SOFTWARE ENGINEERING LABORATORY SERIES SEL-82-002

FORTRAN STATIC SOURCE CODE ANALYZER PROGRAM (SAP) SYSTEM DESCRIPTION

AUGUST 1982

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

N83-13838

FORTRAN STATIC SOURCE CODE ANALYZER PROGRAM (SAP) SYSTEM DESCRIPTION

National Aeronautics and Space Administration
Greenbelt MD

Aug 82

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration, Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

NASA/GSFC (Systems Development and Analysis Branch)
The University of Maryland (Computer Sciences Department)
Computer Sciences Corporation (Flight Systems Operation)

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document. A version of this document was also issued as Computer Sciences Corporation document CSC/SD-82/6045.

Contributors to this document include

William Decker	(Computer Sciences Corporation)
Wayne Taylor	(Computer Sciences Corporation)

Other contributors include

Phil Merwarth	(Goddard Space Flight Center)
Mike O'Neill	(Computer Sciences Corporation)
Charles Goorevich	(Computer Sciences Corporation)
Sharon Waligora	(Computer Sciences Corporation)

Single copies of this document can be obtained by writing to

Frank E. McGarry
Code 582.1
NASA/GSFC
Greenbelt, Maryland 20771

ABSTRACT

This document presents the FORTRAN Static Source Code Analyzer Program (SAP) system description. SAP is a software tool designed to assist Software Engineering Laboratory (SEL) personnel in conducting studies of FORTRAN programs. SAP scans FORTRAN source code and produces reports that present statistics and measures of statements and structures that make up a module. This document presents a description of the processing performed by SAP and of the routines, COMMON blocks, and files used by SAP. The system generation procedure for SAP is also presented.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

SECTION 1 - INTRODUCTION

The FORTRAN Static Source Code Analyzer Program (SAP) automatically produces statistics on occurrences of statements and structures within FORTRAN program modules and provides a facility for reporting the statistics. SAP is available in versions to run on either a PDP-11/70 or a VAX-11/780 computer. This document describes SAP Version 2, a result of program modifications to provide several new reports, additional complexity analysis, and recognition of all statements described in the American National Standards Institute Programming Language FORTRAN standard (FORTRAN 77), ANSI X3.9-1978 (Reference 1).

SAP accepts as input syntactically correct FORTRAN source code written in the FORTRAN 77 standard language. In addition, code written using features in the following languages is also accepted: PDP-11 FORTRAN IV or FORTRAN IV-PLUS (References 2 and 3); VAX-11 FORTRAN (References 4 and 5); IBM S/360 FORTRAN IV Level H Extended, with the exception of the S/360 FORTRAN DEBUG Facility statements (References 6 and 7); and Structured FORTRAN (Reference 8).

Other documents that contain supplementary information are the SAP user's guide (Reference 9) and the SAP design document (Reference 10).

This document describes the SAP software system in detail to assist programmers in maintaining, enhancing, and installing SAP. Section 2 presents an overview of the structure of SAP software and internal tables. Much of the material appearing in this section appeared originally in the SAP design document (Reference 10) and has been updated to reflect the current version of SAP. Section 3 presents descriptions of

each routine in the SAP system. Section 4 presents descriptions of COMMON blocks used by SAP and Section 5 describes each external file referenced by SAP. The instructions for installing SAP on the PDP-11/70 and VAX-11/780 computers are given in Section 6. Section 7 lists areas of concern when moving SAP from one computer to another.

SECTION 2 - SAP STRUCTURE

This section presents an overview of the SAP software structure. The main processing elements are presented in Section 2.1, SAP Processing. The SAP utility software used to support SAP internal data structures is presented in Section 2.2, SAP Utilities. Software that performs housekeeping functions such as report formatting, error processing, and page counting is described only in Section 3, SAP Module Descriptions.

2.1 SAP PROCESSING

This section describes SAP processing, which is divided into the following six phases:

1. Session initialization (Section 2.1.1)
2. Input file loop control and initialization (Section 2.1.2)
3. Source code input (Section 2.1.3)
4. Statement analysis (Section 2.1.4)
5. Module reports and file summary (Section 2.1.5)
6. Project analysis (Section 2.1.6)

The overall program flow is controlled by the main program, SAPMAIN. Figure 2-1 describes the logic of this flow.

The routines called by SAPMAIN are each discussed in the subsections noted in Figure 2-2. Figures 2-3 through 2-7, and Figure 2-9 show portions of Figure 2-2 expanded to greater detail; for reader convenience these figures are contained in the subsections describing the routines shown. The dashed lines in these figures indicate that the routines shown constitute only a portion of SAPMAIN routines.

```
Load keyword table
IF no load error
   DOWHILE no end of file on control input
      Read control input
      IF project analysis is requested
         Initialize SAP data base
      ENDIF
      Initialize global counters
      DOWHILE no end of file on source input
         Initialize module counters
         DOWHILE no END statement
            Read a statement
            Process a statement
         END DOWHILE
         Output module statistics
         IF Halstead summary requested
            Report Halstead summary
         ENDIF
         Collect global statistics
         Output module directory entry
      END DOWHILE
      IF global statistics requested
         Output global statistics
      ENDIF
      Close source input files
   END DOWHILE
   IF project analysis is requested
      Report on each project requested by the user
   ENDIF
ENDIF
Terminate SAP
```

Figure 2-1. Processing Flow for SAPMAIN

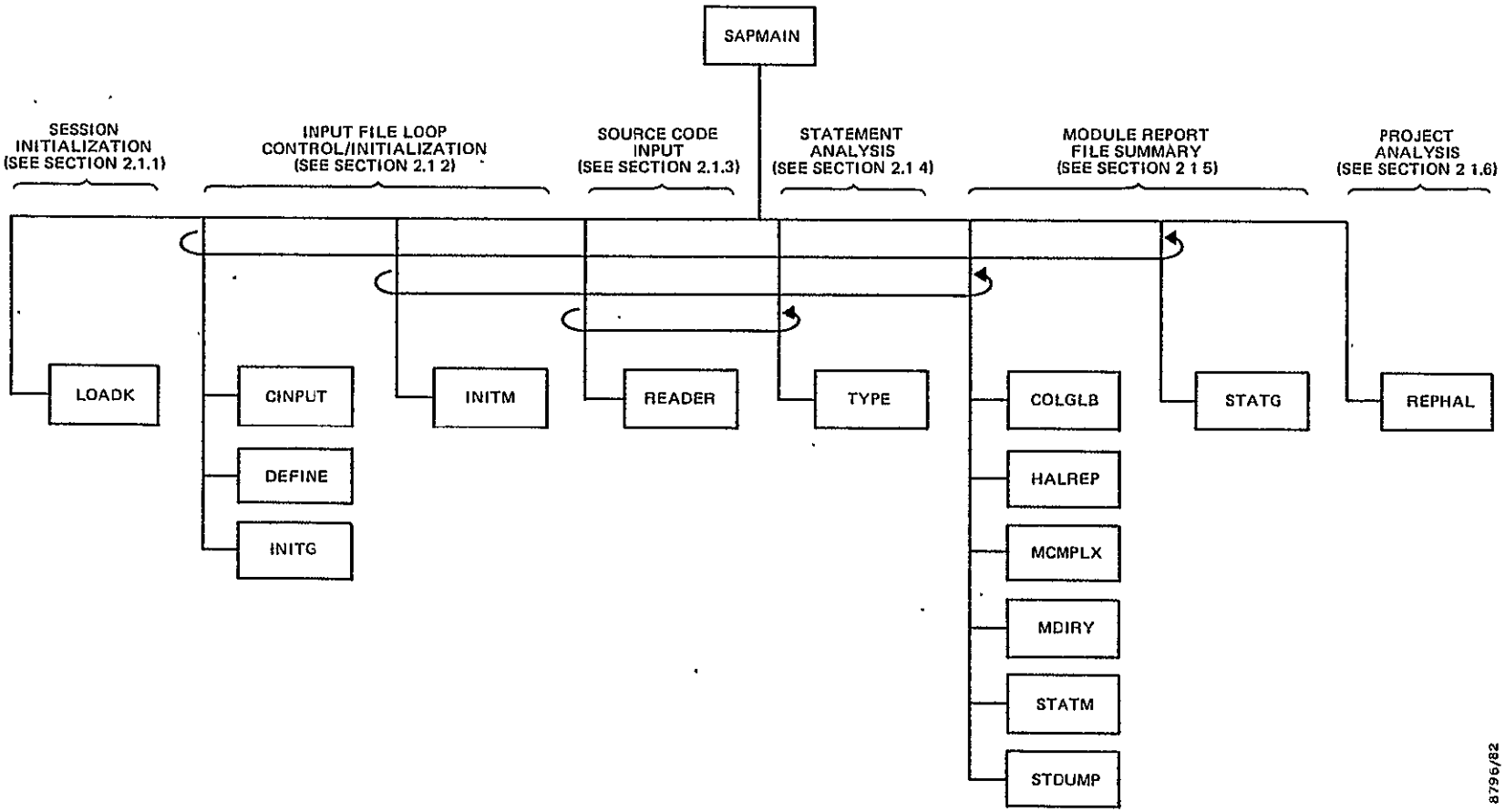


Figure 2-2. Routines Called by SAPMAIN

2.1.1 SESSION INITIALIZATION

SAP uses two table initialization subroutines at the start of each SAP session: LOADK and USRWTS (see Figure 2-3). LOADK loads keywords from the KEYWORDS.SAP file into the keyword table stored in COMMON block KEYCOM; USRWTS loads the WEIGHTS.SAP file into the weights table stored in COMMON block WTSCOM. A user-specified statistical weighting file can be read in Reference 9.

2.1.1.1 LOADK

LOADK (load keyword table) opens the keyword file and loads the keyword table (in COMMON block KEYCOM). An error flag is set to .TRUE. if an open failure or read error occurs.

2.1.1.2 USRWTS

USRWTS (load weights table) opens the default weights file and loads the weights table (in COMMON block WTSCOM). An error flag is set to .TRUE. if an open failure or read error occurs.

2.1.2 FILE LOOP CONTROL AND INITIALIZATION

The processing loop for each input file to SAP is controlled and initialized by the routines shown in Figure 2-4. The routines called by SAPMAIN are discussed below.

2.1.2.1 CINPUT

Control for SAP file loop processing is handled by subroutine CINPUT (read control input). CINPUT calls subroutine INPUT, which reads the user input command line; CINPUT then opens the source input file and interprets the switch settings.

INPUT prompts the user with SAP> and reads one line of control input information from logical unit LUNCIN. Input line syntax is as follows:

```
SAP>FILE.EXT/S1/S2/-S3
```

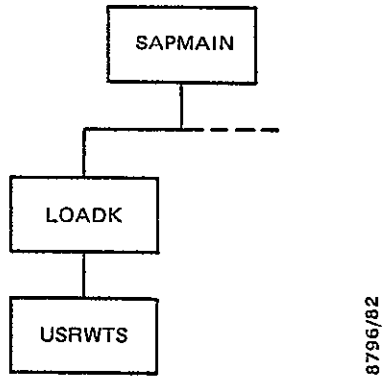


Figure 2-3. Session Initialization Routines

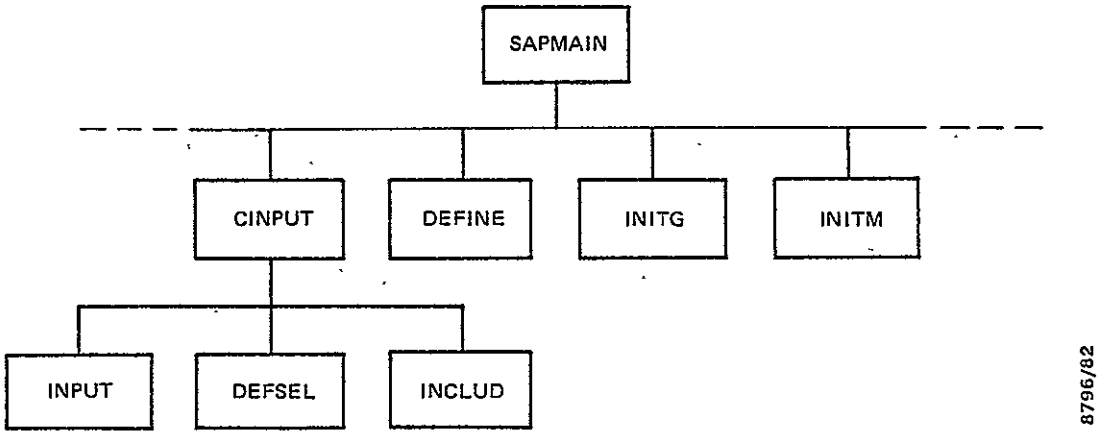


Figure 2-4. File Loop Control and Initialization Routines

where FILE.EXT is the file name and extension of the input source file to be processed and S1, S2, and S3 are option switches (Reference 9).

CINPUT scans the input line for slashes (/), which are assumed to be switch delimiters. The slashes are replaced by zeros and a check is made for minus signs (-) and any switches that are found are set to .TRUE. (or .FALSE. if preceded by a minus sign).

CINPUT calls two routines, DEFSEL and INCLUD, to handle the /SL and /XP control switches, respectively. DEFSEL opens the sequential output file used to communicate with the SEL data base. If an ALL.SAP file exists in the user's default directory, the file is opened with the APPEND option; otherwise it is opened as NEW. INCLUD copies the input file into a scratch file while examining each record looking for an INCLUDE statement. Each INCLUDE statement is replaced with the contents of the included file.

CINPUT opens the input file (or the scratch file created by INCLUD) and returns control to SAPMAIN.

If the user enters an end-of-file (CNTL Z) in response to the request for an input file name, the ENDC flag is set to .TRUE. and CINPUT returns.

2.1.2.2 DEFINE

DEFINE (define SAP data base) initializes or locates a data base file when the /DB control switch is set to on. The user is prompted for the file name to be used as the data base for this session. If the file does not exist, DEFINE opens the specified file as NEW and initializes as many records as specified by the user. If the file does exist, the file is opened. The user is prompted for a project character to be used for identification of the group of modules to be processed.

2.1.2.3 INITG

INITG (initialize globals) resets all variables and arrays used to accumulate statistics describing all the modules within an input file. This routine is called once for each input file.

2.1.2.4 INITM

INITM (initialize module) resets all variables and arrays used to accumulate statistics describing each module within the input file. This routine is called before each module in an input file is processed.

2.1.3 SOURCE CODE INPUT

Each statement from the input file is read as one or more records by routine READER; the routines it calls are shown in Figure 2-5 and discussed below.

2.1.3.1 READER

READER (source code reader) controls all source input; accumulates counts of input lines, statements, comments and comment packets; and generates the packed statement string. READER calls GLINE and HSCAN in performing these functions.

2.1.3.2 GLINE

GLINE (get one line) reads input from the source file one line at a time. GLINE maintains a one-line-look-ahead, calling TABCCC to detect comment and continuation lines. Output is via COMMON block INLCOM.

2.1.3.3 TABCCC

TABCCC (check for tabs, continuations, and comments) checks for the presence of a tab character in any of positions 1 through 6 in the input line. If one is found, it tests the next character to see if it is a nonzero numeric. If it is, the continuation flag is set to .TRUE. and the character is replaced with a blank. If no tab is found, position six is

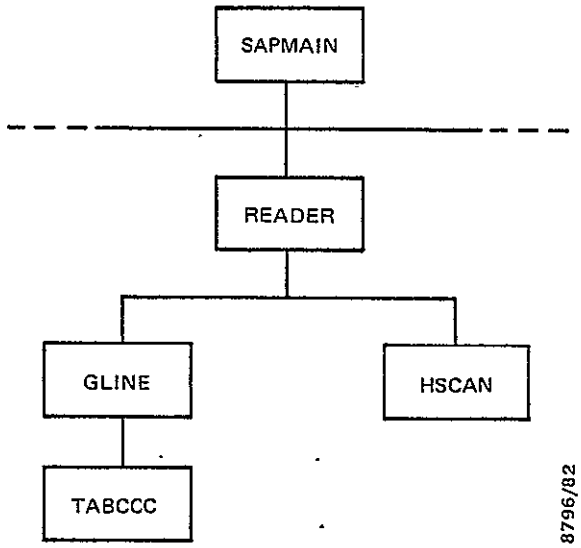


Figure 2-5. Source Code Input Routines

checked for a nonzero, nonblank character. If one is found, the continuation flag is set to .TRUE. and a tab is set in position six. In both cases, the first position is checked for a nonnumeric character. If one is present, the comment card flag is set to .TRUE.. Finally, all tabs following the initial tab are replaced by blanks and TABCCC returns.

2.1.3.4 HSCAN

HSCAN (remove literals, holleriths, and blanks) processes the source code input line which was loaded in array INPUT (in COMMON block INPCOM) by READER and produces a packed source string in the same array. HSCAN first scans INPUT for single quotation marks (apostrophes), removing any character occurring between matched pairs of quotation marks. HSCAN then scans the string for the character "H". Wherever it appears, the previous characters are tested for numerics. If the characters are numeric and are preceded by any of the following characters (/,*'), the field following the H is considered to be a Hollerith field. The numerics are converted to integer and the value tested to make sure that the end of the H field is within the line. Then, the field is replaced by two quotation marks and removed. HSCAN next scans for an exclamation point (!), the PDP-11 delimiter for inline comments. If one is found, the inline comment counter is incremented and the exclamation point is replaced by a null character. LASINP, the end of line pointer, is reset to point to the null character.

Finally, HSCAN removes all embedded blanks remaining and returns.

2.1.4 STATEMENT ANALYSIS (SUBROUTINE TYPE)

SAP processes each input file on a module-by-module basis. The module statistical counters are initialized by routine INITM at the beginning of a module. Module statistics are accumulated until an END card is encountered within the input code.

Statement analysis is controlled by subroutine TYPE, called from SAPMAIN. Figure 2-6 shows the statement analysis portion of SAP. Analysis falls into the following three phases that are discussed separately in Sections 2.1.4.1 through 2.1.4.3:

1. Construction of the delimiter/token table and statement classification
2. Statement specific analysis
3. Statement label processing

DSCAN controls the delimiter/token table building. ASGNID and TESTK identify and classify the statement. STATE controls statement type specific analysis, and LABEL processes statement labels. Section 2.1.4.4 discusses two utilities frequently used while performing statement analysis.

TYPE (control statement analysis) initially calls DSCAN to build the delimiter/token table. TYPE then examines the table produced to locate a tab symbol. The table pointer LDTPTR is set to point to the table location following the tab. If the pointer is not pointing to the end of the table, processing proceeds. If it is, processing of the statement terminates. Next, ASGNID is called to identify assignment statements, followed by TESTK to identify statements with leading keywords. TESTK returns the statement class (specification, control, etc.) and the statement type (IF, DO, etc.). TYPE calls STATE to process the statement

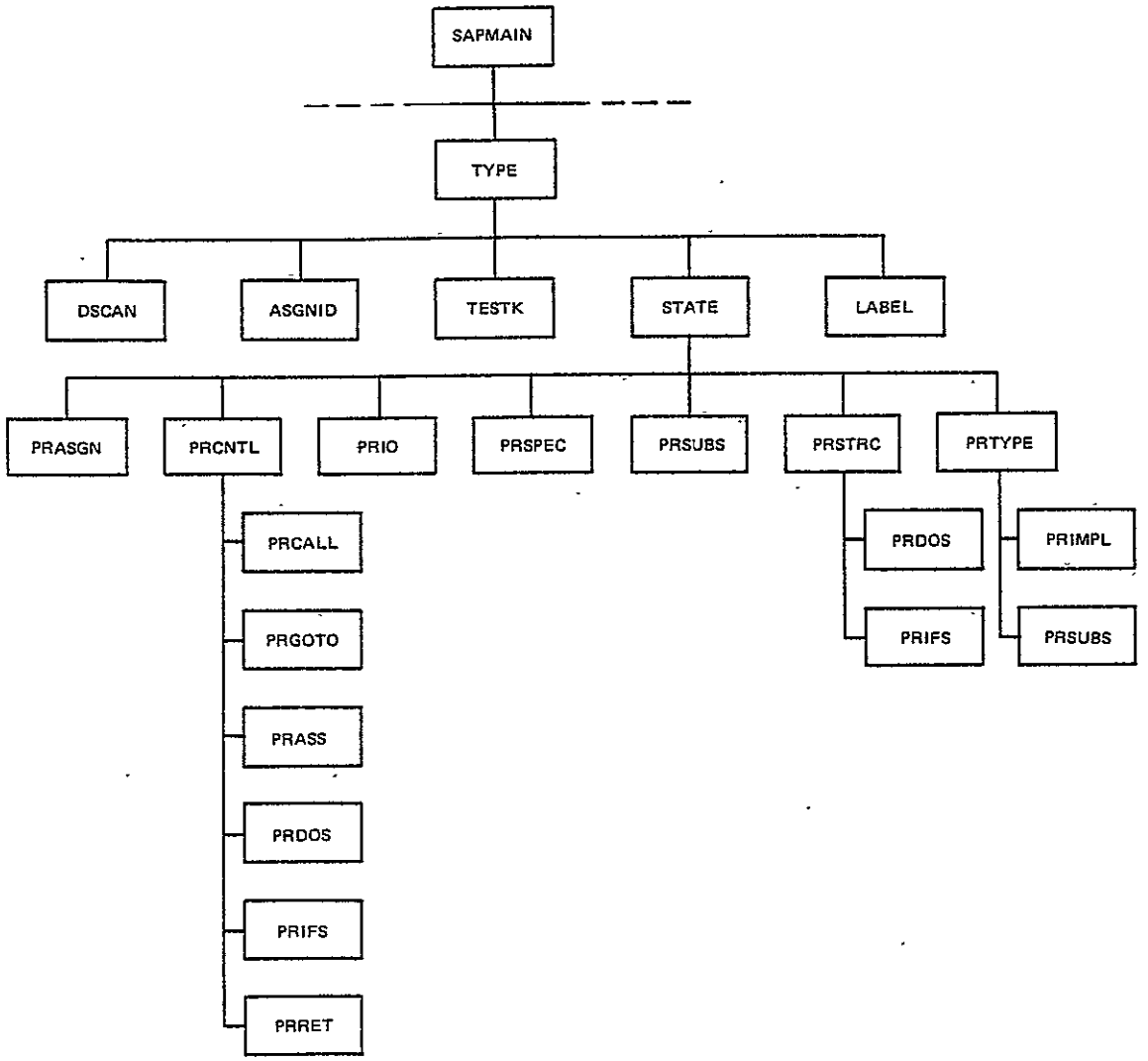


Figure 2-6. Statement Analysis Routines

and then increments the appropriate class and type counters. LABEL is called to store the statement label (if present) in the statement label list and to gather DO loop statistics. TYPE checks the statement type to see if an end statement has been reached. If it has, ENDM is set to .TRUE. to indicate an end of module. Any fatal error return from a subroutine called by TYPE causes ERROR to be set to .TRUE..

