PRE32951
*                                                                                   PRE32952
*     ROUTINE TO CREATE A GO TO STATEMENT                                           PRE32960
*                                                                                   PRE32970
*     TWO PARAMETERS -- LAB AND DES                                                 PRE32980
*                                                                                   PRE32990
*********************************************************************************PRE32991
*                                                                                   PRE32992
           PARMNAME = 'LAB' | 'DES'                                                 PRE33000
           I = 0                                                                    PRE33010
           STMTLEN = GOTOPAT<0>                                                     PRE33020
*                                                                                   PRE33030
*     ERROR IF NUMBER OF STATEMENT COMPONENTS IS <= 0                               PRE33040
*                                                                                   PRE33050
           LE(STMTLEN,0)                                        :S(CRGOTOERR)       PRE33060
CRGOTOLOOP LT(I,STMTLEN)                                        :F(GOTORETSEQ)      PRE33070
           I = I + 1                                                                PRE33080
           GOTOPAT<I> PARMNAME . SELECTPARM SPAN(' ') RPOS(0)                       PRE33090
+                                                               :S(COPYGOTOPARM)    PRE33100
           GOTOPAT<I> PARMNAME . SELECTPARM RPOS(0)             :F(COPYCRGOTO)      PRE33110
           CRGOTO = CRGOTO $SELECTPARM                          :(CRGOTOLOOP)       PRE33120
*                                                                                   PRE33130
*     ERROR IF NOT ENOUGH BLANKS LEFT AFTER PARAMETER NAME IN PROTOTYPE             PRE33140
*                                                                                   PRE33150
COPYGOTOPARM LT(SIZE(GOTOPAT<I>) - SIZE($SELECTPARM),0) :S(GOTOBLKERR)               PRE33160
```

```
*                                                                                PRE33170
*     COPY PARAMETER AND ENOUGH BLANKS TO EQUAL FIELD LENGTH IN PROTOTYPEPRE33180
*                                                                                PRE33190
          CRGOTO = CRGOTO $SELECTPARM DUPL(' ',SIZE(GOTOPAT<I>) -        PRE33200
+                    SIZE($SELECTPARM))                :(CRGOTOLOOP)       PRE33210
COPYCRGOTO CRGOTO = CRGOTO GOTOPAT<I>                  :(CRGOTOLOOP)       PRE33220
*                                                                                PRE33230
*     ADD BLANKS TO MAKE 80 COLUMNS, ERROR IF MORE THAN 80 COLUMNS               PRE33240
*         ALREADY.                                                               PRE33250
*                                                                                PRE33260
GOTORETSEQ CRGOTO = CRGOTO DUPL(' ',80 - SIZE(CRGOTO))                           PRE33270
+                                          :S(RETURN)F(CRGOTOERR)                PRE33280
UNILABEL   UNILABEL = LABELPAT<1>                                                PRE33290
*                                                                                PRE33300
*****************************************************************************PRE33301
*                                                                                PRE33302
*     ROUTINE TO CREATE A UNIQUE LABEL                                           PRE33310
*                                                                                PRE33320
*     NO PARAMETERS                                                              PRE33330
*                                                                                PRE33340
*****************************************************************************PRE33341
*                                                                                PRE33342
*     COMPUTE NUMBER OF ZEROS TO IMBED AND ADD THEM TO THE LABEL'S               PRE33350
*         UNIFORM BEGINNING                                                      PRE33360
*                                                                                PRE33370
          I = LABELPAT<2> - SIZE(LABELNUM)                                       PRE33380
          LT(I,0)                                  :S(TOOMANYLABELS)             PRE33390
ADDLABZEROS UNILABEL = GT(I,0) UNILABEL '0'        :F(ENDLABZEROS)               PRE33400
          I = I - 1                                :(ADDLABZEROS)                PRE33410
*                                                                                PRE33420
*     ADD THE SIGNIFICANT DIGITS AND THE LABEL ENDING                            PRE33430
*                                                                                PRE33440
ENDLABZEROS UNILABEL = UNILABEL LABELNUM LABELPAT<3>                             PRE33450
*                                                                                PRE33460
*     INCREMENT THE NUMBER USED TO MAKE THE LABEL UNIQUE                         PRE33470
*                                                                                PRE33480
          LABELNUM = LABELNUM + 1                  :(RETURN)                     PRE33490
UNIVAR    UNIVAR = VARPAT<1>                                                     PRE33500
*                                                                                PRE33510
*****************************************************************************PRE33511
*                                                                                PRE33512
*     ROUTINE TO CREATE A UNIQUE VARIABLE                                        PRE33520
*                                                                                PRE33530
*     NO PARAMETERS                                                              PRE33540
*                                                                                PRE33550
*****************************************************************************PRE33551
*                                                                                PRE33552
*     COMPUTE NUMBER OF ZEROS TO IMBED AND ADD THEM TO THE VARIABLE'S            PRE33560
```