2.1.4.1 Statement Identification

2.1.4.1.1 DSCAN

DSCAN (scan for delimiters and tokens) processes the packed input string in array INPUT prepared by HSCAN. DSCAN searches the input string for a delimiter (as defined in the delimiter table IDELIM), comparing one character at a time with the first character of each defined delimiter. When a first character match is found, the remainder of the delimiter is compared with the subsequent characters in the INPUT string. If a match is found, DSCAN then checks to see if any nondelimiter characters exist between the current delimiter and the previous delimiter. These characters (if any) are then hashed by IHASH and LOOKS is called to see if a symbol table entry for the token already exists. (Section 2.2.1 contains a complete description of the symbol table utilities, including the hash algorithm used.) If it does not exist, a new symbol table entry is created and entered into the symbol table by calling POKES. The symbol table pointer IPOINT is entered into the delimiter/token table LISTDT, in the next available location.

In creating the symbol table entry, the first character of the token is tested to determine whether the token is an identifier or a constant (numeric or logical). The new token pointer is then entered into LISTDT. The next location in LISTDT is given the value of the negative of the

index of the located delimiter. The LISTDT array thus contains a series of negative and positive numbers, where negative numbers represent delimiters and positive numbers are pointers to the symbol table entries for the intervening tokens. The scanning process proceeds until the scan pointer is pointing at LASINP, the last location in INPUT. DSCAN sets ERROR to .TRUE. if the LISTDT array limits are exceeded or if LOOKS or POKES returns a fatal error.

2.1.4.1.2 ASGNID

ASGNID (assignment statement identification) scans the LISTDT array to identify assignment statements. Statement function definitions identified as assignment statements in ASGNID will be detected and reclassified in routine PRASGN. The following conditions will lead to a classification as an assignment statement:

- The LISTDT array contains an equals sign not enclosed in parentheses.
- All commas following the equals sign are enclosed in parentheses.
- The first token in LISTDT does not start with the keyword PARAMETER.
- The first token in LISTDT does not start with the keyword IF, which is then followed by a pair of matching parentheses, which are followed by a token. (That is, a logical IF statement whose object is an assignment statement is classified as an IF statement at this point.)

Keyword statement classification is done by TESTK and LOOKK. In FORTRAN, most statement types are preceded by a keyword.

2.1.4.1.3 TESTK

TESTK (test for a leading keyword) examines the token pointed to by LDTPTR in the array LISTDT. TESTK calls LOOKK to test this symbol against the keyword list. LOOKK returns the keyword located and its length. If no keyword is located, TESTK sets ISCLAS to 12 (undecoded) and returns. If a keyword shorter than the test token is found, then the initial token is rehashed after the keyword portion is deleted. LDTPTR, the LISTDT pointer, is returned pointing to the location following the keyword.

2.1.4.2 Statement Specific Processing (Subroutine STATE)

STATE (statement type specific analysis) is an executive driver to routines that perform specific statement analysis. Routines that fall into this classification use PR as the first two letters of their name. These routines are described in Sections 2.1.4.2.1 through 2.1.4.2.14. Before calling STATE, the specific statement class and type have been determined. It is the function of routines called by STATE to gather those statistics that are both class and type dependent. Specific classes examined in STATE are assignment statements (PRASGN), control statements (PRCNTL), subprogram statements (PRSUBS), specification statements (PRSPEC), type specification statements (PRTYPE), input/output statements (PRIO), and special structure statements (PRSTRC).

2.1.4.2.1 PRASGN

PRASGN (assignment statement analyzer) is entered when subroutine ASGNID has detected statements of the following form:

v = e

where v = variable name or array element name
e = expression

The analyzer performs a scan of statement tokens to advance program counters and update the status of items in the symbol table. Specifically, PRASGN performs the following functions:

- Counts the number of variables in assignment statements.
- Determines the maximum number of variables in any given assignment statement in the module.
- Counts the number of operators in assignment statements. (Operators are defined as follows: **, *, /, +, -, .AND., .OR., .XOR., .EQV., .NEQV., .NOT., .LE., .LT., .EQ., .NE., .GT., .GE. Operators used in describing array variables or in function arguments are not counted.)
- Detects and flags Arithmetic Statement Function (ASF) definitions.
- Performs analysis on any variables encountered.
- Performs analysis on any function or ASF encountered.
- Marks in the symbol table that specific variables or functions were encountered.

2.1.4.2.2 PRCNTL

PRCNTL (control statement analyzer) accepts statement tokens that have the following FORTRAN keywords: ASSIGN, CALL, CONTINUE, DO, GOTO, IF, PAUSE, RETURN, and STOP. PRCNTL acts as an executive to specific control statement analyzer routines, which scan each type of statement to advance program counters and update the status of items in the symbol table. Specifically, PRCNTL performs the following functions:

- Switches control to the specific statement analyzer routines:
  - PRASS--Keyword ASSIGN
  - PRCALL--Keyword CALL
  - PRDOS--Keyword DO
  - PRGOTO--Keyword GOTO
  - PRIFS--Keyword IF
  - PRRET--Keyword RETURN
- Returns control to subroutine STATE when the specific routines are completed or when keywords CONTINUE, PAUSE, or STOP are encountered

2.1.4.2.3 PRASS

PRASS (ASSIGN statement analyzer) analyzes ASSIGN statements. The ASSIGN statement is used to associate a statement label with an integer variable. The variable can then be used as a transfer destination in a subsequent assigned GOTO statement.

The ASSIGN statement has the form

```
ASSIGN s TO i
```

where s = label of an executable statement
i = integer variable

PRASS will scan for the statement label and add it to the statement label list if it is not in the list.

2.1.4.2.4 PRCALL

PRCALL (CALL statement analyzer) analyzes CALL statements. The CALL statement causes the execution of a SUBROUTINE subprogram; it can also specify an argument list for use by the subroutine.

The CALL statement has the form

```
CALL sub [([a [, a] ....])]
```

where sub = name of a SUBROUTINE subprogram

a = argument to a subprogram. Arguments can be variables, arrays, array elements, constants, expressions, alphanumeric literals, subprogram names, or alternate return label specifiers

The analyzer performs a scan of statement tokens to advance program counters and update the status of items in the symbol table.

Specifically, PRCALL performs the following functions:

- Counts the number of arguments in all CALL statements encountered
- Determines the maximum number of arguments in any CALL statement
- Adds the subroutine name and alternate return labels to the alternate return transfer table list when an alternate return is located
- Performs analysis on any functions or ASFs encountered
- Marks in the symbol table that variables or functions were encountered
- Marks in the symbol table that a subroutine name was encountered

2.1.4.2.5 PRDOS

PRDOS (DO statement analyzer) analyzes loop control statements. The DO and DOWHILE statements are used to specify discrete loop processing. The DO statement causes the statements in its range to be repeatedly executed a specified number of times. DOWHILE statements are used to specify conditional loop processing.

The DO statement has the form

$$\text{DO } [s[,]] \; i = e_1, e_2 \; [,e_3]$$

where s = label of an executable statement
i = integer variable (control variable)
e_1, e_2, e_3 = integer expressions

The DOWHILE statement has the form

$$\text{DO } [s[,]] \text{ WHILE } (e)$$

where s = label of an executable statement
e = logical expression

The analyzer performs a scan of statement tokens to push target labels onto the DO loop stack and update the status of items in the symbol table.

Specifically, PRDOS performs the following functions:

- Completes the identification of DOWHILE statements when a statement label is present.
- Completes the identification of a DO statement when no statement label is present.

- Pushes the target label (if present) and the current statement number onto the stacks LBLSTK and DOSTAN (in COMMON block LBLCOM). A value of zero for the statement label is pushed when no label is present.
- Performs analysis on the control variable and other variables encountered in the expressions.

2.1.4.2.6 PRGOTO

PRGOTO (GOTO statement analyzer) analyzes all types of GOTO statements. A GOTO statement transfers control within a program unit, either to the same statement every time or to one of a set of statements, based on the value of an expression. There are three types of GOTO statements:

- Unconditional GOTO statement (GOTO s) where s is the label on an executable statement
- Computed GOTO statement (GOTO (slist)[,]e) where slist is a list of one or more executable statement labels separated by commas and e is an arithmetic expression
- Assigned GOTO statement (GOTO v[[,](slist]) where slist (when present) is a list of one or more executable statement labels separated by commas and v is an integer variable

The analyzer performs a scan of statement tokens to advance program counters and update the status of items in the symbol table. Specifically, PRGOTO performs the following functions:

- Identifies the specific type of GOTO and maintains counters on the number of unconditional, computed, and assigned GOTO statements encountered

- Marks in COMMON block LBLCOM whether or not a label is a target of an unconditional GOTO
- Adds the statement label (or statement label list) to the unconditional, computed, or assigned GOTO transfer table list

2.1.4.2.7 PRIFS

PRIF (IF statement analyzer) analyzes IF statements. The IF statement causes a conditional control transfer or the conditional execution of a single statement or block of statements. There are four types of IF statements:

- Arithmetic IF statement (IF (e) s_1, s_2, s_3) where e is an arithmetic expression and s_1, s_2, and s_3 are labels of executable statements
- Logical IF statement (IF (e) st) where e is a logical expression and st is a complete FORTRAN statement
- Block IF statement (IF (e) THEN) or (.IF (e)) where e is a logical expression
- ELSEIF statement (ELSEIF (e) THEN) where e is a logical expression

The analyzer performs a scan of statement tokens to advance program counters and update the status of items in the symbol table. Specifically, PRIFS performs the following functions:

- Maintains counters on the number of ELSEIF and logical, arithmetic, and block IF statements
- Performs analysis on any statement labels encountered
- Performs analysis on any variables or arrays encountered

- Performs analysis on any functions or ASFs encountered
- Marks in the symbol table that a variable or function was encountered
- Sets IREPT = .TRUE. for logical IF statements and sets LDTPTR to point to the beginning of an object statement

2.1.4.2.8 PRRET

PRRET (RETURN statement analyzer) analyzes RETURN statements. The RETURN statement is used to return control from a subprogram unit to the calling program unit.

The RETURN statement has the form

RETURN [e]

where e is an integer expression indicating an alternate return.

The analyzer performs a scan of statement tokens to advance program counters. Specifically, PRRET performs the following functions:

- Maintains a counter on the number of normal returns encountered
- Maintains a counter on the number of alternate returns encountered

2.1.4.2.9 PRSUBS

PRSUBS (subprogram statement analyzer) accepts statement tokens that have the following FORTRAN keywords: BLOCKDATA, END, ENTRY, FUNCTION, PROGRAM, and SUBROUTINE. The analyzer performs a scan of the statement tokens to advance the program counters and update the status of items in the symbol

table. Specifically, PRSUBS performs the following functions:

- Determines the module type
- Saves the module name in array MODNAM in COMMON block MODCOM
- Flags ENTRY names in the symbol table
- Counts and flags argument list names passed to a module

2.1.4.2.10 PRSPEC

PRSPEC (specification statement analyzer) accepts statement tokens that have the following FORTRAN keywords: COMMON, DIMENSION, EQUIVALENCE, EXTERNAL, INTRINSIC, PARAMETER, SAVE, and VIRTUAL. The analyzer performs a scan of the statement tokens to advance program counters and update the status of items in the symbol table. Specifically, PRSPEC performs the following functions:

- Flags COMMON block names
- Flags EXTERNAL variable names
- Flags COMMON block variable names
- Flags variable names in DIMENSION and VIRTUAL statements as arrays
- Counts number of dimensions per array
- Flags equivalenced variable names

No processing is performed on INTRINSIC, PARAMETER, or SAVE statements.

2.1.4.2.11 PRTYPE

PRTYPE (type specification statement analyzer) accepts statement tokens having the following FORTRAN keywords: BYTE, CHARACTER, COMPLEX, DOUBLECOMPLEX, DOUBLEPRECISION,

IMPLICIT, INTEGER, LOGICAL, and REAL. The analyzer performs a scan of the statement tokens to advance program counters and update the status of items in the symbol table. Specifically, PRTYPE performs the following functions:

- Flags dimensioned arrays
- Counts the number of dimensions per array
- Deconcatenates the length specifier (if any) from the first variable name token
- Checks for the FUNCTION keyword and reclassifies the statement if it is found
- Calls PRIMPL to process an IMPLICIT statement
- Calls PRSUBS to process a typed FUNCTION

2.1.4.2.12 PRIO

PRIO (input/output statement analyzer) accepts statement tokens that have the following FORTRAN keywords: ACCEPT, BACKSPACE, CLOSE, DECODE, DEFINEFILE, DELETE, ENCODE, ENDFILE, FIND, INQUIRE, OPEN, PRINT, READ, REWIND, REWRITE, TYPE, WRITE, and UNLOCK. The analyzer performs a scan of the statement tokens to advance program counters and update the status of items in the symbol table. Specifically, PRIO performs the following functions:

- Counts the number of statements that use ERR =
- Counts the number of statements that use END =
- Performs analysis on any variables encountered in an input/output list
- Performs analysis on any functions or ASFs encountered in the input/output list
- Marks in the symbol table that a variable or function was encountered in the input/output list

- Performs analysis on any label encountered after an END = or ERR =
- Adds the statement label to the END = or ERR = transfer table list

2.1.4.2.13 PRIMPL

PRIMPL (implicit statement analyzer) accepts statement tokens following the IMPLICIT statement in groups beginning with the following FORTRAN keywords: BYTE, COMPLEX, CHARACTER, INTEGER, LOGICAL, DOUBLEPRECISION, DOUBLECOMPLEX, and REAL. The analyzer performs a scan of the statement tokens to advance program counters and update the status of items in the symbol table. Specifically, PRIMPL performs the following functions for each group of tokens:

- Determines the keyword type
- Deconcatenates the length specifier (if any) from the keyword type
- Stores (in COMMON block IMPCOM) the default type to be assigned to untyped variables whose name starts with the specified letters

2.1.4.2.14 PRSTRC

PRSTRC (structured construct analyzer) accepts statement tokens that have the following FORTRAN keywords: DOWHILE, ELSEIF, ELSE, ENDDO, ENDIF, .IF, and THEN. The analyzer performs a scan of the statement tokens to advance program counters and update the status of items in the symbol table. Specifically, PRSTRC performs the following functions:

- Calls PRDOS to process DOWHILE statements
- Calls PRIFS to process .IF and ELSEIF statements
- Pops the DO loop target STACK if an ENDDO statement has no label

- Adjusts the IF block nesting level if the statement is an ENDIF

2.1.4.3 Statement Label Processing

SAP statistical processing requires analysis of statement labels; this analysis falls into two categories:

1. Processing of target labels encountered in ASSIGN, DO, DOWHILE, GOTO and IF statements
2. Gathering DO loop statistics at the time the loop target statement is processed

The first function is performed by the statement processors described in the previous sections; the second is performed by subroutine LABEL. Both LABEL and the statement processors utilize two label processing utilities, LABLST and INTGR4.

2.1.4.3.1 LABEL

LABEL (process DO loop target label) is called by TYPE for all non-FORMAT statements. LABEL tests the first token in the LISTDT array for the presence of a tab. If a tab is found, no label is present and LABEL returns. If no tab is found, a statement label is present and LABEL calls LOOKP to fetch the token, and then calls INTGR4 to convert it to INTEGER*4 format. LABEL then calls LABLST to add the label to the label list array LBLIST in COMMON block LBLCOM. LABLST returns the location of the label in LBLIST. If the label is the target of a DO loop, its integer representation will have been previously pushed onto the DO loop target stack LBLSTK (in COMMON block LBLCOM). LABEL tests this stack and pops it if a match is found. If the label is a target, the DO loop length counter and depth of nesting counter are updated and LABEL returns.

2.1.4.3.2 LABLST

LABLST (add a label to the label list) searches the LBLIST array (in COMMON block LBLCOM) for a match to the input label. If a match is found, a pointer is set to the entry in LBLIST, and LABLST returns. If no match is found, LABLST adds the label to the end of LBLIST and returns with the pointer indicating the new entry. If no space remains in LBLIST, the error flag is set to .TRUE..

2.1.4.3.3 INTGR4

INTGR4 (convert a token to INTEGER*4 representation) converts the ASCII input array into an integer and returns it. INTGR4 utilizes DECODE and is limited to five decimal digits (the maximum label size). Any illegal decimal conversion will result in a syntax error message and a returned value of zero.

2.1.4.4 Token Processing Utilities

SAP processing requires that several standard counts and calculations be applied to each token encountered while parsing a statement. These standard operations are performed by routine FLVARI for specification and declaration statements and by routine PRTOKE for executable statements. These routines are discussed below.

2.1.4.4.1 FLVARI

FLVARI (flag variables) identifies arrays and sets flag bits in the symbol table.

Processing includes the following:

- Counting the number of dimensions within parentheses following the token (if any)
- Classifying the token as a variable or array depending upon the presence of parentheses following the token

- Combining the token type with a bit mask using the OR function

2.1.4.4.2 PRTOKE

PRTOKE (process token) identifies and processes a token as a constant, variable, or function.

Processing includes

- Determining subscript complexity (level of parentheses and operators)
- Classifying the token as a function or ASF, constant, variable, or variable array and as either CHARACTER or numeric
- Counting the number of arguments to a function or ASF

The item is processed until a balancing of parentheses occurs.

2.1.5 MODULE REPORTS AND FILE SUMMARY

The results of each module loop and input file loop are gathered and reported by the routines shown in Figure 2-7. Each routine called by SAPMAIN in this phase is discussed below.

2.1.5.1 STATM

STATM (module statistics report) produces a report of the statistics for each module in an input file. STATM calls subroutine TABLES to accumulate token use statistics from a scan of the entire symbol table and to count statement label use from a scan of the statement label list.

STATM produces each paragraph of the module statistics report (except the complexity paragraph) based upon the current settings of the control switches.

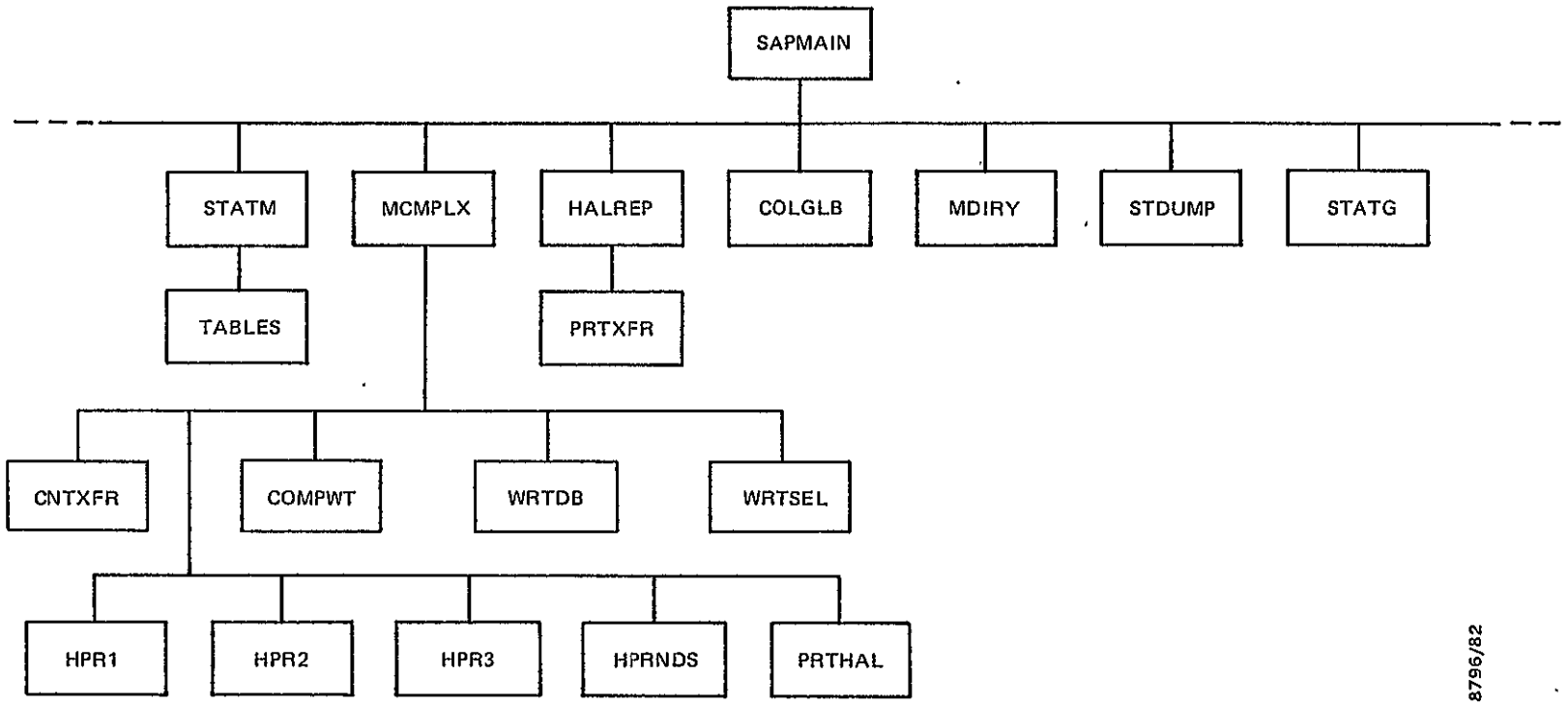


Figure 2-7. Module Report and File Summary Routines

2.1.5.2 MCMPLX

MCMPLX (module complexity) controls the calculation and presentation of the source code complexity measures. Subroutine COMPWT is called to calculate the SEL complexity using the weights table. Subroutines HPR1, HPR2, HPR3, and CNTXFR are called to count the delimiter, keyword, procedure, and transfer Halstead operators, respectively. Subroutine HPRNDS counts the Halstead operands. After the measures have been calculated, MCMPLX produces the complexity paragraph of the module statistics report.

Subroutines WRTDB and WRTSEL are called to write to external SAP files if the respective /DB and /SL control switches are set.

2.1.5.3 HALREP

HALREP (Halstead report) is called from SAPMAIN when the /HL control switch is set. HALREP produces a report showing all Halstead operators and operands detected in a module and their use counts. Subroutine PRTXFR is called to produce the paragraph that reports on the Halstead transfer operators.

2.1.5.4 COLGLB

COLGLB (collect global statistics) adds the module statistic accumulators to the input file accumulators. The global maxima variables are adjusted when exceeded by the module maxima variables. COLGLB is called after each module is processed.

2.1.5.5 MDIRY

MDIRY (module directory report) is called from SAPMAIN to write a module entry in the module directory. The module directory always appears as part of SAP output and is not influenced by any of the listing control switches.

2.1.5.6 Symbol Table Dump (Subroutine STDUMP)

Figure 2-8 is an example of a symbol table dump produced when the control switch /DU is set. In the example given, there are 6000 words in the symbol table, of which 441 were used. Only a representative sample of the symbol table dump is shown in Figure 2-8. Each hash table entry pointing to a chain of symbol table entries is shown. A description of each symbol table entry in the chain follows the description of the hash table entry. The explanation of the items in a symbol table entry is as follows:

Item	Meaning
NEXT	Location in the symbol table of next entry in the linked list. If this is the last entry in this list, the value will be zero
LAST	Location in the symbol table of previous entry in the linked list. If this is the first entry in this list, the value will be zero
NACTIV	Halstead operand activity counter. Incremented each time this entry was accessed while parsing an executable statement
ICLASS	Binary value, indicating the class of the token: = -2, Arithmetic Statement Function (ASF) = -1, Function = 0, Undefined (initially set to this) = 1, Constant = 2, Variable (further defined by ITYPE) = 3, Array (further defined by ITYPE) = 4, Other name (further defined by ITYPE)
ITYPE	Token type defined when ICLASS = 2, 3, or 4. The interpretation of ITYPE is as follows: If ICLASS = 2 or 3, ITYPE should be interpreted as a bit string with the following attributes assigned to the token if the indicated bit is set. (Bits are numbered from zero starting with the least significant bit)

```
DOTEST.FOR/DU
SYMBOL TABLE DUMP, MAXSYM =  6000 NEXSYM =   441
SECOND RECORD = NEXT, LAST, NACTIV, ICLASS, ITYPE, IUSED, LTOKE, TOKEN

AT HASH LOCATION   77, IPOINT =   220
AT    220 RECORD =     0    0    1    0    0    0    1 L

AT HASH LOCATION   85, IPOINT =    44
AT     44 RECORD =     0    0    1    3    8    1    1 T

AT HASH LOCATION   97, IPOINT =   141
AT    141 RECORD =     0    0    1    1    0    1    2 2.

AT HASH LOCATION   98, IPOINT =    36
AT     36 RECORD =     0    0    1    1    0    0    2 10


AT HASH LOCATION  216, IPOINT =   432
AT    432 RECORD =     0    0    0    0    0    0    3 END

AT HASH LOCATION  241, IPOINT =   394
AT    394 RECORD =     0    0    1    2    8    1    3 ITS

AT HASH LOCATION  298, IPOINT =   385
AT    385 RECORD =   413    0    1    2    8    1    4 MINE
AT    413 RECORD =     0  385    0    0    0    0    4 ELSE
```

Figure 2-8. Sample Symbol Table Dump

Item	Meaning
ITYPE (Cont'd)	Bit 0 set, argument to module Bit 1 set, equivalenced Bit 2 set, appears in COMMON Bit 3 set, numeric variable or array Bit 4 set, CHARACTER variable or array If ICLASS = 4, ITYPE should be interpreted as a binary value with the following meanings: = 1, Module name = 2, ENTRY name = 3, EXTERNAL name = 4, COMMON block name = 5, NAMELIST name = 6, Externally defined subroutine or function
IUSED	Symbol utilization count. Incremented each time token is used in an executable statement
LTOKE	Length of token in characters
TOKEN	Token

2.1.5.7 STATG

STATG (global statistics report) produces a report of the statistics for each input file. The global accumulators and global maxima are used in preparing this report. STATG is called from SAPMAIN when the /GB control switch is set.

2.1.6 PROJECT ANALYSIS

The SAP project analysis phase produces an optional summary report of data stored in a SAP data base file. The project analysis is controlled by subroutine REPHAL, which is called by SAPMAIN as shown in Figure 2-9.

REPHAL searches the data base to locate each record with a project character that matches the requested project. The data on each located record is passed to routine ESTIM, where the derived Halstead quantities (References 9 and 11) are calculated. REPHAL reports the data from the data base and the Halstead quantities in the project summary report.

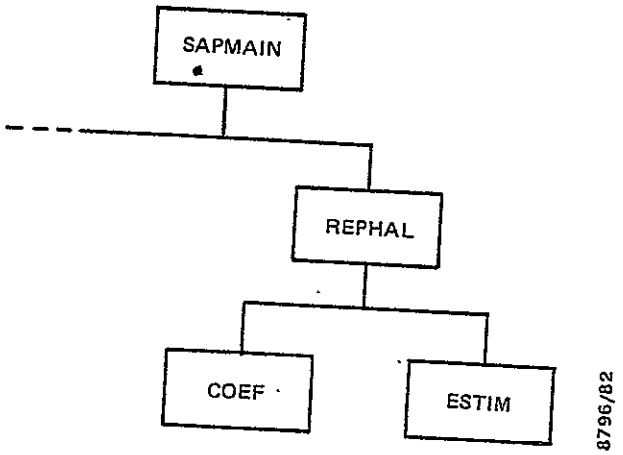


Figure 2-9. Project Analysis Routines

After the project summary report is produced, routine COEF calculates and reports the correlation coefficient matrix for the requested project.

2.2 SAP UTILITIES

SAP processing is based upon the use of three internal data tables: the symbol table, the delimiter/token table, and the transfer table. The following subsections discuss each table and the utility routines used to maintain them.

2.2.1 SYMBOL TABLE UTILITIES

A central feature of the SAP design is the symbol table. The SAP symbol table, which is stored in COMMON block SYMCOM, is a hash-keyed linked list that is used to store all nondelimiter symbols identified in the statement scan. A set of utility routines allows access to this table via the single table entry COMMON block STECOM. Subroutines LOOKS and LOOKP allow read access to the table; POKES and POKEP allow write access (see Sections 2.2.1.1 through 2.2.1.4).

Deletion of a symbol table entry is accomplished by KILLP (Section 2.2.1.6), which relinks around a designated symbol and flags it for deletion. Compression of deleted symbols is done whenever there is insufficient space to add a new symbol; a "garbage collection" subroutine, GARCOL (Section 2.2.1.7), compresses and relinks the table. If still more symbol table space is needed, the table is structured to allow easy implementation of a paging algorithm.

The SYMCOM and STECOM COMMON blocks are described in detail in Section 4. COMMON block SYMCOM may be thought of as a file with variable length records and COMMON block STECOM as a single record from that file. Access to COMMON block SYMCOM is via the hash table stored in COMMON block HSHCOM;

the hash table (the pointer to which is calculated by the function IHASH (Section 2.2.1.5)) points to a position in COMMON block SYMCOM (Figure 2-10). This position is the beginning of the symbol "record." In cases of hash collisions, the NEXT pointer points to the next symbol table "record" having the same hash value. The list search and comparison necessary to find a symbol is performed by LOOKS and a utility comparison routine, COMPAR. Necessary transfers between COMMON blocks STECOM and SYMCOM are performed by LOOKP and POKEP. COMMON block SYMCOM linkage also includes backward pointers (Figure 2-11). Unlinked pointers (upward at the top of the chain and downward at the bottom) are set to zero. Linking in of new symbols is accomplished in POKES, which utilizes the auxiliary pointer NEXSYM, which points to the next available unused position in COMMON block SYMCOM. A formatted sample symbol table dump was shown in Figure 2-8 and described in Section 2.1.5.6.

2.2.1.1 LOOKS

LOOKS (symbol look-up) searches the symbol table (COMMON block SYMCOM) for a specified input string. LOOKS requires as input, IHPNTR, the hash table pointer (hash value of the input string). The hash table value at IHPNTR points to the head of a symbol table chain, which is searched for the required string. An empty chain results in the symbol table pointer variable (IPOINT) being set to zero. A chain that is not empty but does not contain the desired string is indicated by a negative IPOINT value where the absolute value of IPOINT points to the last entry in the chain. If a matching string is located, IPOINT is returned pointing to the entry.

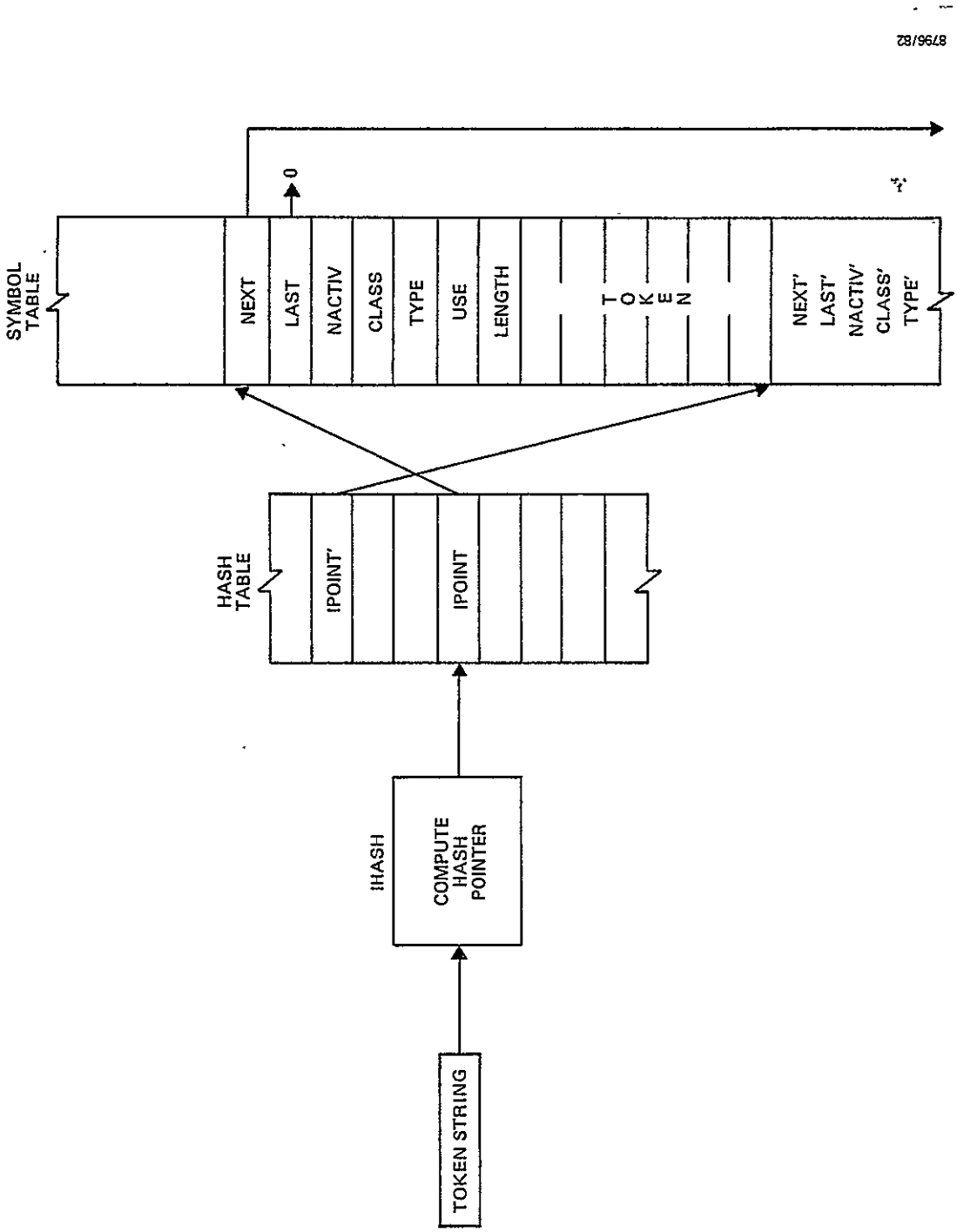


Figure 2-10. Symbol Table Access

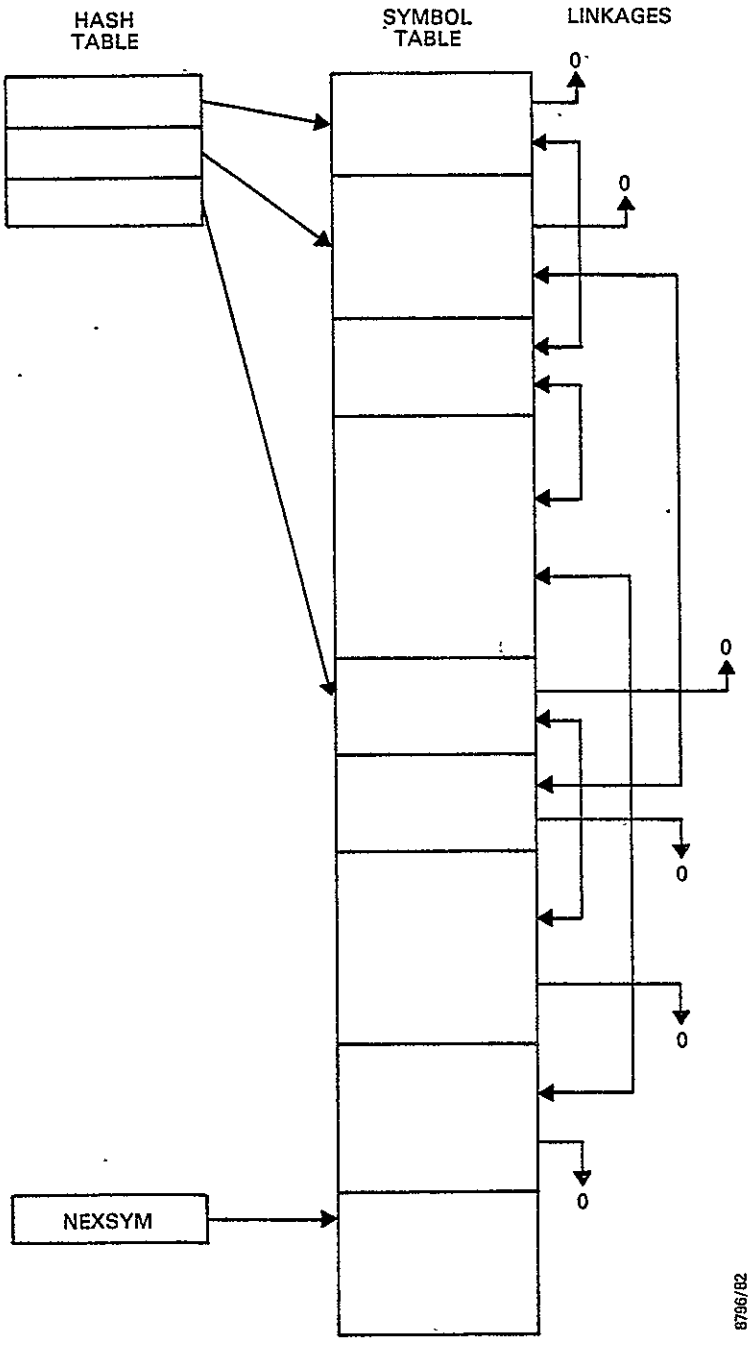


Figure 2-11. Symbol Table Linkages

2.2.1.2 LOOKP

LOOKP loads the symbol table entry beginning at IPOINT into COMMON block STECOM. If IPOINT is invalid, the error flag is set to .TRUE. and no transfer takes place.

2.2.1.3 POKES

POKES (poke entry into symbol table) establishes a new entry in the symbol table. POKES requires IPOINT (as defined in LOOKS) and IHPNTR. If IPOINT indicates that a chain exists, POKES updates that chain to point to NEXSYM. If a chain does not exist, one is started at NEXSYM (that is, IPOINT is set equal to NEXSYM, and the hash table is updated to point to NEXSYM). In either case, NEXSYM is checked against MAXSYM to see if the input symbol in COMMON block STECOM will fit. If it will not, GARCOL is called to compress COMMON block SYMCOM. If sufficient space is available, POKEP is called to insert the symbol; if not, the error flag is set to .TRUE.. In the case where IPOINT is greater than zero, POKES simply calls POKEP to insert the symbol at the already established location.

2.2.1.4 POKEP

POKEP (write a symbol table entry) moves the contents of COMMON block STECOM into COMMON block SYMCOM starting at location IPOINT. The error flag is set to .TRUE. if IPOINT is out of range.

2.2.1.5 IHASH

IHASH (compute a hash pointer) computes the hash table pointer by summing the characters in the input array STRING, shifting LHSHFT bits to the right, and masking out all but the low bits and adding one. LHSHFT and LHMASK (the bit mask) are stored in COMMON block HSHCOM and are set to zero and 1777 (octal), respectively.

2.2.1.6 KILLP

KILLP (delete a symbol table entry) deletes a symbol table entry by removing its linkages to the rest of the symbol table and marking it for compression. KILLP first calls LOOKP to load the symbol into COMMON block STECOM. If the forward and backward pointers, NEXT and LAST, are both zero, KILLP deletes the hash table entry by zeroing location IHPNTR in the IHTBLE array and sets location IPOINT in COMMON block SYMCOM to -1. If NEXT is 0 and LAST is non-zero, location IPOINT is set to -1 and location LAST is set to 0 (the chain is terminated at LAST). If NEXT and LAST are both nonzero locations, LAST is set to NEXT and NEXT is set to -1. If NEXT is not zero but LAST is 0, IHTBLE (IHPNTR) is set to NEXT and location IPOINT is set to -1. The error flag is set to .TRUE. if any illegal address is encountered.

2.2.1.7 GARCOL

GARCOL (symbol table compression) frees space by compressing out symbol table entries flagged for deletion by KILLP. GARCOL proceeds by starting at the top of the symbol table, calculating the length of the first entry, checking its forward pointer for a delete flag (-1), compressing out the entry if the delete flag is on, resetting the hash table entry to point to the new location (for head of chain only), and then iterating until NEXSYM is reached. NEXSYM is reset to point to the last entry +1 and GARCOL returns. Any illegal address calculation will cause error to be set to .TRUE..

2.2.2 DELIMITER/TOKEN TABLE UTILITY (LOOKAH)

The delimiter/token table is the result of the statement decomposition performed by subroutine DSCAN. The table is contained in array LISTDT in COMMON block LDTCOM. The entries in this table are either positive integers, which point to tokens in the symbol table (Section 2.2.1) or negative integers, which point to one of the delimiters in COMMON block DELCOM (Section 4). The sequence of pointers is terminated by a pointer to the null delimiter (IYNULL).

The interpretation of the contents of the delimiter/token table is specific to each individual statement parsing routine. Each parsing routine will advance a pointer through the delimiter/token table while performing a specific analysis of the FORTRAN statement. As the pointer is advanced, two functions are usually performed: (1) each token encountered is marked in the symbol table and (2) a limited syntax check is performed. One utility, LOOKAH, is used when examining the delimiter/token table.

LOOKAH (parsing look-ahead) searches the delimiter/token table for a specific entry. LOOKAH searches the delimiter/token table between specified limits until one of the following conditions is met:

- The end of the table is encountered
- An unmatched close parenthesis is encountered
- The end of the specified range in the table is encountered
- The first occurrence of the specified entry which is not enclosed within parentheses is encountered

2.2.3 TRANSFER OPERATOR LIST UTILITIES

The transfer operator list is used to track the occurrences of individual Halstead transfer operators. The transfer list and the pointers associated with the list are stored in COMMON block XFRCOM.

The transfer list is a set of six singly-linked lists that are built from the same list of available space. Each list is made up of nodes of variable lengths. Each node is made up of three or more cells. The first cell of each node points to the first cell of the next node in the list. If the node is the last node in a list, the first cell contains a zero. The second cell contains the use count of the transfer operator. The third cell contains a count of the number of cells belonging to the node which follow the third cell. The cells following the third cell contain pointers to the tokens in the symbol table which make up the transfer operator. Table 2-1 shows which tokens from the transfer operators are pointed to by these cells.

A set of utility routines is used to establish and maintain the transfer list. These routines are described below.

2.2.3.1 INITN

INITN (initialize the transfer lists) creates six empty nodes from the list of available space. These header nodes never contain data and serve only as starting points for each list. Six pointers to the header nodes are also established.

2.2.3.2 NEWPOT

NEWPOT (establish new potential node) obtains three cells from the list of available space and initializes each cell to zero. An error flag is returned as .TRUE. if there are insufficient cells remaining in the list of available space.