```
*          UNIFORM BEGINNING                                                    PRE33570
*                                                                               PRE33580
             I = VARPAT<2> - SIZE(VARNUM)                                       PRE33590
             LT(I,0)                                  :S(TOOMANYVARS)           PRE33600
ADDVARZEROS  UNIVAR = GT(I,0) UNIVAR '0'              :F(ENDVARZEROS)           PRE33610
             I = I - 1                                :(ADDVARZEROS)            PRE33620
*                                                                               PRE33630
*     ADD THE SIGNIFICANT DIGITS AND THE VARIABLE ENDING                        PRE33640
*                                                                               PRE33650
ENDVARZEROS  UNIVAR = UNIVAR VARNUM VARPAT<3>                                   PRE33660
*                                                                               PRE33670
*     ADD THE NEW VARIABLE TO THE LIST FOR LATER DECLARATION                    PRE33680
*          AND INCREMENT THE INDEX INTO THAT LIST AND THE NUMBER                PRE33690
*          USED TO MAKE THE VARIABLE UNIQUE                                     PRE33700
*                                                                               PRE33710
             VARLIST<NEXTVAR> = UNIVAR                                          PRE33720
             NEXTVAR = NEXTVAR + 1                                              PRE33730
             VARNUM = VARNUM + 1                  :(RETURN)                     PRE33740
POP.SEM                                                                         PRE33750
*                                                                               PRE33760
*********************************************************************************PRE33771
*                                                                               PRE33770
*     ROUTINE TO POP ONE TO SIX ITEMS FROM THE TOP OF THE                       PRE33780
*          SEMANTIC STACK                                                       PRE33790
*                                                                               PRE33800
*     TWO TO SEVEN PARAMETERS -- NOARGS, THE NUMBER OF ITEMS TO                 PRE33810
*          BE POPPED; A1 - A6, THE NAMES OF THE VARIABLES (IN STRING            PRE33820
*          FORM) INTO WHICH THE POPPED VALUES ARE TO BE PLACED                  PRE33830
*                                                                               PRE33840
*********************************************************************************PRE33841
*                                                                               PRE33842
             I = 1                                                              PRE33850
*                                                                               PRE33860
*     POP A VALUE FROM THE STACK AND ASSIGN IT TO A PARAMETER                   PRE33870
*                                                                               PRE33880
POP          $($('A' I)) = SEMSTACK<TOP>          :F(EMPTYSTK)                  PRE33890
*                                                                               PRE33900
*     DECREMENT THE POINTER TO THE TOP OF THE STACK                             PRE33910
*                                                                               PRE33920
             TOP = TOP - 1                                                      PRE33930
*                                                                               PRE33940
*     IF NOT DONE POPPING, BRANCH TO POP THE NEXT VALUE                         PRE33950
*                                                                               PRE33960
             I = LT(I,NOARGS) I + 1               :S(POP)                       PRE33970
*                                                                               PRE33980
*     RETURN WITH A NULL STRING VALUE                                           PRE33990
*                                                                               PRE34000
             POP.SEM =                            :(RETURN)                     PRE34010
```

```
PUSH.SEM                                                                PRE34020
*                                                                       PRE34030
*************************************************************************PRE34031
*                                                                       PRE34032
*     ROUTINE TO PUSH ONE TO SIX ITEMS ON THE TOP OF THE                PRE34040
*          SEMANTIC STACK                                               PRE34050
*                                                                       PRE34060
*     TWO TO SEVEN PARAMETERS -- NOARGS, THE NUMBER OF ITEMS TO BE      PRE34070
*          PUSHED; A1 - A6, THE VALUES TO BE PUSHED ONTO THE STACK      PRE34080
*                                                                       PRE34090
*************************************************************************PRE34091
*                                                                       PRE34092
          I = 1                                                         PRE34100
*                                                                       PRE34110
*     INCREMENT THE POINTER TO THE TOP OF THE STACK                     PRE34120
*                                                                       PRE34130
PUSH      TOP = TOP + 1                                                 PRE34140
*                                                                       PRE34150
*     PUSH A VALUE ONTO THE STACK                                       PRE34160
*                                                                       PRE34170
          SEMSTACK<TOP> = $('A' I)                  :F(TOOFULLSTK)      PRE34180
*                                                                       PRE34190
*     IF NOT DONE PUSHING, BRANCH TO PUSH THE NEXT VALUE                PRE34200
*                                                                       PRE34210
          I = LT(I,NOARGS) I + 1                    :S(PUSH)            PRE34220
*                                                                       PRE34230
*     RETURN WITH A NULL STRING VALUE                                   PRE34240
*                                                                       PRE34250
          PUSH.SEM =                                :(RETURN)           PRE34260
END                                                                     PRE34270
```

A STRUCTURED PROGRAMMING PREPROCESSOR FOR A VARIETY OF BASE LANGUAGES

by

Mary L. Love

B.S., Kansas State University, 1975

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

1977

ABSTRACT

This report describes the design and implementation of a preprocessor which is intended to allow and encourage the use of structured programming techniques in a variety of base languages. Eight structured programming control structures are implemented in the preprocessor which currently works with the PL/I, SNOBOL, and FORTRAN programming languages. The extended constructs are translated by the preprocessor into statements which are standard in the base language. The input provided by the user includes information about the base language syntax, optional macro definitions, and a program written in the extended version of the language. The extended version includes the features of the base language, user-defined macros, and the built-in structured programming control structures. The output produced by the preprocessor consists of a printed listing and a temporary disk file. The disk file may be used as input to the usual translator for the base language. The macro facility is a simple text replacement macro processor, included because it is not possible for one person to think of all the control structures which may be desirable at all times. The facility is crude and allows no semantic stack manipulation. A set of guidelines is provided for use in writing macro definitions to reduce the likelihood of writing macros which conflict with the principles of structured programming and, therefore, undermine the purpose of the preprocessor.