Table 2-1. Transfer Operators

Statement Type	Syntax	Token Pointer List
Alternate Return	CALL sub[([a[,a]...])]	List = sub,a,...,a where each argument(a) in the token list is an alternate return specifier label; this operator exists only if at least one argument is an alternate return
Any I/O statement	IO Keyword (...[,END=s]...)	List = s
Any I/O statement	IO Keyword (...[,ERR=s]...)	List = s
Unconditional GOTO	GOTO s	List = s
Computed GOTO	GOTO(s[,s]...)[,]i	List = s,...,s,i where the index (i) is included in the token list
Assigned GOTO	GOTO i[,](s[,s]...)]	List = i where the statement label list is not included in the token list

2.2.3.3 ADDPOT

ADDPOT (add a cell to potential node) obtains the next cell from the list of available space and attaches it to the potential node. An error flag is returned as .TRUE. if the list of available space is empty. A pointer to a token in the symbol table is placed in the new cell and the node's third (length) cell is incremented.

2.2.3.4 LOOKND

LOOKND (test potential node) compares the potential node to each node in a specified list. A match between the potential node and a node in the list occurs when the lists of pointers into the symbol table are the same. If a match is found, the potential node is erased and the use count cell in the matching node is incremented. If a match is not found, the potential node is linked into the list with a use count of one, and a new potential node is obtained.

2.2.3.5 LNKPOT

LNKPOT (link potential node to list) adds the potential node to the end of a specified list.

2.2.3.6 ERAPOT

ERAPOT (erase the potential node) returns the potential node's symbol table pointer cells to the list of available space. The potential node's third (length) cell is reset to zero.

SECTION 3 - SAP MODULE DESCRIPTIONS

The detailed module descriptions provided in this section are arranged alphabetically by module name. In addition to the modules listed, SAP uses the following system modules: ISHFT, ERRSET, DATE, and TIME.

All SAP modules are written in structured FORTRAN (Reference 8), although not all modules use the extensions permitted by this language.

ROUTINE: ADDPOT

TYPE: Subroutine

PURPOSE: Adds an item to the comparison portion of a potential node in the transfer operator list.

USAGE:

1. Calling Sequence:

CALL ADDPOT (ITEM, ERROR)

FORTRAN Name	I/O	Type	Dimension	Description
ITEM	I	I*2	-	Item to add to node
ERROR	O	L*2	-	= .FALSE., processing complete = .TRUE., not enough room to add item to node

2. COMMON Blocks Used: LUNCOM, XFRCOM

3. Subroutines Used: None

4. Subroutines Called by: PRCALL, PRGOTO, PRIO

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: ASGNID

TYPE: Subroutine

PURPOSE: Performs an initial scan of the delimiter/token list to recognize assignment statements.

USAGE:

1. Calling Sequence:

```
CALL ASGNID (LDTPTR, INDIC, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Pointer to start of statement in delimiter/token list
INDIC	O	I*2	-	= 0, statement is not assignment statement = 1, statement is an assignment statement
ERROR	O	L*2	-	= .FALSE., processing completed = .TRUE., unrecoverable error

2. COMMON Blocks Used: DELCOM, LDTCOM, MODCOM, STECOM, TYPCOM

3. Subroutines Used: LOOKAH, LOOKK, LOOKP

4. Subroutines Called by: TYPE

5. External Data Sets Referenced: None

ROUTINE: CINPUT

TYPE: Subroutine

PURPOSE: Requests the command line, interprets the switches, and opens the appropriate file.

USAGE:

1. Calling Sequence:

```
CALL CINPUT (ENDC, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ENDC	O	L*2	-	Control input end-of-file flag
ERROR	O	L*2	-	= .FALSE., processing complete = .TRUE., error opening source input file

2. COMMON Blocks Used: INFCOM, LUNCOM, SWICOM

3. Subroutines Used: COMPAR, INPUT, DEFSEL, INCLUD

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write
2	FOR002.DAT	Open
11	FOR011.DAT	Open

ROUTINE: CNTXFR

TYPE: Subroutine

PURPOSE: Accumulates the count of distinct operators and total operators from the transfer operator list.

USAGE:

1. Calling Sequence:

```
CALL CNTXFR
```

2. COMMON Blocks Used: OPCOM, XFRCOM

3. Subroutines Used: None

4. Subroutines Called by: MCMPLX

5. External Data Sets Referenced: None

ROUTINE: COEF

TYPE: Subroutine

PURPOSE: Computes the correlation coefficients for the project summary analysis.

USAGE:

1. Calling Sequence:

CALL COEF (NCOL, NLINES, K, TITLE)

FORTRAN Name	I/O	Type	Dimension	Description
NCOL	I	I*2	-	Number of measures correlated
NLINES	I	I*2	-	Number of modules correlated
K	I	I*4	(100, 10)	Matrix of data to be correlated
TITLE	I	R*8	(10)	Title of rows and columns

2. COMMON Blocks Used: LUNCOM

3. Subroutines Used: None

4. Subroutines Called by: REPHAL

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
8	FOR008.DAT	Write

ROUTINE: COLGLB

TYPE: Subroutine

PURPOSE: Collects the global statistics for output by routine STATG.

USAGE:

1. Calling Sequence:

```
CALL COLGLB
```

2. COMMON Blocks Used: CT1COM, CT2COM, CT3COM, CT4COM, CT5COM, GLBCOM, MODCOM, TYPCOM

3. Subroutines Used: None

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced: None

ROUTINE: COMPAR

TYPE: Subroutine

PURPOSE: Compares two strings of ASCII characters for equality.

USAGE:

1. Calling Sequence:

```
CALL COMPAR (STR1, STR2, L1, L2, SAME)
```

FORTRAN Name	I/O	Type	Dimension	Description
STR1	I	L*1	1	Comparison string one
STR2	I	L*1	1	Comparison string two
L1	I	I*2	-	Length of comparison string one
L2	I	I*2	-	Length of comparison string two
SAME	O	L	-	Truth switch: = .TRUE., strings equal = .FALSE., strings not equal

2. COMMON Blocks Used: LUNCOM

3. Subroutines Used: None

4. Subroutines Called by: LOOKS, PRDOS, PRIFS, WRTDB, PRIO, CINPUT

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: COMPWT

TYPE: Subroutine

PURPOSE: Computes the SEL complexity from the collected statistics and the current weights file data.

USAGE:

1. Calling Sequence:

   CALL COMPWT

2. COMMON Blocks Used: CT1COM, CT2COM, CT3COM, CT4COM, CT5COM, KEYCOM, WTSCOM

3. Subroutines Used: None

4. Subroutines Called by: MCMPLX

5. External Data sets Referenced: None

ROUTINE: DEFINE

TYPE: Subroutine

PURPOSE: Initializes or locates a data base file when the /DB control switch is set true. Prompts user for a data base name, maximum record count, and project character to be used for identification in the correlation coefficient report.

USAGE:

1. Calling Sequence:

```
CALL DEFINE (DBFILE, PROJ)
```

FORTRAN Name	I/O	Type	Dimension	Description
DBFILE	O	L*1	70	Data base file name
PROJ	O	L*1	1	Project character

2. COMMON Blocks Used: LUNCOM

3. Subroutines Used: FINDIT

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
9	User supplied	Open, write, close
5	Terminal	Read
6	FOR006.DAT	Write

ROUTINE: DEFSEL

TYPE: Subroutine

PURPOSE: Opens the ALL.SAP sequential file if control switch /SL is set to on. If an ALL.SAP file exists in the user's default directory, the file is opened with the APPEND option; otherwise it is opened as NEW.

USAGE:

1. Calling Sequence:

```
CALL DEFSEL
```

2. COMMON Blocks Used: LUNCOM, SELCOM

3. Subroutines Used: None

4. Subroutines Called by: CINPUT

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
5	Terminal	Read
6	FOR006.DAT	Write
12	ALL.SAP	Open

ROUTINE: DSCAN

TYPE: Subroutine

PURPOSE: Scans the packed input array, locating delimiters and testing tokens against the symbol table. Any new tokens are entered into the symbol table, and a list of delimiters and tokens is created in /LDTCOM/.

USAGE:

1. Calling Sequence:

```
CALL DSCAN (ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ERROR	O	L*2	-	= .FALSE., processing complete = .TRUE., error in locating and/or entering token in symbol table

2. COMMON Blocks Used: DLICOM, INPCOM, LDTCOM, LUNCOM, STECOM

3. Subroutines Used: IHASH, LOOKS, NUMER, POKES

4. Subroutines Called by: TYPE

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: ERAPOT

TYPE: Subroutine

PURPOSE: Resets the potential node in the transfer operator list to empty.

USAGE:

1. Calling Sequence:

```
CALL ERAPOT
```

2. COMMON Blocks Used: XFRCOM

3. Subroutines Used: None

4. Subroutines Called by: LOOKND, PRCALL, PRGOTO, PRIO

5. External Data Sets Referenced: None

ROUTINE: ERRMSG

TYPE: Subroutine

PURPOSE: Lists the source statement and delimiter/token list contents that have caused a syntax error during SAP processing.

USAGE:

1. Calling Sequence:

```
CALL ERRMSG (LIST, PARSED, LDTPTR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LIST	I	L*2	-	= .TRUE., print card image
PARSED	I	L*2	-	= .TRUE., print card image by token and delimiter
LDTPTR	I	I*2	-	Pointer to beginning of card image in LISTDT array

2. COMMON Blocks Used: DLICOM, INPCOM, LDTCOM, LUNCOM, MODCOM, STECOM

3. Subroutines Used: LOOKP

4. Subroutines Called by: PRTOKE, STATE

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: ESTIM

TYPE: Subroutine

PURPOSE: Computes a number of Halstead's complexity measures (predicted program length, program volume, potential volume, language and program level, effort required, programming time, and predicted bugs).

USAGE:

1. Calling Sequence:

```
CALL ESTIM (ICTHIO, IETA1, IETA2, NETA1, NETA2,
            IETA, NETA, LENGTH, IVOL, PRGLVL,
            ALNGLV, IEFORT, TOTIM, NBUGS, IVSTAR,
            STROUD, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ICTHIO	I	I*2	-	Sum of count of argument variables (including ENTRY arguments) and count of referenced COMMON variables
IETA1	I	I*2	-	Number of unique operators
IETA2	I	I*2	-	Number of unique operands
NETA1	I	I*2	-	Total number of operators
NETA2	I	I*2	-	Total number of operands
IETA	O	I*2	-	Number of unique operators and operands
NETA	O	I*2	-	Total number of operators and operands
LENGTH	O	I*2	-	Predicted length
IVOL	O	I*2	-	Program volume
PRGLVL	O	R*4	-	Program level
ALNGLV	O	R*4	-	Language level
IEFORT	O	I*2	-	Effort required
TOTIM	O	R*4	-	Total program time required in hours
NBUGS	O	I*2	-	Predicted number of bugs

FORTRAN Name	I/O	Type	Dimension	Description
IVSTAR	O	I*2	-	Potential volume
STROUD	O	I*2	-	Stroud number (discriminations/hour)
ERROR	O	L*2	-	Error flag

2. COMMON Blocks Used: None

3. Subroutines Used: None

4. Subroutines Called by: REPHAL

5. External Data Sets Referenced: None

ROUTINE: FINDIT

TYPE: Subroutine

PURPOSE: Extracts a character string, up to a specified delimiter, from an input character string.

USAGE:

1. Calling Sequence:

CALL FINDIT (INFILE, IC, DELIM, N, OUTPUT, ICX)

FORTRAN Name	I/O	Type	Dimension	Description
INFILE	I	L*1	80	Input source character string
IC	I/O	I*2	-	Number of characters processed in INFILE
DELIM	I	L*1	-	Delimiter character
N	I	I*2	-	Number of characters in INFILE
OUTPUT	O	L*1	80	Extracted character string up to delimiting character
ICX	O	I*2	-	Number of characters in OUTPUT

2. COMMON Blocks Used: None

3. Subroutines Used: None

4. Subroutines Called by: DEFINE, INCLUD

5. External Data Sets Referenced: None

ROUTINE: FLVARI

TYPE: Subroutine

PURPOSE: Flags variables and arrays in the symbol table and counts array dimensions.

USAGE:

1. Calling Sequence:

```
CALL FLVARI (IC, MASK, SYNERR, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
IC	I/O	I*2	-	Pointer to next location within the delimiter/token list
MASK	I	I*2	-	Mask for numeric or character data types
SYNERR	O	L*2	-	Syntax error flag: = .FALSE., no syntax error = .TRUE., syntax error
ERROR	O	L*2	-	Fatal error flag: = .FALSE., processing complete = .TRUE., error processing symbol table entries

2. COMMON Blocks Used: CT2COM, CT5COM, DELCOM, LDTCOM, LUNCOM, STECOM

3. Subroutines Used: LOOKP, POKEP, PAGER

4. Subroutines Called by: PRSPEC, PRTYPE

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: FNNAME

TYPE: Subroutine

PURPOSE: Extracts a character string, up to and including a specified delimiter, from an input character string.

USAGE:

1. Calling Sequence:

```
CALL FNNAME (INFILE, IC, DELIM, N, OUTPUT, ICX)
```

FORTRAN Name	I/O	Type	Dimension	Description
INFILE	I	L*1	80	Input character string
IC	I/O	I*2	-	Location of delimiter within the input string
DELIM	I	L*1	-	Specified delimiter
N	I	I*2	-	Number of characters in INFILE
OUTPUT	O	L*1	80	Extracted character string including delimiter
ICX	O	I*2	-	Number of characters in OUTPUT

2. COMMON Blocks Used: None

3. Subroutines Used: None

4. Subroutines Called by: INCLUD

5. External Data Sets Referenced: None

ROUTINE: GARCOL

TYPE: Subroutine

PURPOSE: Compresses the symbol table by removing areas flagged for deletion and relinking the chain pointers.

USAGE:

1. Calling Sequence:

```
CALL GARCOL (ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: HSHCOM, LDTCOM, LUNCOM, SYMCOM

3. Subroutines Used: IHASH

4. Subroutines Called by: POKES

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: GLINE

TYPE: Subroutine

PURPOSE: Reads input source code into a two-line rotating buffer.

USAGE:

1. Calling Sequence:

```
CALL GLINE (INITR, ICOMM, NCOMM, ICONT, NCONT, ENDN,
            ENDS, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
INITR	I/O	L*2	-	Initial read flag
ICOMM	I/O	L*2	-	Current card comment flag
NCOMM	I/O	L*2	-	Next card comment flag
ICONT	I/O	L*2	-	Current card continuation flag
NCONT	I/O	L*2	-	Next card continuation flag
ENDN	O	L*2	-	End of input on read flag
ENDS	O	L*2	-	End of input on initial read flag
ERROR	O	L*2	-	Read error flag

2. COMMON Blocks Used: INLCOM, LUNCOM, MODCOM, SWICOM

3. Subroutines Used: TABCCC, PAGER

4. Subroutines Called by: READER

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
2	FOR002.DAT	Read
6	FOR006.DAT	Write

ROUTINE: HALREP

TYPE: Subroutine

PURPOSE: Prints the specific Halstead operators (delimiters, keywords, procedures, and transfers) and operands when the /HL control switch is set to on.

USAGE:

1. Calling Sequence:

```
CALL HALREP
```

2. COMMON Blocks Used: DLICOM, HSHCOM, LUNCOM, OPCOM, STECOM

3. Subroutines Used: LOOKP, PAGER, PRTXFR

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write
7	FOR007.DAT	Write

ROUTINE: HOPRN

TYPE: Subroutine

PURPOSE: Increments the activity pointer for a symbol in the symbol table when a Halstead operand has been encountered.

USAGE:

1. Calling Sequence:

```
CALL HOPRN (IPOINT)
```

FORTRAN Name	I/O	Type	Dimension	Description
IPOINT	I	I*2	-	Starting location for symbol block in symbol table

2. COMMON Blocks Used: STECOM

3. Subroutines Used: POKEP

4. Subroutines Called by: PRASGN, PRASS, PRCALL, PRDOS, PRGOTO, PRIFS, PRIO, PRSTRC

5. External Data Sets Referenced: None

ROUTINE: HOPTR1

TYPE: Subroutine

PURPOSE: Determines whether a given delimiter is a Halstead operator and increments the associated counter.

USAGE:

1. Calling Sequence:

```
CALL HOPTR1 (IDLM)
```

FORTRAN Name	I/O	Type	Dimension	Description
IDLM	I	I*2	-	Delimiter code from delimiter/token table

2. COMMON Blocks Used: OPCOM

3. Subroutines Used: None

4. Subroutines Called by: PRSTRC

5. External Data Sets Referenced: None

ROUTINE: HOPTR3

TYPE: Subroutine

PURPOSE: Increments the counter corresponding to the procedure (subroutine or function) specified by the current symbol in the delimiter/token table.

USAGE:

1. Calling Sequence:

   CALL HOPTR3

2. COMMON Blocks Used: LUNCOM, OPCOM, STECOM

3. Subroutines Used: None

4. Subroutines Called by: PRASGN, PRIFS, PRSTRC

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: HPR1

TYPE: Subroutine

PURPOSE: Calculates the contributions to the unique and total operator counts from the delimiter operators.

USAGE:

1. Calling Sequence:

```
CALL HPR1 (LINE)
```

FORTRAN Name	I/O	Type	Dimension	Description
LINE		I*2	-	Not used

2. COMMON Blocks Used: DLICOM, LUNCOM, OPCOM

3. Subroutines Used: None

4. Subroutines Called by: MCMPLX

5. External Data Sets Referenced: None

ROUTINE: HPR2

TYPE: Subroutine

PURPOSE: Calculates the contributions to the unique and total operator counts from the keyword operators.

USAGE:

1. Calling Sequence:

```
CALL HPR2 (LINE)
```

FORTRAN Name	I/O	Type	Dimension	Description
LINE		I*2	-	Not used

2. COMMON Blocks Used: LUNCOM, OPCOM

3. Subroutines Used: None

4. Subroutines Called by: MCMPLX

5. External Data Sets Referenced: None

ROUTINE: HPR3

TYPE: Subroutine

PURPOSE: Calculates the contribution to the unique and total operator counts from the procedure operators.

USAGE:

1. Calling Sequence:

```
CALL HPR3 (LINE)
```

FORTRAN Name	I/O	Type	Dimension	Description
LINE		I*2	-	Not used

2. COMMON Blocks Used: LUNCOM, OPCOM.

3. Subroutines Used: None

4. Subroutines Called by: MCMPLX

5. External Data Sets Referenced: None

ROUTINE: HPRNDS

TYPE: Subroutine

PURPOSE: Calculates the count of unique and total operands from a scan of the symbol table.

USAGE:

1. Calling Sequence:

```
CALL HPRNDS
```

2. COMMON Blocks Used: HSHCOM, LUNCOM, OPCOM, STECOM, SYMCOM

3. Subroutines Used: IHASH, LOOKP

4. Subroutines Called by: None

5. External Data Sets Referenced: None

ROUTINE: HSCAN

TYPE: Subroutine

PURPOSE: Scans the input line, removing literals, Hollerith strings, embedded blanks, and inline comments.

USAGE:

1. Calling Sequence:

```
CALL HSCAN (ICTSXP, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ICTSXP	O	I*2	-	Inline comment counter
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: INPCOM, LUNCOM, MODCOM

3. Subroutines Used: None

4. Subroutines Called by: READER

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: IHASH

TYPE: Function

PURPOSE: Hashes the input character string to obtain a pointer into the symbol table.

USAGE:

1. Calling Sequence:

```
IHASH (STRING, LHASH)
```

FORTRAN Name	I/O	Type	Dimension	Description
IHASH	O	I*2	-	Hash value of STRING
STRING	I	L*1	1	Input character string to be hashed
LHASH	I	I*2	-	Length of input string

2. COMMON Blocks Used: HSHCOM, LUNCOM

3. Subroutines Used: None

4. Subroutines Called by: POKES

5. External Data Sets Referenced: None

ROUTINE: INCLUD

TYPE: Subroutine

PURPOSE: Expands INCLUDE statements, nested up to three deep, when the /XP control switch is set on.

USAGE:

1. Calling Sequence:

```
CALL INCLUD (FILEI, FILEO, NPS)
```

FORTRAN Name	I/O	Type	Dimension	Description
FILEI	I	L*1	72	Input source file name
FILEO	O	L*1	72	Expanded source file name
NPS	I	I*2	-	Length of name in FILEI

2. COMMON Blocks Used: LUNCOM

3. Subroutines Used: FNNAME

4. Subroutines Called by: CINPUT

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
1	From INCLUDE statement	Open, read, close
2	From INCLUDE statement	Open, read, close
3	From INCLUDE statement	Open, read, close
4	From INCLUDE statement	Open, read, close
11	FOR011.DAT	Open, write, close
6	FOR006.DAT	Write

ROUTINE: INITG

TYPE: Subroutine

PURPOSE: Initializes symbol table and global counter variables.

USAGE:

1. Calling Sequence:

```
CALL INITG (ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: GLBCOM, LUNCOM, SYMCOM, WTSCOM

3. Subroutines Used: None

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced: None

ROUTINE: INITM

TYPE: Subroutine

PURPOSE: Initializes the symbol table and the module counter variables.

USAGE:

1. Calling Sequence:

```
CALL INITM
```

2. COMMON Blocks Used: CT1COM, CT2COM, CT3COM, CT4COM, CT5COM, DLICOM, OPCOM, HSHCOM, IMPCOM, LBLCOM, LUNCOM, MODCOM, SYMCOM

3. Subroutines Used: None

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced: None

ROUTINE: INITN

TYPE: Subroutine

PURPOSE: Creates the initial header node for the transfer lists.

USAGE:

1. Calling Sequence:

```
CALL INITN (ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ERROR	O	L*2	-	= .FALSE., processing complete = .TRUE., error creating first potential node

2. COMMON Blocks Used: XFRCOM

3. Subroutines Used: NEWPOT

4. Subroutines Called by: INITM

5. External Data Sets Referenced: None

ROUTINE: INPUT

TYPE: Subroutine

PURPOSE: Obtains a line of control input from the user. The user may specify an indirect file to be used as a source of control input until the file is exhausted.

USAGE:

1. Calling Sequence:

```
CALL INPUT (PROMPT, RSPOND, LENRSP, MAXRSP, EXTFIL,
            TERM, EOFTRM)
```

FORTRAN Name	I/O	Type	Dimension	Description
PROMPT	I	L*1	1	Prompt displayed when requesting from terminal or echoing from indirect file (Must be terminated by '@' character)
RSPOND	O	L*1	1	Input string
LENRSP	O	I*2	-	Length of input string
MAXRSP	I	I*2	-	Maximum length of input string allowed
EXTFIL	I	I*2	-	Logical unit number for indirect file
TERM	I/O	L*1	-	Input logical unit flag: = .TRUE., terminal is current input file = .FALSE., indirect file is current input file
EOFTERM	O	L*1	-	= .TRUE., last input from terminal was end of file character (CNTL Z) = .FALSE., no end of file from terminal

2. COMMON Blocks Used: None

3. Subroutines Used: LOCCHR, SKPCHR

4. Subroutines Called by: CINPUT

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
5	Terminal	Read
6	FOR006.DAT	Write
10	User supplied	Open, read, close

ROUTINE: INTGR4

TYPE: Subroutine

PURPOSE: Converts a character string to INTEGER*4 internal form.

USAGE:

1. Calling Sequence:

```
CALL INTGR4 (STRING, L, N, SYNERR)
```

FORTRAN Name	I/O	Type	Dimension	Description
STRING	I	L*1	1	Input string for conversion
L	I	I*2	-	Length of input string
N	O	I*4	-	INTEGER*4 value of string
SYNERR	O	L*2	-	Conversion syntax error flag

2. COMMON Blocks Used: LUNCOM, MODCOM

3. Subroutines Used: PAGER

4. Subroutines Called by: PRCALL, PRIO, PRGOTO, PRASS, LABEL, PRDOS, PRIMPL

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: KILLP

TYPE: Subroutine

PURPOSE: Unlinks an entry from the symbol table and flags it for deletion by routine GARCOL.

USAGE:

1. Calling Sequence:

```
CALL KILLP (IPOINT, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
IPOINT	I	I*2	-	Pointer to entry to be unlinked
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: HSHCOM, LUNCOM, STECOM, SYMCOM

3. Subroutines Used: IHASH, LOOKP, POKEP

4. Subroutines Called by: PRDOS, PRTYPE, TESTK

5. External Data Sets Referenced: None

ROUTINE: LABEL

TYPE: Subroutine.

PURPOSE: Checks statement labels and, if required, adds them to the label list. Checks labels against the DO loop target stack and, if required, pops the stack and gathers DO loop statistics.

USAGE:

1. Calling Sequence:

```
CALL LABEL (ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT5COM, DELCOM, LBLCOM, LDTCOM, LUNCOM, MODCOM, STECOM

3. Subroutines Used: LOOKP, INTGR4, LABLST

4. Subroutines Called by: TYPE

5. External Data Sets Referenced: None

ROUTINE: LABLST

TYPE: Subroutine

PURPOSE: Checks whether a referenced label is in the label list. If not found, adds it to the list.

USAGE:

1. Calling Sequence:

```
CALL LABLST (LABL, LOC, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LABL	I	I*4	-	Integer representation of statement label
LOC	O	I*2	-	Location of label in array LABLST
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: LBLCOM

3. Subroutines Used: None

4. Subroutines Called by: LABEL, PRCALL, PRDOS, PRGOTO, PRASS, PRIO

5. External Data Sets Referenced: None

ROUTINE: LNKPOT

TYPE: Subroutine

PURPOSE: Links a potential node into a specific transfer operator list.

USAGE:

1. Calling Sequence:

```
CALL LNKPOT (LIST)
```

FORTRAN Name	I/O	Type	Dimension	Description
LIST	I	I*2	-	Pointer to header node of transfer operator list

2. COMMON Blocks Used: XFRCOM

3. Subroutines Used: None

4. Subroutines Called by: LOOKND

5. External Data Sets Referenced: None

ROUTINE: LOADK

TYPE: Subroutine

PURPOSE: Loads the file KEYWORDS.SAP into KEYCOM.

USAGE:

1. Calling Sequence:

```
CALL LOADK (ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ERROR	O	L*2	-	= .FALSE., processing complete = .TRUE., error opening or reading KEYWORDS.SAP

2. COMMON Blocks Used: KEYCOM, LUNCOM, SWICOM

3. Subroutines Used: USRWTS

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
1	KEYWORDS.SAP	Open, read, close

ROUTINE: LOCCHR

TYPE: Function

PURPOSE: Locates the first occurrence of a specified character starting at the beginning of a character string.

USAGE:

1. Calling Sequence:

```
LOCCHR (CHAR, STRING, LENGTH)
```

FORTRAN Name	I/O	Type	Dimension	Description
LOCCHR	O	I*2	-	= 0, character not found in STRING > 0, location of character within STRING
CHAR	I	L*1	-	Character to be searched for
STRING	I	L*1	LENGTH	Character string to be searched
LENGTH	I	I*2	-	Length of character string in bytes

2. COMMON Blocks Used: None

3. Subroutines Used: None

4. Subroutines Called by: INPUT

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
5	Terminal	Write

ROUTINE: LOOKAH

TYPE: Subroutine

PURPOSE: Searches for a target item between specified limits in the delimiter/token table. Sets a pointer to the first occurrence of the target that is not enclosed within parentheses.

USAGE:

1. Calling Sequence:

CALL LOOKAH (LOOKFR, ISTART, IEND, IPTR, ERROR)

FORTRAN Name	I/O	Type	Dimension	Description
LOOKFR	I	I*2	-	Target to search for
ISTART	I	I*2	-	Start location in delimiter/token table
IEND	I	I*2	-	End location in the delimiter/token table
IPTR	O	I*2	-	= 0, target not found because it was between parenthesis or an unmatched close parenthesis was found or end of the delimiter/token table was encountered ≠ 0, position in the delimiter/token table
ERROR	O	L*2	-	= .FALSE., processing complete = .TRUE., encountered the end of the delimiter/token table

2. COMMON Blocks Used: DELCOM, LDTCOM

3. Subroutines Used: None

4. Subroutines Called by: ASGNID, PRIFS

5. External Data Sets Referenced: None

ROUTINE: LOOKK

TYPE: Subroutine

PURPOSE: Looks within keyword table for a match to the token. A match is indicated even when only the leading part of the token is the same as a keyword.

USAGE:

1. Calling Sequence:

CALL LOOKK (STRING, L, IKEY, LK, ISCLAS, IEXEC)

FORTRAN Name	I/O	Type	Dimension	Description
STRING	I	L*4	1	Input string to be tested for keyword
L	I	I*2	-	Length of STRING
IKEY	O	I*2	-	Integer index of located keyword, if found; otherwise, set to zero
LK	O	I*2	-	Length of keyword pointed to by IKEY
ISCLAS	O	I*2	-	Statement class corresponding to keyword
IEXEC	O	L*2	-	Executability flag of keyword

2. COMMON Blocks Used: KEYCOM, TYPCOM

3. Subroutines Used: None

4. Subroutines Called by: ASGNID, TESTK, PRIMPL, PRTYPE

5. External Data Sets Referenced: None

ROUTINE: LOOKND

TYPE: Subroutine

PURPOSE: Searches for a match to the potential node in a specific transfer operator list. If a match is found, it is counted and the potential node is erased. If no match is found, the potential node is added to the list.

USAGE:

1. Calling Sequence:

```
CALL LOOKND (LIST,ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LIST	I	I*2	-	Pointer to header node of specific list to search
ERROR	O	L*2	-	= .FALSE., processing complete = .TRUE., could not obtain a new potential node

2. COMMON Blocks Used: XFRCOM

3. Subroutines Used: ERAPOT, LNKPOT, NEWPOT

4. Subroutines Called by: PRIO, PRGOTO, PRCALL

5. External Data Sets Referenced: None

ROUTINE: LOOKP

TYPE: Subroutine

PURPOSE: Locates the token starting at position IPOINT in the symbol table and loads it into COMMON /STECOM/.

USAGE:

1. Calling Sequence:

```
CALL LOOKP (IPOINT, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
IPOINT	I	I*2	-	Pointer to desired token
ERROR	O	L*2	-	= .FALSE., processing complete = .TRUE., when IPOINT is out of range

2. COMMON Blocks Used: LUNCOM, MODCOM, STECOM, SYMCOM

3. Subroutines Used: None

4. Subroutines Called by: ASGNID, ERRMSG, FLVARI, HPRNDS, KILLP, LABEL, POKES, PRASGN, PRASS, PRCALL, PRDOS, PRGOTO, PRIFS, PRIMPL, PRIO, PRSPEC, PRTOKE, PRTYPE, STDUMP, TABLES, TESTK

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: LOOKS

TYPE: Subroutine

PURPOSE: Searches the symbol table for STRING and returns a pointer to the corresponding symbol table entry.

USAGE:

1. Calling Sequence:

CALL LOOKS (IHPNTR, STRING, L, IPOINT, ERROR)

FORTRAN Name	I/O	Type	Dimension	Description
IHPNTR	I	I*2	-	Hash table pointer
STRING	I	L*1	1	String to be located
L	I	I*2	-	Length of STRING
IPOINT	O	I*2	-	Symbol table pointer: > 0, pointer value = 0, no pointer value < 0, pointer magnitude set to last entry
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: HSHCOM, STECOM

3. Subroutines Used: COMPAR, IHASH, LOOKP

4. Subroutines Called by: DSCAN, PRDOS, PRTYPE, TESTK

5. External Data Sets Referenced: None

ROUTINE: MCMPLX

TYPE: Subroutine

PURPOSE: Computes the module complexities. Writes assembled data to the data base if the /DB control switch set to on and to ALL.SAP if the /SL control switch is set on.

USAGE:

1. Calling Sequence:

```
CALL MCMPLX (DBFILE, PROJ)
```

FORTRAN Name	I/O	Type	Dimension	Description
DBFILE	I	L*1	70	Name of data base file in use
PROJ	I	L*1	-	Current project character to tag module name in data base

2. COMMON Blocks Used: CT1COM, CT2COM, CT3COM, CT4COM, CT5COM, DELCOM, MODCOM, OPCOM, SELCOM, SWICOM, TYPCOM, WTSCOM

3. Subroutines Used: CNTXFR, COMPWT, HPR1, HPR2, HPR3, HPRNDS, PRTHAL, UCPLX1, UCPLX2, WRTDB, WRTSEL

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced: None

ROUTINE: MDIRY

TYPE: Subroutine

PURPOSE: Generates the module directory listing.

USAGE:

1. Calling Sequence:

```
CALL MDIRY (INLPAG, LASTPG, IPRTLN, FIRST, KNT)
```

FORTRAN Name	I/O	Type	Dimension	Description
INLPAG	I	I*2	-	Page number for module summary for this module
LASTPG	I/O	I*2	-	Page counter for directory file
IPRTLN	I/O	I*2	-	Total line counter (including blank lines)
FIRST	I	L*2	-	Page header switch for first page header
KNT	I	I*2	-	Printed line counter

2. COMMON Blocks Used: CT1COM, CT2COM, LUNCOM, MODCOM, OPCOM, PAGCOM, SWICOM, WTSCOM

3. Subroutines Used: PAGER

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
8	FOR008.DAT	Write

ROUTINE: NEWPOT

TYPE: Subroutine

PURPOSE: Creates the header portion of a potential node in the transfer operator list.

USAGE:

1. Calling Sequence:

```
CALL NEWPOT (ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ERROR	O	L*2	-	= .FALSE., processing completed = .TRUE., insufficient space for creating a new potential node

2. COMMON Blocks Used: LUNCOM, XFRCOM

3. Subroutines Used: None

4. Subroutines Called by: INITN, LOOKND

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: NUMER

TYPE: Subroutine

PURPOSE: Determines whether a character is numeric (including decimal points) or nonnumeric.

USAGE:

1. Calling Sequence:

```
CALL NUMER (IN, ANSWER)
```

FORTRAN Name	I/O	Type	Dimension	Description
IN	I	L*1	-	Character to be tested
ANSWER	O	L*2	-	= .FALSE., nonnumeric = .TRUE., numeric or decimal point

2. COMMON Blocks Used: None

3. Subroutines Used: None

4. Subroutines Called by: DSCAN, PRASS, PRDOS, TESTK

5. External Data Sets Referenced: None

ROUTINE: OPERAT

TYPE: Subroutine

PURPOSE: Determines whether a delimiter is an operator, and returns the operator classification.

USAGE:

1. Calling Sequence:

```
CALL OPERAT (ID, IOP)
```

FORTRAN Name	I/O	Type	Dimension	Description
ID	I	I*2	-	Delimiter code as defined in DLICOM common
IOP	O	I*2	-	Operator classification = 0, nonoperator = 1, arithmetic operator = 2, relational operator = 3, Boolean operator

2. COMMON Blocks Used: None

3. Subroutines Used: None

4. Subroutines Called by: PRTOKE, PRASGN

5. External Data Sets Referenced: None

ROUTINE: PAGER

TYPE: Subroutine

PURPOSE: Maintains the line and page counts for listing files, prints a page header when lines to be written exceed page line maximum.

USAGE:

1. Calling Sequence:

```
CALL PAGER (LINES, LUN, ILINE, IPAGE)
```

FORTRAN Name	I/O	Type	Dimension	Description
LINES	I	I*2	-	Number of lines to be written
LUN	I	I*2	-	LUN on which write is to occur
ILINE	O	I*2	-	New line count for LUN
IPAGE	O	I*2	-	Current page count for LUN

2. COMMON Blocks Used: INFCOM, MODCOM, PAGCOM

3. Subroutines Used: DATE, TIME

4. Subroutines Called by: COEF, GLINE, HALREP, HSCAN, INTGR4, MDIRY, NEWPOT, POKES, PRASGN, PRCALL, PRDOS, PRGOTO, PRIFS, PRIO, PRSPEC, PRSUBS, PRTHAL, PRTOKE, PRTXFR, REPHAL, STATG, STATM, STDUMP

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
specified unit numbers		Write

ROUTINE: POKEP

TYPE: Subroutine

PURPOSE: Transfers the token block in /STECOM/ into the symbol table.

USAGE:

1. Calling Sequence:

```
CALL POKEP (IPOINT, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
IPOINT	I	I*2	-	Starting location for insertion in SYMCOM
ERROR	O	L*2	-	= .FALSE., processing completed = .TRUE., IPOINT out of symbol table range

2. COMMON Blocks Used: LUNCOM, MODCOM, SYMCOM, STECOM

3. Subroutines Used: None

4. Subroutines Called by: FLVARI, HOPRN, KILLP, POKES, PRASGN, PRCALL, PRSPEC, PRSUBS, PRTOKE

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: POKES

TYPE: Subroutine

PURPOSE: Inserts a string into the symbol table. Creates a new token block, if one does not exist.

USAGE:

1. Calling Sequence:

```
CALL POKES (IHPNTR, IPOINT, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
IHPNTR	I	I*2	-	Hash table pointer
IPOINT	I	I*2	-	Symbol table pointer
ERROR	O	L*2	-	= .FALSE., processing completed = .TRUE., IPOINT out of symbol table range

2. COMMON Blocks Used: HSHCOM, LUNCOM, MODCOM, STECOM, SYMCOM

3. Subroutines Used: GARCOL, LOOKP, PAGER, POKEP

4. Subroutines Called by: DSCAN, PRDOS, PRTYPE, TESTK

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: PRASGN

TYPE: Subroutine

PURPOSE: Parses assignment statements; identifies arithmetic statement function definitions.

USAGE:

1. Calling Sequence:

```
CALL PRASGN (LDTPTR, ISCLAS, ISTYPE, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Points to next location in delimiter/token table
ISCLAS	O	I*2	-	Statement class
ISTYPE	O	I*2	-	Statement type
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT2COM, CT5COM, DELCOM, LDTCOM, LUNCOM, MODCOM, OPCOM, STECOM, TYPCOM

3. Subroutines Used: HOPRN, HOPTR1, HOPTR3, LOOKP, OPERAT, PAGER, POKEP, PRTOKE

4. Subroutines Called by: STATE

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: PRASS

TYPE: Subroutine

PURPOSE: Parses the ASSIGN statement, adding the referenced label to the label list array.

USAGE:

1. Calling Sequence:

```
CALL PRASS (LDTPTR, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Points to next location in delimiter/token table
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: LDTCOM, LBLCOM, OPCOM, STECOM

3. Subroutines Used: LABLST, INTGR4, LOOKP, NUMER

4. Subroutines Called by: PRCNTL

5. External Data Sets Referenced: None

ROUTINE: PRCALL

TYPE: Subroutine

PURPOSE: Parses CALL statements.

USAGE:

1. Calling Sequence:

```
CALL PRCALL (LDTPTR, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Points to next location in delimiter/token table
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT5COM, DELCOM, LBLCOM, LDTCOM, LUNCOM, MODCOM, OPCOM, STECOM, XFRCOM

3. Subroutines Used: ADDPOT, ERAPOT, HOPTR1, HOPTR3, HOPRN, INTGR4, LABLST, LOOKND, LOOKP, PAGER, POKEP, PRTOKE

4. Subroutines Called by: PRCNTL

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: PRCNTL

TYPE: Subroutine

PURPOSE: Controls the processing of control statements. Actual analysis will be performed by one of the called routines.

USAGE:

1. Calling Sequence:

```
CALL PRCNTL (LDTPTR, ISTYPE, IREPT, LREPT, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I/O	I*2	-	Points to next location in delimiter/token table
ISTYPE	I/O	I*2	-	Statement type being processed
IREPT	I/O	L*2	-	Repeat flag, set in routine PRIFS when this statement is a logical IF
LREPT	I/O	L*2	-	Set if this statement is object of a logical IF
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: DELCOM, LDTCOM, TYPCOM

3. Subroutines Used: PRCALL, PRGOTO, PRASS, PRDOS, PRIFS, PRRET

4. Subroutines Called by: STATE

5. External Data Sets Referenced: None

ROUTINE: PRDOS

TYPE: Subroutine

PURPOSE: Parses DO statements by performing an initial scan of the delimiter/token table. Determines whether the DO statement is a DOWHILE statement.

USAGE:

1. Calling Sequence:

```
CALL PRDOS (LDTPTR, ISTYPE, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Points to next location in delimiter/token table
ISTYPE	I/O	I*2	-	Statement type
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT5COM, DELCOM, LBLCOM, LDTCOM, MODCOM, OPCOM, STECOM, TYPCOM

3. Subroutines Used: COMPAR, HOPTR1, HOPTR3, HOPRN, IHASH, INTGR4, KILLP, LOOKP, LOOKS, NUMER, PAGER, POKEP, POKES, PRTOKE

4. Subroutines Called by: PRCNTL, PRSTRC

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: PRGOTO

TYPE: Subroutine

PURPOSE: Parses GOTO statements.

USAGE:

1. Calling Sequence:

```
CALL PRGOTO (LDTPTR, LREPT, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Points to next location in delimiter/token table
LREPT	I	L*2	-	Indicates statement is object of a logical IF statement
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT5COM, DELCOM, LBLCOM, LDTCOM, LUNCOM, MODCOM, OPCOM, STECOM, XFRCOM

3. Subroutines Used: ADDPOT, ERAPOT, HOPRN, HOPTR1, INTGR4 LABLST, LOOKND, LOOKP, PAGER, PRTOKE

4. Subroutines Called by: PRCNTL

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: PRIFS

TYPE: Subroutine

PURPOSE: Parses IF statements.

USAGE:

1. Calling Sequence:

```
CALL PRIFS (LDTPTR, ISTYPE, IREPT, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Points to next location in delimiter/token table
ISTYPE	I	I*2	-	Statement type
IREPT	O	L*2	-	Repeat flag, set true if statement is a logical IF
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT5COM, DELCOM, LDTCOM, LUNCOM, MODCOM, OPCOM, STECOM, TYPCOM

3. Subroutines Used: COMPAR, HOPRN, HOPTR1, HOPTR3, LOOKAH, LOOKP, PAGER, POKEP, PRTOKE

4. Subroutines Called by: PRCNTL, PRSTRC

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: PRIMPL

TYPE: Subroutine

PURPOSE: Parses IMPLICIT statements to change the default types for untyped variables.

USAGE:

1. Calling Sequence:

```
CALL PRIMPL (LDTPTR, SYNERR, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Points to next location in delimiter/token table
SYNERR	O	L*2	-	Syntax error flag
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: DELCOM, IMPCOM, LDTCOM, LUNCOM, STECOM, TYPCOM

3. Subroutines Used: LOOKK, LOOKP, INTGR4

4. Subroutines Called by: PRTYPE

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: PRIO

TYPE: Subroutine

PURPOSE: Parses input/output statements.

USAGE:

1. Calling Sequence:

```
CALL PRIO (LDTPTR, ISTYPE, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Points to next location in delimiter/token table
ISTYPE	I	I*2	-	Statement type
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT5COM, DELCOM, LBLCOM, LDTCOM, LUNCOM, MODCOM, OPCOM, STECOM, TYPCOM, XFRCOM

3. Subroutines Used: ADDPOT, COMPAR, ERAPOT, HOPRN, INTGR4, LABLST, LOOKP, LOOKND, PAGER, PRTOKE

4. Subroutines Called by: STATE

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: PRRET

TYPE: Subroutine

PURPOSE: Parses RETURN statements.

USAGE:

1. Calling Sequence:

```
CALL PRRET (LDTPTR, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Points to next location in delimiter/token table
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT5COM, LDTCOM

3. Subroutines Used: None

4. Subroutines Called by: PRCNTL

5. External Data Sets Referenced: None

ROUTINE: PRSPEC

TYPE: Subroutine

PURPOSE: Parses specification statements.

USAGE:

1. Calling Sequence:

```
CALL PRSPEC (LDTPTR, ISTYPE, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Points to next location in delimiter/token table
ISTYPE	I	I*2	-	Statement type
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: DELCOM, LDTCOM, LUNCOM, MODCOM, STECOM, TYPCOM

3. Subroutines Used: FLVARI, LOOKP, PAGER, POKEP

4. Subroutines Called by: STATE

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: PRSTRC

TYPE: Subroutine

PURPOSE: Parses structured FORTRAN statements.

USAGE:

1. Calling Sequence:

```
CALL PRSTRC (ISTYPE, LDTPTR, IREPT, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ISTYPE	I	I*2	-	Statement type
LDTPTR	I	I*2	-	Points to next location in delimiter/token table
IREPT	I/O	L*2	-	Repeat flag, set in routine PRIFS when the statement is a logical IF
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT5COM, LBLCOM, LUNCOM, LDTCOM, OPCOM, TYPCOM

3. Subroutines Used: HOPRN, HOPTR1, HOPTR3, PRDOS, PRIFS, PRTOKE

4. Subroutines Called by: STATE

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: PRSUBS

TYPE: Subroutine

PURPOSE: Parses subprogram statements.

USAGE:

1. Calling Sequence:

```
CALL PRSUBS (LDTPTR, ISTYPE, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Points to next location in delimiter/token table
ISTYPE	I	I*2	-	Statement type
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT5COM, DELCOM, LDTCOM, LUNCOM, MODCOM, STECOM, TYPCOM

3. Subroutines Used: LOOKP, PAGER, POKEP

4. Subroutines Called by: STATE

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: PRTHAL

TYPE: Subroutine

PURPOSE: Prints the complexity analysis on the module statistics summary, if the /MO or /CA control switch is set on.

USAGE:

1. Calling Sequence:

```
CALL PRTHAL (ICTHIO, IETA1, IETA2, LUNMSS, NETA1,
             NETA2, IDECIS)
```

FORTRAN Name	I/O	Type	Dimension	Description
ICTHIO	I	I*2	-	Sum of count of argument variables (including ENTRY arguments) and count of referenced COMMON variables
IETA1	I	I*2	-	Number of unique operators in module
IETA2	I	I*2	-	Number of unique operands in module
LUNMSS	I	I*2	-	LUN for module statistics summary report
NETA1	I	I*2	-	Total number of operators in module
NETA2	I	I*2	-	Total number of operands in module
IDECIS	I	I*2	-	Total number of decisions in module

2. COMMON Blocks Used: WTSCOM

3. Subroutines Used: ESTIM, PAGER

4. Subroutines Called by: MCMPLX

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
7	FOR007.DAT	Write

ROUTINE: PRTOKE

TYPE: Subroutine

PURPOSE: Processes a token to identify it as a variable or a function.

USAGE:

1. Calling Sequence:

```
CALL PRTOKE (LDTPTR, IFUNC, SYNERR, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Pointer to next location in delimiter/token table
IFUNC	O	L*2	-	Switch set true when token is function or arithmetic statement function
SYNERR	O	L*2	-	Switch set true if syntax error encountered
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT5COM, DELCOM, IMPCOM, LDTCOM, LUNCOM, MODCOM, STECOM

3. Subroutines Used: ERRMSG, LOOKP, PAGER, POKEP, OPERAT

4. Subroutines Called by: PRASGN, PRCALL, PRDOS, PRGOTO, PRIFS, PRIO, PRSTRC

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: PRTXFR

TYPE: Subroutine

PURPOSE: Lists the distinct transfer operators and their frequency on the module statistics file when the /HL control switch is set to on.

USAGE:

1. Calling Sequence:

```
CALL PRTXFR
```

2. COMMON Blocks Used: LUNCOM, STECOM, XFRCOM

3. Subroutines Used: LOOKP, PAGER

4. Subroutines Called by: HALREP

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write
7	FOR007.DAT	Write

ROUTINE: PRTYPE

TYPE: Subroutine

PURPOSE: Parses type specification statements and tests for secondary keyword in the case of a typed FUNCTION statement.

USAGE:

1. Calling Sequence:

```
CALL PRTYPE (LDTPTR, ISCLAS, ISTYPE, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I	I*2	-	Pointer to next location in delimiter/token table
ISCLAS	I/O	I*2	-	Statement class
ISTYPE	I/O	I*2	-	Statement type
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: DELCOM, IMPCOM, LDTCOM, MODCOM, STECOM, TYPCOM

3. Subroutines Used: FLVARI, IHASH, KILLP, LOOKK, LOOKP, LOOKS, NUMER, POKES, PRIMPL, PRSUBS, TESTK

4. Subroutines Called by: STATE

5. External Data Sets Referenced: None

ROUTINE: READER

TYPE: Subroutine

PURPOSE: Controls the building of the packed statement string and accumulates statistics on total cards, comment cards, and comment packets.

USAGE:

1. Calling Sequence:

CALL READER (INITR, EXEC1, ENDN, ENDS, ERROR)

FORTRAN Name	I/O	Type	Dimension	Description
INITR	I	L*2	-	Initial read flag, .TRUE. for new file
EXEC1	I	L*2	-	Executable statement flag, .TRUE. after first executable statement
ENDN	O	L*2	-	End of file flag for the initial read
ENDS	O	L*2	-	End of file flag
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CTlCOM, INPCOM, INLCOM, LUNCOM

3. Subroutines Used: GLINE, HSCAN

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: REPHAL

TYPE: Subroutine

PURPOSE: Extracts and reports on data from the data base when the /DB control switch is set on.

USAGE:

1. Calling Sequence:

```
CALL REPHAL (DSNAME, PROJN)
```

FORTRAN Name	I/O	Type	Dimension	Description
DSNAME	I	L*1	70	Data base to be read
PROJN	I	L*1	-	Project identifier used to select modules for inclusion in report

2. COMMON Blocks Used: INFCOM, LUNCOM, MODCOM, PAGCOM

3. Subroutines Used: COEF, ESTIM, PAGER

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write
9	User supplied	Open, read, close
8	FOR008.DAT	Write

ROUTINE: SAPMAIN

TYPE: Main program

PURPOSE: Performs analysis of FORTRAN source code.

USAGE:

1. Calling Sequence: None
2. COMMON Blocks Used: LUNCOM, SWICOM
3. Subroutines Used: CINPUT, COLGLB, DEFINE, HALREP, INITG, INITM, LOADK, MCMPLX, MDIRY, READER, REPHAL, STATG, STATM, STDUMP, TYPE
4. Subroutines Called by: None
5. External Data Sets Referenced:

LUN	File Name	Operation(s)
2	FOR002.DAT	Close
6	FOR006.DAT	Write, close
5	Terminal	Read, write
12	ALL.SAP	Close

ROUTINE: SKPCHR

TYPE: Function

PURPOSE: Locates the first nonoccurrence of a specified character starting at the beginning of a character string.

USAGE:

1. Calling Sequence:

SKPCHR (CHAR, STRING, LENGTH)

FORTRAN Name	I/O	Type	Dimension	Description
SKPCHR	O	I*2	-	= 0, CHAR is the only type of character in STRING ≠ 0, value specifies first byte location in STRING that is not CHAR
CHAR	I	L*1	-	Character to be skipped over
STRING	I	L*1	LENGTH	Character string to be searched
LENGTH	I	I*2	-	Length of character string

2. COMMON Blocks Used: None

3. Subroutines Used: None

4. Subroutines Called by: INPUT

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
5	Terminal	Write

ROUTINE: STATE

TYPE: Subroutine

PURPOSE: Statement processing executive module. All statement processing is performed by the called processing modules.

USAGE:

1. Calling Sequence:

```
CALL STATE (LDTPTR, ISCLAS, ISTYPE, IREPT, LREPT,
            ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I/O	I*2	-	Points to next location in delimiter/token table
ISCLAS	I/O	I*2	-	Statement class
ISTYPE	I/O	I*2	-	Statement type
IREPT	I/O	L*2	-	Repeat flag set .TRUE., after parsing a logical IF statement
LREPT	I/O	L*2	-	Logical flag set .TRUE. if this statement is object of a logical IF statement
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: None

3. Subroutines Used: ERRMSG, PRASGN, PRCNTL, PRIO, PRSPEC, PRSTRC, PRSUBS, PRTYPE

4. Subroutines Called by: TYPE

5. External Data Sets Referenced: None

ROUTINE: STATG

TYPE: Subroutine

PURPOSE: Computes and prints the global statistics when the /GB control switch is set to on.

USAGE:

1. Calling Sequence:

```
CALL STATG
```

2. COMMON Blocks Used: GLBCOM, KEYCOM, LUNCOM, MODCOM, TYPCOM, WTSCOM

3. Subroutines Used: PAGER

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
8	FOR008.DAT	Write

ROUTINE: STATM

TYPE: Subroutine

PURPOSE: Computes and prints the module statistics when the /MO control switch is set to on.

USAGE:

1. Calling Sequence:

```
CALL STATM (INLPAG)
```

FORTRAN Name	I/O	Type	Dimension	Description
INLPAG	O	I*2	-	Page number for module summary produced

2. COMMON Blocks Used: CT1COM, CT2COM, CT3COM, CT4COM, CT5COM, KEYCOM, LUNCOM, MODCOM, OPCOM, SWICOM, TYPCOM

3. Subroutines Used: PAGER, TABLES

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
7	FOR007.DAT	Write

ROUTINE: STDUMP

TYPE: Subroutine

PURPOSE: Produces a formatted listing of the contents of the symbol table.

USAGE:

1. Calling Sequence:

```
CALL STDUMP (LDUMP)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDUMP	I	I*2	-	Logical unit on which to list symbol table

2. COMMON Blocks Used: HSHCOM, STECOM, SYMCOM

3. Subroutines Used: IFASH, LOOKP, PAGER

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: TABCCC

TYPE: Subroutine

PURPOSE: Checks the first six bytes of each source code record for tabs, comment and continuation characters. If a tab is found, the tab character is replaced with a blank. When no tab is found, a tab is inserted in column 6 to facilitate the statement parsing.

USAGE:

1. Calling Sequence:

```
CALL TABCCC (LCOMM, LCONT)
```

FORTRAN Name	I/O	Type	Dimension	Description
LCOMM	O	L*2	-	= .TRUE., if current record is a comment line
LCONT	O	L*2	-	= .TRUE., if current record is a continuation line

2. COMMON Blocks Used: INLCOM, LUNCOM

3. Subroutines Used: None

4. Subroutines Called by: GLINE

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
6	FOR006.DAT	Write

ROUTINE: TABLES

TYPE: Subroutine

PURPOSE: Extracts name and variable usage statistics from the symbol table. The statistics are presented in the module summary report.

USAGE:

1. Calling Sequence:

```
CALL TABLES (ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT2COM, CT5COM, HSHCOM, LBLCOM, STECOM, SYMCOM

3. Subroutines Used: LOOKP

4. Subroutines Called by: STATM

5. External Data Sets Referenced: None

ROUTINE: TESTK

TYPE: Subroutine

PURPOSE: Tests the leading keyword, rehashes any token concatenated to the keyword, and advances the delimiter/token table pointer.

USAGE:

1. Calling Sequence:

```
CALL TESTK (LDTPTR, ISCLAS, ISTYPE, IEXEC, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
LDTPTR	I/O	I*2	-	Delimiter/token table pointer
ISCLAS	O	I*2	-	Statement class identified for this statement
ISTYPE	O	I*2	-	Statement type identified for this statement
IEXEC	O	I*2	-	Executability flag for this statement
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: LDTCOM, STECOM, TYPCOM

3. Subroutines Used: IHASH, KILLP, LOOKK, LOOKP, LOOKS, NUMER, POKES

4. Subroutines Called by: TYPE

5. External Data Sets Referenced: None

ROUTINE: TYPE

TYPE: Subroutine

PURPOSE: Executive control module for statement classification.

USAGE:

1. Calling Sequence:

```
CALL TYPE (EXEC1, ENDM, ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
EXEC1	O	L*2	-	Set .TRUE. after first executable statement has been processed
ENDM	O	L*2	-	Set .TRUE. when an END statement has been encountered at end of module
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: CT3COM, CT4COM, DELCOM, LDTCOM, MODCOM, TYPCOM

3. Subroutines Used: ASGNID, DSCAN, LABEL, STATE, TESTK

4. Subroutines Called by: SAPMAIN

5. External Data Sets Referenced: None

ROUTINE: UCPLX1

TYPE: Subroutine

PURPOSE: A dummy subroutine for which the user may substitute a routine to calculate a complexity measure.

USAGE:

1. Calling Sequence:

```
CALL UCPLX1 (USER1)
```

FORTRAN Name	I/O	Type	Dimension	Description
USER1	O	R*4	-	User complexity

2. COMMON Blocks Used: WTSCOM

3. Subroutines Used: None

4. Subroutines Called by: MCMPLX

5. External Data Sets Referenced: None

ROUTINE: UCPLX2

TYPE: Subroutine

PURPOSE: A dummy subroutine for which the user may substitute a routine to calculate a complexity measure.

USAGE:

1. Calling Sequence:

```
CALL UCPLX2 (USER2)
```

FORTRAN Name	I/O	Type	Dimension	Description
USER2	O	R*4	-	User complexity value

2. COMMON Blocks Used: WTSCOM

3. Subroutines Used: None

4. Subroutines Called by: MCMPLX

5. External Data Sets Referenced: None

ROUTINE: USRWTS

TYPE: Subroutine

PURPOSE: Reads the WEIGHTS.SAP file by default, or reads a user-specified weights file if the /UW control switch is set to on.

USAGE:

1. Calling Sequence:

```
CALL USRWTS (ERROR)
```

FORTRAN Name	I/O	Type	Dimension	Description
ERROR	O	L*2	-	Fatal error flag

2. COMMON Blocks Used: LUNCOM, SWICOM, WTSCOM

3. Subroutines Used: None

4. Subroutines Called by: INITG, LOADK

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
3	WEIGHTS.SAP or User supplied	Open, read, close
5	Terminal	Read
6	FOR006.DAT	Write

ROUTINE: WRTDB

TYPE: Subroutine

PURPOSE: Writes a record to the SAP data base file when the /DB control switch is set to on.

USAGE:

1. Calling Sequence:

```
CALL WRTDB (DBFILE, ICTARG, ICTCBV, ICTCCL, ICTCOM,
            ICTEXC, ICTEXT, ICTHIO, ICTIFF, ICTIO,
            ICTSLN, IDECIS, IETA1, IETA2, LUNCIN,
            LUNDB, MODNAM, NETA1, NETA2, PROJ)
```

FORTRAN Name	I/O	Type	Dimension	Description
DBFILE	I	L*1	70	SAP data base file name
ICTARG	I	I*2	-	Number of arguments passed to module
ICTCBV	I	I*2	-	Number of variables in COMMON blocks
ICTCCL	I	I*2	-	Number of comment lines
ICTCOM	I	I*2	-	Number of COMMON blocks in module
ICTEXC	I	I*2	-	Number of executable statements in module
ICTEXT	I	I*2	-	Number of external references in module
ICTHIO	I	I*2	-	Sum of count of argument variables (including ENTRY arguments) and count of referenced COMMON variables
ICTIFF	I	I*2	-	Number of IF and .IF statements
ICTIO	I	I*2	-	Number of input/output statements
ICTSLN	I	I*2	-	Number of source lines
IDECIS	I	I*2	-	Number of decisions
IETA1	I	I*2	-	Number of unique operators

FORTRAN Name	I/O	Type	Dimension	Description
IETA2	I	I*2	-	Number of unique operands
LUNCIN	I	I*2	-	Command input LUN
LUNDB	I	I*2	-	SAP data base LUN
MODNAM	I	L*1	8	Module name
NETA1	I	I*2	-	Total number of operators
NETA2	I	I*2	-	Total number of operands
PROJ	I	L*1	-	Project character descriptor

2. COMMON Blocks Used: None

3. Subroutines Used: None

4. Subroutines Called by: MCMPLX

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
5	Terminal	Write
9	User supplied	Open, read, write, close

ROUTINE: WRTSEL

TYPE: Subroutine

PURPOSE: Writes a record to ALL.SAP when the /SL control switch is set to on.

USAGE:

1. Calling Sequence:

```
CALL WRTSEL (ICTARG, ICTCBV, ICTCCL, ICTCOM,
             ICTEXC, ICTHIO, ICTIFF, ICTIO, ICTSLN,
             IDECIS, IETA1, IETA2, LUNCIN, LUNSEL,
             MODNAM, NETA1, NETA2, PREFIX, PROJNM,
             ICTCBU, ICTDOS, ICTFNR, ICTSTR, KARGAC,
             KASGN, KCALL, KFMT)
```

FORTRAN Name	I/O	Type	Dimension	Description
ICTARG	I	I*2	-	Number of arguments in module
ICTCBV	I	I*2	-	Number of COMMON block variables
ICTCCL	I	I*2	-	Number of comment lines
ICTCOM	I	I*2	-	Number of COMMON blocks
ICTEXC	I	I*2	-	Number of executable statements
ICTHIO	I	I*2	-	Sum of count of argument variables (including ENTRY arguments) and count of referenced COMMON variables)
ICTIFF	I	I*2	-	Number of IF and .IF statements
ICTIO	I	I*2	-	Number of input/output statements
ICTSLN	I	I*2	-	Number of source lines
IDECIS	I	I*2	-	Number of decisions
IETA1	I	I*2	-	Number of unique operators
IETA2	I	I*2	-	Number of unique operands
LUNCIN	I	I*2	-	Command input LUN
LUNSEL	I	I*2	-	Data base LUN

FORTRAN Name	I/O	Type	Dimen-sion	Description
MODNAM	I	L*1	8	Module name
NETA1	I	I*2	-	Total number of operators
NETA2	I	I*2	-	Total number of operands
PREFIX	I	L*2	-	Prefix descriptor
PROJNM	I	L*1	8	Project name descriptor
ICTCBU	I	I*2	-	Number of COMMON block variables used
ICTDOS	I	I*2	-	Number of DO and DOWHILE statements
ICTFNR	I	I*2	-	Number of function references
ICTSTR	I	I*2	-	Number of structure statements
KARGAC	I	I*2	-	Total number of variables passed to external references
KASGN	I	I*2	-	Number of assignment statements
KCALL	I	I*2	-	Number of CALL statements
KFMT	I	I*2	-	Number of FORMAT statements

2. COMMON Blocks Used: None

3. Subroutines Used: None

4. Subroutines Called by: MCMPLX

5. External Data Sets Referenced:

LUN	File Name	Operation(s)
12	ALL.SAP	Write

SECTION 4 - SAP COMMON BLOCK INFORMATION

Some of the variables used by SAP for communication between modules appear in labeled COMMON blocks. All COMMON blocks are initialized by an associated BLOCK DATA routine except COMMON /INFCOM/. Table 4-1 contains a list of the BLOCK DATA routine file names and the associated COMMON block.

Detailed descriptions of the COMMON block variables used by SAP are presented on the following pages arranged alphabetically by COMMON block name. The variables in each description are listed in the order in which they are stored. The number (if any) enclosed within parenthesis following the variable definition is the value assigned to the variable in the BLOCK DATA routine.

Table 4-1. SAP BLOCK DATA File Names

BLOCK DATA File Name	COMMON Block Name
CT1BLK.FPP	CT1COM
CT2BLK.FPP	CT2COM
CT3BLK.FPP	CT3COM
CT4BLK.FPP	CT4COM
CT5BLK.FPP	CT5COM
DELBLK.FPP	DELCOM
DLIBLK.FPP	DLICOM
GLBBLK.FPP	GLBCOM
HSHBLK.FPP	HSHCOM
IMPBLK.FPP	IMPCOM
(NONE)	INFCOM
INLBLK.FPP	INLCOM
INPBLK.FPP	INPCOM
KEYBLK.FPP	KEYCOM
LBLBLK.FPP	LBLCOM
LDTBLK.FPP	LDTCOM
LUNBLK.FPP	LUNCOM
MODBLK.FPP	MODCOM
OPBLK.FPP	OPCOM
PAGBLK.FPP	PAGCOM
SELBLK.FPP	SELCOM
STEBLK.FPP	STECOM
SWIBLK.FPP	SWICOM
SYMBLK.FPP	SYMCOM
TYPBLK.FPP	TYPCOM
WTSBLK.FPP	WTSCOM
XFRBLK.FPP	XFRCOM

COMMON BLOCK: /CT1COM/

PURPOSE: Contains the module statistics describing module comments.

Variable	Dimension	Type	Definition
MAXCT1		I*2	Number of I*2 words to follow (16)
AVESCD		R*4	Average number of lines of code between comments
AVESCM		R*4	Average number of lines per nonprolog comment packets
ICTSLN		I*2	Sum of all source lines
ICTSCD		I*2	Sum of all coded source lines
ICTCCL		I*2	Sum of all comment card lines (ICTSLN - ICTSCD)
ICTMLC		I*2	Maximum number of lines in code packet
ICTNCD		I*2	Number of code packets
ICTPRO		I*2	Length of prolog
ICTSCM		I*2	Sum of all embedded (nonprolog) comments
ICTSXP		I*2	Sum of comments following a "!" (DEC computers)
ICTMCM		I*2	Maximum size of embedded comment packet
ICTNCM		I*2	Number of embedded comment packets
ICTSBC		I*2	Sum of all blank comment lines
NSINCE		I*2	Number of lines since last comment

COMMON BLOCK: /CT2COM/

PURPOSE: Contains the module statistics describing external communications, variable names, and array dimensions.

Variable	Dimension	Type	Definition
MAXCT2		I*2	Number of I*2 words to follow (25)
IDUMC2		I*2	Dummy alignment variable
AVECHR		R*4	Average number of characters per variable name
AVEDIM		R*4	Average number of dimensions in an array
ICTCHR		I*2	Total number of characters in variable names
MAXCHR		I*2	Length of longest variable name
ICTVAR		I*2	Number of variables in module
ICTFUN		I*2	Number of functions referenced in module
ICTFNR		I*2	Number of function references in module
ICTCON		I*2	Number of constants in module
ICTSUB		I*2	Number of subroutine names referenced in module
ICTENT		I*2	Number of entry point names in module
ICTCOM		I*2	Number of COMMON block names in module
ICTCBV		I*2	Number of variables in COMMON blocks
ICTCBU		I*2	Number of COMMON block variables used
ICTNAM		I*2	Number of NAMELIST names in module
ICTEXT		I*2	Number of external variables in module
ICTEXR		I*2	Number of references to externally defined names
ICTASF		I*2	Number of arithmetic statement function (ASF) names in module
ICTASR		I*2	Number of references to ASFs

Variable	Dimension	Type	Definition
ICTREF		I*2	Number of variables referenced in module
ICTEQV		I*2	Number of variables appearing in EQUIVALENCEs
ICTDIM		I*2	Total number of dimensions of arrays in module
MAXDIM		I*2	Maximum number of dimensions in an array
ICTDMV		I*2	Number of dimensioned variables in module

COMMON BLOCK: /CT3COM/

PURPOSE: Contains the module statistics describing statement breakdown by class and in terms of executable and nonexecutable statements.

Variable	Dimension	Type	Definition
MAXCT3		I*2	Number of I*2 words to follow (45)
IDUMC3		I*2	Dummy for boundary alignment
PCTEXC		R*4	Percent executable statements
PCTNEX		R*4	Percent nonexecutable statements
PCTSTC	13	R*4	Percent statements in each class type
ICTEXC		I*2	Number of executable statements
ICTNEX		I*2	Number of nonexecutable statements
ICTSTC	13	I*2	Number of statements in each class type

COMMON BLOCK: /CT4COM/

PURPOSE: Contains individual statement type counters pertinent to the keywords file. The statements are ordered as in the KEYWORDS.SAP data file.

Variable	Dimension	Type	Definition
MAXCT4		I*2	Number of I*2 variables in COMMON block (65)
IDUMC4		I*2	Boundary alignment space variable
ICTSTT	65	I*2	Array containing counts of statement types, array ordered as in KEYWORDS.SAP

COMMON BLOCK: /CT5COM/

PURPOSE: This COMMON contains the module statistics describing control statements and complexities for subscripted variables.

Variable	Dimension	Type	Definition
MAXCT5		I*2	Number of I*2 variables in COMMON block (50)
IDUMC5		I*2	Boundary alignment space variable
AVECAL		R*4	Average number of arguments in CALL statements
AVEEPA		R*4	Average number of arguments in entry point
AVEFNN		R*4	Average number of functions/ASF in assignments
AVEVRI		R*4	Average number of variables in assignments
AVEOPR		R*4	Average number of operators in assignments
AVEDON		R*4	Average level of nesting in DO loops
AVEDOL		R*4	Average length of DO loops
AVESSC		R*4	Average single statement complexity
ICTIFL		I*2	Number of logical IFs
ICTIFA		I*2	Number of arithmetic IFs
ICTIFG		I*2	Number of GO TOs that are objects of IFs
ICTGUN		I*2	Number of unconditional GO TOs
ICTGAS		I*2	Number of assigned GO TOs
ICTGCM		I*2	Number of computed GO TOs
ICTGCP		I*2	(not used)
ICTGLB		I*2	Number of labels used as targets of GO TOs
ICTERR		I*2	Number of ERR=
ICTEND		I*2	Number of END=
ICTRNN		I*2	Number of normal RETURNs

Variable	Dimension	Type	Definition
ICTRNI		I*2	Number of RETURN Is
ICTCAL		I*2	Number of arguments in all CALL statements
MAXCAL		I*2	Maximum number of arguments in any CALL statement
ICTAMP		I*2	Number of ampersands in CALL statements
ICTEPA		I*2	Number of arguments in all entry points
MAXEPA		I*2	Maximum number of arguments in any entry point
ICTFNN		I*2	Number of functions, ASF in any assignments
MAXFNN		I*2	Maximum number of functions, ASF in any assignment
ICTVRI		I*2	Number of variables in all assignments
MAXVRI		I*2	Maximum number of variables in any assignment
ICTOPR		I*2	Number of operators in all assignments
MAXOPR		I*2	Maximum number of operators in any assignment
ICTARG		I*2	Number of arguments in module calling sequence
ICTDWT		I*2	Number of unconditional downward transfers
ICTUPT		I*2	Number of unconditional upward transfers
ICTDON		I*2	Number of levels of nesting of DO loops
MAXDON		I*2	Maximum level of nesting
ICTDOL		I*2	Number of statements in all DO loops
MAXDOL		I*2	Maximum number of statements in any DO loop
ICTSSV		I*2	Number of references to subscripted variables

Variable	Dimension	Type	Definition
ICTSSC		I*2	Total subscript complexity
MAXSSC		I*2	Maximum subscript complexity
ICTTBR		I*2	Total number of branches
ICTIFB		I*2	IF block counter
ICTEIF		I*2	ELSE IF counter
IFLEV		I*2	Level of IF block
MIFLEV		I*2	Maximum level of IF blocks

COMMON BLOCK: /DELCOM/

PURPOSE: Contains the integer codes for the delimiters contained in the IDELIM array in COMMON /DLICOM/.

Variable	Dimension	Type	Definition
IYCCAT		I*2	Integer code for //
IYEXPO		I*2	Integer code for **
IYMULT		I*2	Integer code for *
IYDIVI		I*2	Integer code for /
IYADDX		I*2	Integer code for +
IYMINU		I*2	Integer code for -
IYEQUA		I*2	Integer code for =
IYOPAR		I*2	Integer code for (
IYCPAR		I*2	Integer code for)
IYCOMA		I*2	Integer code for ,
IYAPOS		I*2	Integer code for '
IYAMPR		I*2	Integer code for &
IYCOLN		I*2	Integer code for :
IYQUOT		I*2	Integer code for "
IYLEFT		I*2	Integer code for <
IYRIGH		I*2	Integer code for >
IYTAB		I*2	Integer code for Tab
IYNULL		I*2	Zero
IYNEXX		I*2	Integer code for .NE.
IYLTXX		I*2	Integer code for .LT.
IYLEXX		I*2	Integer code for .LE.
IYEQXX		I*2	Integer code for .EQ.
IYGEXX		I*2	Integer code for .GE.
IYGTXX		I*2	Integer code for .GT.
IYANDX		I*2	Integer code for .AND.
IYORXX		I*2	Integer code for .OR.
IYXORX		I*2	Integer code for .XOR.
IYEQVX		I*2	Integer code for .EQV.

Variable	Dimension	Type	Definition
IYNOTX		I*2	Integer code for .NOT.
IYNEQV		I*2	Integer code for .NEQV.

COMMON BLOCK: /DLICOM/

PURPOSE: Contains the character representation of valid delimiters and their lengths.

Variable	Dimension	Type	Definition
NDELIM		I*2	Number of delimiters (30)
LDELIM	30	I*2	Array of delimiter lengths
IDELIM	6,30	L*1	Array of delimiters

COMMON BLOCK: /GLBCOM/

PURPOSE: Contains the accumulated global statistics for the input file.

Variable	Dimension	Type	Definition
MAXGLB		I*2	Size of global counter array (100)
MAXSTC		I*2	Size of statement class array (13)
MAXSTT		I*2	Size of statement type arrays (65)
IDUMG		I*2	Dummy alignment variable
AVEGBL	100	R*4	Global averages array
IGTSTC	13	I*2	Global statement class counters
IGTSTT	65	I*2	Global statement type counters
MAXGBL	100	I*2	Global maxima array
NUMGBL	100	I*2	Global counter arrays
IEXGBL	100	I*2	Global counters for auxiliary counts

COMMON BLOCK: /HSHCOM/

PURPOSE: This COMMON contains the pointers to the symbol table entries for the hashed input character string. The hash is computed by the square sum central bit algorithm.

Variable	Dimension	Type	Definition
NHASH		I*2	Size of hash table (1024)
LHMASK		I*2	Mask for hash bits (1777_8)
LHSHFT		I*2	Number of bits to shift hash key (0)
IHTBLE	1024	I*2	Table of pointers to symbol table entries

COMMON BLOCK: /IMPCOM/

PURPOSE: Contains codes used to type variables typed by default or by an IMPLICIT statement.

Variable	Dimension	Type	Definition
IVASC	26	BYTE	ASCII representation of letters A through Z
IVTYP	26	I*2	Assigned data type for letters A through Z
IVBYTE		I*2	Type number for variable type BYTE
IVLOG		I*2	Type number for variable type LOGICAL
IVLOG1		I*2	Type number for variable type LOGICAL*1
IVLOG2		I*2	Type number for variable type LOGICAL*2
IVLOG4		I*2	Type number for variable type LOGICAL*4
IVINT		I*2	Type number for variable type INTEGER
IVINT2		I*2	Type number for variable type INTEGER*2
IVINT4		I*2	Type number for variable type INTEGER*4
IVREA		I*2	Type number for variable type REAL
IVREA4		I*2	Type number for variable type REAL*4
IVREA8		I*2	Type number for variable type REAL*8
IVRE16		I*2	Type number for variable type REAL*16
IVCPX		I*2	Type number for variable type COMPLEX
IVCPX8		I*2	Type number for variable type COMPLEX*8
IVCP16		I*2	Type number for variable type COMPLEX*16
IVDBP		I*2	Type number for variable type DOUBLE PRECISION

Variable	Dimension	Type	Definition
IVDBC		I*2	Type number for variable type DOUBLE COMPLEX
IVCHAR		I*2	Type number for variable type CHARACTER
MASKNU		I*2	Type mask for numeric type variable (8)
MASKCH		I*2	Type mask for character type variable (16)

COMMON BLOCK: /INFCOM/

PURPOSE: Contains the user's command line.

Variable	Dimension	Type	Definition
INF		I*2	Length of INFORM array
INFORM	80	L*1	Command line array

COMMON BLOCK: /INLCOM/

PURPOSE: Contains the two-line rotating input buffer used by SAP while processing the source code input.

Variable	Dimension	Type	Definition
MAXINL		I*2	Size of INLINE array (100)
LASINL		I*2	Last valid character in INLINE (0)
INLPTR		I*2	Current line pointer
INLDUM		I*2	Dummy alignment variable
INLINE	100,2	L*1	Rotating input line buffer

COMMON BLOCK: /INPCOM/

PURPOSE: Contains all the characters in one input source statement. INPUT has the capability to hold up to 19 continuation cards.

Variable	Dimension	Type	Definition
MAXINP		I*2	Size of INPUT character array (1440)
LASINP		I*2	Location of last character in INPUT (0)
INPUT	1440	L*1	Input source statement array

COMMON BLOCK: /KEYCOM/

PURPOSE: Contains information read from the KEYWORDS.SAP file.

Variable	Dimension	Type	Definition
MAXKEY		I*2	Size of keywords array (65)
LASKEY		I*2	Last entry in keywords table (0)
CLASS	65	I*2	Statement class of keyword
EXEC	65	L*2	Statement executability flag: = .TRUE., executable = .FALSE., nonexecutable
LKEY	65	I*2	Keyword length array
KEY	16,65	L*1	Keyword array

COMMON BLOCK: /LBLCOM/

PURPOSE: Contains pointers to a label list array for GO TO statements and DO loop targets

Variable	Dimension	Type	Definition
MAXLBL		I*2	Size of LBLIST, LABLOC, and GOTARG arrays (256)
NEXLBL		I*2	Pointer to next free location in label list (1)
MAXSTK		I*2	Maximum stack depth (size of LBLSTK) (20)
ISTKPT		I*2	Pointer to current top of stack (0)
LBLIST	256	I*4	List of all non-FORMAT labels in module
LABLOC	256	I*2	List of corresponding statement numbers of labeled statements
GOTARG	256	L*1	Set .TRUE. if label is target of a GO TO
LBLSTK	20	I*4	Push down stack for DO loop targets
DOSTAN	20	I*2	Corresponding stack of statement numbers of DO statements

COMMON BLOCK: /LDTCOM/

PURPOSE: Contains the list of pointers to the delimiters and tokens making up the current statement.

Variable	Dimension	Type	Definition
MAXLDT		I*2	Size of LISTDT array (256)
LASLDT		I*2	Location of last entry in LISTDT array
LISTDT	256	I*2	List of delimiter and token pointers

COMMON BLOCK: /LUNCOM/

PURPOSE: Contains the logical unit assignments for SAP.

Variable	Dimension	Type	Definition
LUNKEY		I*2	LUN of keyword file (1)
LUNSOR		I*2	LUN of source input file (2)
LUNWTS		I*2	LUN of weights file (3)
LUNOUT		I*2	Not used
LUNCIN		I*2	LUN of command input unit (5)
LUNLST		I*2	LUN of listings and error message file (6)
LUNMSS		I*2	LUN of module statistics summary file (7)
LUNGSS		I*2	LUN of global statistics summary file (8)
LUNDB		I*2	LUN of data base (9)
LUNDIR		I*2	LUN of indirect file input (10)
LUNSCl		I*2	LUN of INCLUDE file (11)
LUNSEL		I*2	LUN of intermediate Halstead file (12)

COMMON BLOCK: /MODCOM/

PURPOSE: Contains the current module type, name, statement count, and SAP error and warning counts.

Variable	Dimension	Type	Definition
MODTYP		I*2	Module type: = 1, main program (default) = 2, subroutine = 3, function = 4, block data
MODNAM		L*1	Module name (8 characters maximum), (default name = MAIN)
ISN		I*2	Current statement number
NERR		I*2	Number of SAP errors in current module
NWARN		I*2	Number of SAP warnings in current module

COMMON BLOCK: /OPCOM/

PURPOSE: Contains the counts for the operators and operands.

Variable	Dimension	Type	Definition
SUB	50	R*8	List of subroutines, entry points, and functions found so far
NSUB	50	I*2	Use count of each subroutine/ entry/function found
MXSUB		I*2	Maximum number of different subroutine/entry/functions allowed (50)
NDLM	30	I*2	Use count of each delimiter operator
KLOGIF		I*2	Number of logical IF statements
KARTIF		I*2	Number of arithmetic IF statements
KSTIF		I*2	Number of structured IF statements
KELSIF		I*2	Number of ELSE IF statements
KELSE		I*2	Number of ELSE statements
KDO		I*2	Number of DO statements
KDOWH		I*2	Number of DOWHILE statements
KASGN2		I*2	Number of ASSIGN TO statements
KEOS		I*2	Number of end-of-statement (EOS)
IETA1		I*2	Number of unique operators (+, -, ', /, .EQ., .GE., etc.)
IETA2		I*2	Number of unique operands (e.g., variable, constant)
NETA1		I*2	Total number of operators
NETA2		I*2	Total number of operands
IDECIS		I*2	Number of decisions (IF, .IF., DO, DOWHILE, etc.)
NKEYWD		I*2	Number of keyword operators (9)
AKEYWD	9	R*8	Labels for keyword operator report

COMMON BLOCK: /PAGCOM/

PURPOSE: Contains the page count and line counts for each logical unit written by SAP.

Variable	Dimension	Type	Definition
HEAD	5,12	R*8	Page header
LPAGE	12	I*2	Page number (12*0)
LINCNT	12	I*2	Current line counter (12*9999)
MAXLIN	12	I*2	Maximum lines per page per logical unit (12*59)

COMMON BLOCK: /SELCOM/

PURPOSE: Contains the project name and prefix code for the sequential output file (ALL.SAP).

Variable	Dimension	Type	Definition
PROJNM	8	L*1	Project name
PREFIX		L*2	Prefix code of two characters

COMMON BLOCK: /STECOM/

PURPOSE: Contains the current token block from the symbol table.

Variable	Dimension	Type	Definition
MAXTOK		I*2	Maximum size of token block in words (23) = (Maximum Token Length + 1)/2 + 7
NEXT		I*2	Pointer to next block with same key
LAST		I*2	Pointer to previous block with same key
NACTIV		I*2	Activity counter for Halstead operands
ICLASS		I*2	Token class (variable, constant, etc.)
ITYPE		I*2	Token type (subclass)
IUSED		I*2	Symbol utilization count
LTOKE		I*2	Length of token
TOKEN	32	L*1	Token

COMMON BLOCK: /SWICOM/

PURPOSE: Contains the switch variables corresponding to SAP control switches.

Variable	Dimension	Type	Definition
NSWIT		I*2	Number of switches defined
LSWIT	2,20	L*1	Array of two-character control switches
ISWLI		L*2	Output listing switch (F)
ISWGB		L*2	Output global statistics switch (T)
ISWMO		L*2	Output module statistics switch (T)
ISWDU		L*2	Output diagnostic symbol table dump switch (F)
ISWUW		L*2	Accept user weights switch (F)
ISWEC		L*2	Output external communication statistics switch (F)
ISWCO		L*2	Output commenting statistics switch (F)
ISWSC		L*2	Output statement class statistics switch (F)
ISWST		L*2	Output statement type statistics switch (F)
ISWCS		L*2	Output control statement statistics switch (F)
ISWAS		L*2	Output assignment statement statistics switch (F)
ISWSP		L*2	Output specification statement statistics switch (F)
ISWCA		L*2	Output complexity analysis switch (F)
ISWHL		L*2	Print Halstead measures switch (F)
ISWDB		L*2	Write to Halstead data base switch (F)
ISWXP		L*2	Expand INCLUDEs statements switch (F)
ISWSL		L*2	Write to sequential output file switch (F)
ISWXX	3	L*2	Spares

COMMON BLOCK: /SYMCOM/

PURPOSE: Contains the symbol table values and pointers.

Variable	Dimension	Type	Definition
MAXSYM		I*2	Size of symbol table (6000)
NEXSYM		I*2	Next unused symbol table location (1)
IOURFL		I*2	Not used
ISYDUM		I*2	Not used
ISYMBL	6000	L*1	Symbol table

COMMON BLOCK: /TYPCOM/

PURPOSE: Contains pointers to each statement type recognized by SAP.

Variable	Dimension	Type	Definition
IZASFD		I*2	Arithmetic Statement Function Definition
IZASSI		I*2	Assignment Statement
IZACCE		I*2	ACCEPT
IZASGN		I*2	ASSIGN
IZBACK		I*2	BACKSPACE
IZBLOC		I*2	BLOCKDATA
IZBYTE		I*2	BYTE
IZCALL		I*2	CALL
IZCHAR		I*2	CHARACTER
IZCLOS		I*2	CLOSE
IZCOMM		I*2	COMMON
IZCOMP		I*2	COMPLEX
IZCONT		I*2	CONTINUE
IZDATA		I*2	DATA
IZDECO		I*2	DECODE
IZDEFI		I*2	DEFINEFILE
IZDELE		I*2	DELETE
IZDIME		I*2	DIMENSION
IZDOUC		I*2	DOUBLECOMPLEX
IZDOUB		I*2	DOUBLEPRECISION
IZDOWH		I*2	DOWHILE
IZDOXX		I*2	DO
IZELSI		I*2	ELSEIF
IZELSE		I*2	ELSE
IZENCO		I*2	ENCODE
IZENDD		I*2	ENDDO
IZENDF		I*2	ENDFILE
IZENDI		I*2	ENDIF

Variable	Dimension	Type	Definition
IZENDX		I*2	END
IZENTR		I*2	ENTRY
IZEQUI		I*2	EQUIVALENCE
IZEXTR		I*2	EXTERNAL
IZFIND		I*2	FIND
IZFORM		I*2	FORMAT
IZFUNC		I*2	FUNCTION
IZGOTO		I*2	GOTO
IZSTIF		I*2	.IF
IZIFXX		I*2	IF
IZIMPL		I*2	IMPLICIT
IZINCL		I*2	INCLUDE
IZINQU		I*2	INQUIRE
IZINTE		I*2	INTEGER
IZINTR		I*2	INTRINSIC
IZLOGI		I*2	LOGICAL
IZNAME		I*2	NAMELIST
IZOPEN		I*2	OPEN
IZPARA		I*2	PARAMETER
IZPAUS		I*2	PAUSE
IZPRIN		I*2	PRINT
IZPROG		I*2	PROGRAM
IZREAD		I*2	READ
IZREAL		I*2	REAL
IZRETU		I*2	RETURN
IZREWI		I*2	REWIND
IZREWR		I*2	REWRITE
IZSAVE		I*2	SAVE
IZSTOP		I*2	STOP
IZSUBR		I*2	SUBROUTINE
IZTHEN		I*2	THEN

Variable	Dimension	Type	Definition
IZTYPE		I*2	TYPE
IZWRIT		I*2	WRITE
IZBADK		I*2	undecoded
IZUNLO		I*2	UNLOCK
IZVIRT		I*2	VIRTUAL

COMMON BLOCK: /WTSCOM/

PURPOSE: Contains the statistical weights used to compute the SEL complexity.

Variable	Dimension	Type	Definition
MAXWTS		I*2	Number of elements in weight array (256)
IZWTS		I*2	Boundary alignment variable
TOTLWT		R*4	Computed module weight
WEIGHT	256	R*4	Weighting factors for SEL complexity computation

COMMON BLOCK: /XFRCOM/

PURPOSE: Contains the information on module transfer operator analyses.

Variable	Dimension	Type	Definition
LUGOTO		I*2	Pointer to header node of unconditional GO TO list
LCGOTO		I*2	Pointer to header node of computed GO TO list
LAGOTO		I*2	Pointer to header node of assigned GO TO list
LERR		I*2	Pointer to header node of ERR = list
LEND		I*2	Pointer to header node of END = list
LPROC		I*2	Pointer to header node of procedure alternate return list
LXFR	512	I*2	Cells of transfer list
NAVAIL		I*2	Pointer to next available cell
NPOT		I*2	Pointer to first cell of 'potential' node
KPOT		I*2	Pointer to 'length' cell of potential node
LNULL		I*2	Value used for end-of-list (0)
MAXXFR		I*2	Total length of transfer list (512)

SECTION 5 - SAP FILE STRUCTURE

Table 5-1 contains a list of the files used in the SAP system. Files named KEYWORDS.SAP and WEIGHTS.SAP are found in the directories (VAX) DBB1:[TOOLS] and (PDP) DB1:[213,2]. All other files are located within the user's directory. Listings of either the default or sample files are presented in the SAP user's guide (Reference 9) for the keywords, weights, module statistics, global statistics, data base, and sequential output files.

Figure 5-1 shows the relationship between the SAP software and the SAP data files. Each data flow path to a file is labeled with the logical unit name and number. A data flow path which is dependent upon a particular SAP control switch setting (/XX or /-XX) is indicated. Most of the files and processes shown are also labeled with the name of the subroutine (Section 3) that is primarily responsible for the process or file.

Detailed descriptions of each file used by SAP are presented on the following pages. The descriptions are arranged by logical unit number in ascending order (as presented in Table 5-1).

Table 5-1. SAP File Names and Usages

Logical Unit Variable	LUN	I/O	File Name	Use
LUNINN	1	I	FOR001.DAT	Source input containing INCLUDEs
LUNINN	2	I	FOR002.DAT	Included source (level one)
LUNINN	3	I	FOR003.DAT	Included source (level two)
LUNINN	4	I	FOR004.DAT	Included source (level three)
LUNKEY	1	I	KEYWORDS.SAP	Keywords file
LUNSOR	2	I	FOR002.DAT	Source input file
LUNWTS	3	I	WEIGHTS.SAP or User supplied	Weights file
LUNOUT	4		Not used	
LUNCIN	5	I	FOR005.DAT	User terminal
LUNLST	6	O	FOR006.DAT	Error message and source listing file
LUNMSS	7	O	FOR007.DAT	Module statistics file
LUNGSS	8	O	FOR008.DAT	Global statistics file
LUNDB	9	I/O	User supplied	Data base file
LUNDIR	10	I	User supplied	Indirect file
LUNSCI	11	I/O	FOR011.DAT	Scratch file
LUNSEL	12	O	ALL.SAP	Sequential file

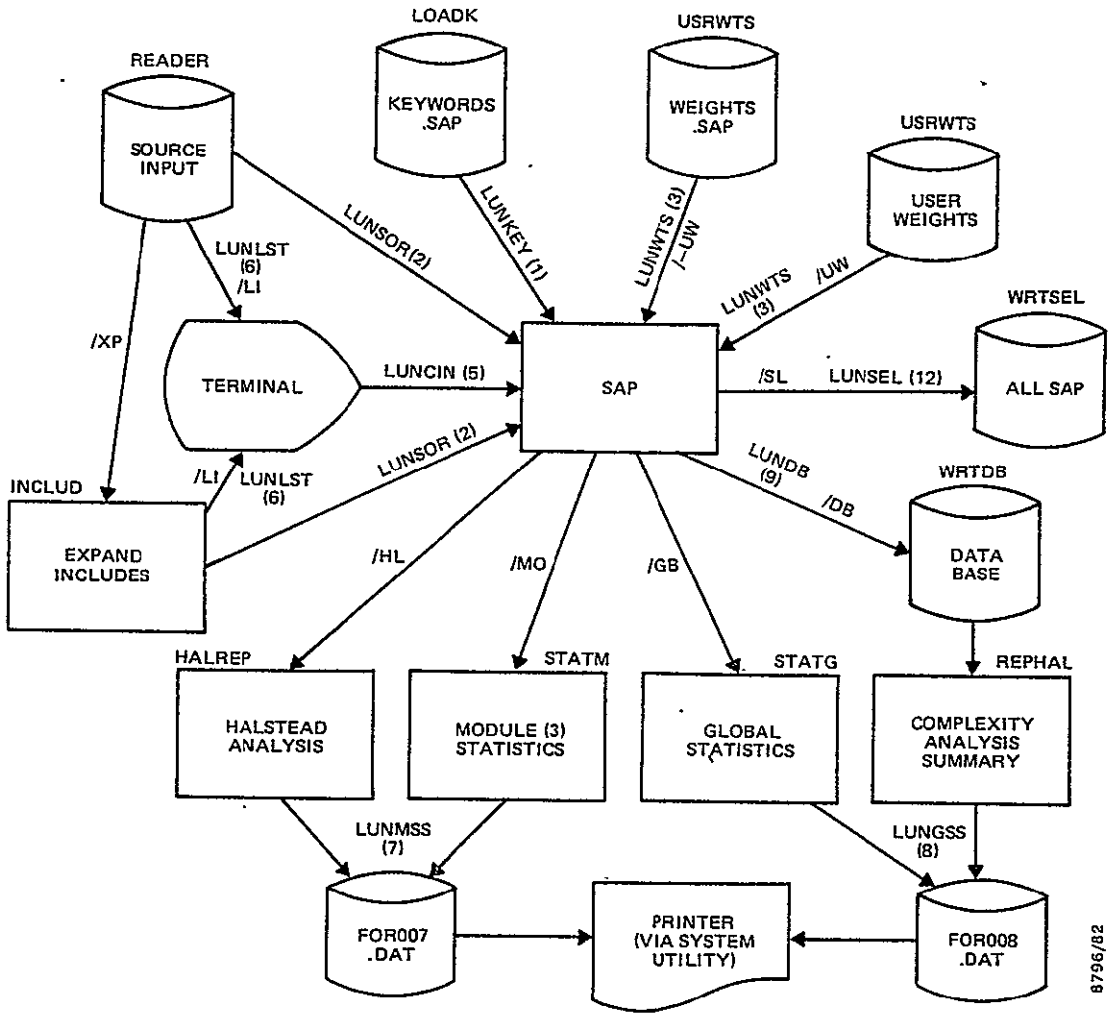


Figure 5-1. SAP Data Flow Diagram

FILE (Logical Unit): FOR001.DAT, FOR002.DAT, FOR003.DAT, FOR004.DAT (LUNINN)

DEVICE/DIRECTORY: User's default

PURPOSE: Internal scratch files to expand INCLUDE statements when the /XP switch is set to on. When an INCLUDE is read, the included file is opened, read, and written to unit FOR011.DAT. The INCLUDE files can be nested to a depth of three INCLUDE statements.

FILE OPERATION BY SUBROUTINE:

Open INCLUD
Close INCLUD
Read INCLUD

FILE LAYOUT:

1. Format: Formatted, variable length
2. Access: Sequential

FILE (Logical Unit): KEYWORDS.SAP (LUNKEY)

DEVICE/DIRECTORY: VAX-11/780 DBB1:[TOOLS]
PDP-11/70 DB1:[213,1]

PURPOSE: Allows flexibility in classifying statements and in marking statements executable or nonexecutable.

FILE OPERATION BY SUBROUTINE:

Open LOADK
Read LOADK
Close LOADK

FILE LAYOUT:

1. Format: Formatted; fixed length
2. Access: Sequential
3. Record Length: 32 bytes
4. Record Description:

Format Code	Byte Position	Contents
L3	1-3	Statement executability flag
I3	4-6	Obsolete
I3	7-9	Obsolete
I3	10-12	Statement class
I3	13-15	Number of characters in the keyword
1X	16	Blank
16A1	17-32	Keyword

FILE (Logical Unit): FOR002.DAT (LUNSOR)

DEVICE/DIRECTORY: User's default

PURPOSE: The source code that is to be processed by SAP is read from this unit. If the /XP switch is set to on to expand INCLUDEs, the input source is read from this file and the expanded source is written to a scratch file and then read. (See the description of file FOR011.DAT.)

FILE OPERATION BY SUBROUTINE:

Open CINPUT
Read GLINE
Close SAPMAIN

FILE LAYOUT:

1. Format: Formatted, variable length
2. Access: Sequential

FILE (Logical Unit): WEIGHTS.SAP (LUNWTS)

DEVICE/DIRECTORY: VAX-11/780 DBB1:[TOOLS]
PDP-11/70 DB1:[213,2]

PURPOSE: Contains a weight or weights to be applied to a particular statistic or range of statistics. If the user specifies a weights file with the /UW switch, that weights file must match the file layout given below.

FILE OPERATION BY SUBROUTINE:

Open USRWTS

Read USRWTS

Close USRWTS

NOTE: These operations apply to both the default and user specified weights files.

FILE LAYOUT:

1. Format: Formatted; fixed length
2. Access: Sequential
3. Record Length: 16 bytes
4. Record Description:

Format Code	Byte Position	Contents
I5	1-5	Lower limit of module statistic number range
I5	6-10	Upper limit of module statistic number range
F6.1	11-16	Statistical weight assigned to all statistics in the specified range

FILE (Logical Unit): FOR005.DAT (LUNCIN)

DEVICE/DIRECTORY: User's default

PURPOSE: Assigned to the user input device. The user's commands are read from this unit.

FILE OPERATION BY SUBROUTINE:

Read INPUT

FILE LAYOUT:

1. Format: Formatted, variable length
2. Access: Sequential

FILE (Logical Unit): FOR006.DAT (LUNLST)

DEVICE/DIRECTORY: User's default

PURPOSE: Displays any error messages encountered during SAP processing. If the /LI switch is set to on, the source code processed by SAP is listed on this unit.

FILE OPERATION BY SUBROUTINE:

The following operation is performed only when the /LI switch is set to on:

Write GLINE

Almost all SAP routines contain code to write error or warning messages to this file. The following operation is performed before SAP is terminated:

Close SAPMAIN

FILE LAYOUT:

1. Format: Formatted, variable length
2. Access: Sequential

FILE (Logical Unit): FOR007.DAT (LUNMSS)

DEVICE/DIRECTORY: User's default

PURPOSE: Module statistics are written to this unit. The statistics are added to this unit as each module is processed. The operator/operand summary is written to this file when the /HL switch is set to on.

FILE OPERATION BY SUBROUTINE:

Write PRTHAL, HALREP, STATG, PRTXFR

FILE LAYOUT:

1. Format: Formatted, variable length
2. Access: Sequential

FILE (Logical Unit): FOR008.DAT (LUNGSS)

DEVICE/DIRECTORY: User's default

PURPOSE: Module directory, global summary, and project summary are written to this file.

FILE OPERATION BY SUBROUTINE:

Write COEF, STATG, MDIRY, REPHAL

FILE LAYOUT:

1. Format: Formatted, variable length
2. Access: Sequential

FILE (Logical Unit): SAP Data base (LUNDB)

DEVICE/DIRECTORY: User's default

PURPOSE: Stores statistical data, when the /DB switch is set to on. The statistics are gathered for each module processed while the /DB switch is on. The correlation summary is produced from the contents of this file.

FILE OPERATION BY SUBROUTINE:

Open DEFINE, WRTDB
Read WRTDB
Write DEFINE, WRTDB
Close DEFINE WRTDB

FILE LAYOUT:

1. Format: Formatted, fixed length
2. Access: Direct
3. Record Length: 80 bytes
4. Record Description: (2 records per module)

Format Code	Header Record Byte Position	Contents
1X	1	Blank
I4	2-5	Maximum records allowed in this file
	6-80	Blank filled

Format Code	First Record Byte Position	Contents
1X	1	Blank
A1	2	Project Identifier
8A1	3-10	Module name
	11-80	Blank-filled

Format Code	Second Record Byte Position	Contents
1X	1	Blank
I3	2-4	Number of arguments passed to the module
I3	5-7	Number of variables in COMMON blocks
I3	8-10	Number of comment lines
I2	11-12	Number of COMMON blocks
I4	13-16	Number of executable statements
I2	17-18	Number of external references (subroutines and functions)
I2	19-20	Number of I/O statements
I4	21-24	Number of source lines
I3	25-27	Number of unique operators
I3	28-30	Number of unique operands
I4	31-34	Total number of operators
I4	35-38	Total number of operands
I3	39-41	Total number of (IF and .IF) statements
I3	42-44	Total number of decisions
I3	45-47	Sum of count of argument variables (including ENTRY arguments) and count of referenced COMMON variables
	48-80	Blank-filled

FILE (Logical Unit): FOR010.DAT (LUNDIR)

DEVICE/DIRECTORY: User's default

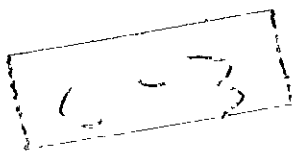
PURPOSE: Gives the user the capability to use an indirect command file as input to SAP.

FILE OPERATION BY SUBROUTINE:

Open INPUT
Read INPUT
Close INPUT

FILE LAYOUT:

1. Format: Formatted, variable length
2. Access: Sequential



FILE (LOGICAL UNIT): FOR011.DAT (LUNSC1)

DEVICE/DIRECTORY: User's default

PURPOSE: The expanded source code is written to this unit when the /XP switch is set to on. The logical unit variable is then redefined as LUNSOR for SAP processing of the current file. The expanded source code is deleted after processing is complete.

FILE OPERATION BY SUBROUTINE:

Open INCLUD
Write INCLUD
Close INCLUD

FILE LAYOUT:

1. Format: Formatted, variable length
2. Access: Sequential

FILE (Logical Unit): ALL.SAP (LUNSEL)

DEVICE/DIRECTORY: User's default

PURPOSE: Stores statistical data to be used by other analysis programs. When the /SL switch is set to on, the file is either created or extended. One record for each module is written to this file while the /SL switch is on.

FILE OPERATION BY SUBROUTINE:

Open DEFSEL, WRTSEL
Write WRTSEL
Close SAPMAIN

FILE LAYOUT:

1. Format: Formatted, fixed length
2. Access: Sequential
3. Record Length: 78 bytes
4. Record Description:

Format Code	Byte Position	Contents
8A1	1-8	Project name
A2	9-10	Project prefix characters
6A1	11-16	Module name
I3	17-19	Number of arguments passed to module
I3	20-22	Number of comment lines in module
I4	23-26	Number of executable statements in module
I2	27-28	Number of I/O statements in module
I4	29-32	Number of source lines in module
I3	33-35	Number of unique operators in module
I3	36-38	Number of unique operands in module
I4	39-42	Total number of operators in module
I4	43-46	Total number of operands in module
I3	47-49	Total number of (IF and .IF) statements in module
I3	50-52	Total number of decisions in module

Format Code	Byte Position	Contents
I3	53-55	Sum of count of argument variables (including ENTRY arguments) and count of referenced COMMON variables
I3	56-58	Number of common block variables used in module
I2	59-60	Total number of DO & DOWHILE statements in module
I3	61-63	Number of function references in module
I3	64-66	Number of structured statements in module
I3	67-69	Number of variables passed to external references in module
I3	70-72	Number of assignment statements in module
I3	73-75	Number of CALL statements in module
I3	76-78	Number of FORMAT statements in module

SECTION 6 - SYSTEM GENERATION

The SAP system can be generated from the source code by executing a few commands. The system generation procedure for the PDP-11/70 is described in Section 6.1, and for the VAX-11/780 in Section 6.2.

6.1 PDP-11/70 SYSTEM GENERATION

To generate the SAP system for the PDP-11/70, only three command procedures need to be executed: GENFPPSAP.CMD, GENSAP.CMD, and SAP.CMD. Figure 6-1 is a listing of the GENFPPSAP.CMD command procedure used to preprocess the structured SAP source code. The OD: preceding each routine name tells the FPP task image where each source code file is located. An assignment, (for example: > ASN OD=DB0:), before executing the GENFPPSAP.CMD is necessary. Two files, LOADK.FPP and USRWTS.FPP, may need to be edited to change the disk (DB1) and UIC ([213,3]) to reflect the disk and UIC in which the keywords and weights files reside. Figure 6-2 is a listing of the GENSAP.CMD command procedure, which compiles the SAP preprocessed source code. Figure 6-3 is a listing of the SAP.CMD command procedure that generates the SAP task image. Figure 6-4 is a listing of the SAP overlay used by the SAP.CMD task build command procedure. The PDP-11/70 SAP system is generated by executing the following commands in the sequence shown:

```
> @GENFPPSAP
> @GENSAP
> @SAP
```

6.2 VAX-11/780 SYSTEM GENERATION

To generate the SAP system for the VAX-11/780, only two command procedures are executed: GENFPPSAP.COM and GENSAP.COM. Figure 6-5 is a listing of the GENFPPSAP.COM command procedure. This command procedure preprocesses the

structured SAP source code. Before executing this command procedure, two routines, LOADK.FPP and USRWTS.FPP, may nee to be edited and the disk (DBB1:) and UIC [TOOLS] assignments changed to reflect the disk and UIC containing the keywords and weights files. Figure 6-6 is a listing of th GENSAP.COM command procedure. This command procedure compiles the source code, generates an object module library the SAP system, and generates the SAP executable task image. The VAX-11/780 SAP system is generated by executin the following commands in the sequence shown:

```
$ @GENFPPSAP
$ @GENSAP
```

```
:  @GENFPPSAP
;
;  THIS COMMAND PROCEDURE WILL PREPROCESS
:  THE SAP FORTRAN ROUTINES
:
:  THERE WILL BE TWO DATA SETS GENERATED PER ROUTINE
:  A *.FLS (LISTING) AN A *.FTN (FORTRAN)
:
:  NOTE: BEFORE EXECUTING THIS COMMAND PROCEDURE
:        THE USER SHOULD EDI ROUTINE LOADK.FPP AND CHANGE
:        THE [UIC] ON THE OPEN STATEMENT FOR THE KEYWORDS.SAP FILE
:        AND ON THE WEIGHTS.SAP FILE IN ROUTINE USRWTS.FPP
;
FPP OD:ADDPUT
FPP OD:ASGNID
FPP OD:CINPUT
FPP OD:CNTXFR
FPP OD:COET
FPP OD:COLGLB
FPP OD:COMPAR
FPP OD:COMPUT
FPP OD:CT1BLK
FPP OD:CT2BLK
FPP OD:CT3BLK
FPP OD:CT4BLK
FPP OD:CT5BLK
FPP OD:DEFINE
FPP OD:DEFSEL
FPP OD:DELBLK
FPP OD:DLIBLK
FPP OD:DSCAN
FPP OD:ERRPOT
FPP OD:ERRMSG
FPP OD:ESTIM
FPP OD:FINDIT
FPP OD:FLYAPI
FPP OD:FNNAME
FPP OD:GARCOL
FPP OD:GLBBLK
FPP OD:GLINE
FPP OD:HALREP
FPP OD:HOPRM
FPP OD:HOPTR1
FPP OD:HOPTR3
FPP OD:HPRMDS
FPP OD:HPR1
FPP OD:HPR2
FPP OD:HPR3
FPP OD:HSCAN
FPP OD:HSHBLK
FPP OD:IHASH
FPP OD:IMPBLK
FPP OD:INCLUD
FPP OD:INITG
FPP OD:INITM
```

Figure 6-1. SAP PDP-11/70 Preprocessing Command Procedure (1 of 3)

```
FPP OD:INITN
FPP OD:INLBLK
FPP OD:INPBLK
FPP UD:INPUT
FPP OD:INTGR4
FPP OD:KEYBLK
FPP OD:KILLP
FPP OD:LABEL
FPP OD:LABLST
FPP OD:LBLBLK
FPP OD:LDTBLK
FPP OD:LNKPOT
FPP OD:LOADK
FPP OD:LOCCHR
FPP OD:LOOKAH
FPP UD:LOOKND
FPP OD:LOOKK
FPP OD:LOOKP
FPP OD:LOOKS
FPP OD:LUNBLK
FPP OD:NCMPLX
FPP OD:NDIRY
FPP OD:MODBLK
FPP OD:NEWPOT
FPP OD:NUMER
FPP OD:OPBLK
FPP OD:OPERAT
FPP OD:PAGBLK
FPP OD:PAGER
FPP OD:POKEP
FPP OD:POKES
FPP OD:PRASGN
FPP OD:PRASS
FPP OD:PRCALL
FPP OD:PPCNTL
FPP OD:PRDOS
FPP OD:PRGOTO
FPP OD:PRIFS
FPP OD:PRIMPL
FPP OD:PRIO
FPP OD:PPRET
FPP OD:PRSPEC
FPP OD:PRSTRC
FPP OD:PRSUBS
FPP OD:PRTHAL
FPP OD:PRTOKE
FPP OD:PRTXFR
FPP OD:PRTYPE
FPP OD:READER
FPP OD:REPHAL
FPP OD:SAPMAIN
FPP OD:SELBLK
FPP OD:SKPCHR
FPP OD:STATE
FPP OD:STATG
```

Figure 6-1. SAP PDP-11/70 Preprocessing Command Procedure (2 of 3)

```
15-JUN-82                GENFPPSAP.CMD                PAGE  3

FPP OD:STATI1
FPP OD:STDUMP
FPP OD:STEBLK
FPP OD:SWIBLK
FPP OD:SYMBLK
FPP OD:TABCCC
FPP OD:TABLES
FPP OD:TESTK
FPP OD:TYPBLK
FPP OD:TYPE
FPP OD:UCPLX1
FPP OD:UCPLX2
FPP OD:USRWTS
FPP OD:WRTDB
FPP OD:WRTSEL
FPP OD:WTSBLK
FPP OD:XFRBLK
```

Figure 6-1. SAP PDP-11/70 Preprocessing Command Procedure (3 of 3)

```
;
:  @GENSAP
;
:  THIS COMMAND PROCEDURE WILL COMPILE
:  THE PREPROCESSED STRUCTURED CODE FOR
:  THE SAP.EXE LOAD MODULE
:
FOR ADDPOT,ADDPOT=ADDPOT
FOR ASGNID.ASGNID=ASGNID
FOR CINPUT.CINPUT=CINPUT
FOR CNTXFR,CNTXFR=CNTXFR
FOR COEF  .COEF  =COEF
FOR COLGLB.COLGLB=COLGLB
FOR COMPAR.COMPAR=COMPAR
FOR COMPWT.COMPWT=COMPWT
FOR CT1BLK,CT1BLK=CT1BLK
FOR CT2BLK,CT2BLK=CT2BLK
FOR CT3BLK.CT3BLK=CT3BLK
FOR CT4BLK.CT4BLK=CT4BLK
FOR CT5BLK.CT5BLK=CT5BLK
FOR DEFINE.DEFINE=DEFINE
FOR DEFSEL,DEFSEL=DEFSEL
FOR DELBLK.DELBLK=DELBLK
FOR DLIBLK.DLIBLK=DLIBLK
FOR DSCAN .DSCAN =DSCAN
FOR ERAPOT.ERAPOT=ERAPOT
FOR ERRMSG.ERRMSG=ERRMSG
FOR ESTIM .ESTIM =ESTIM
FOR FINDIT.FINDIT=FINDIT
FOR FLVARI.FLVARI=FLVARI
FOR FNNAME.FNNAME=FNNAME
FOR GARCOL,GARCOL=GARCOL
FOR GLBBLK.GLBBLK=GLBBLK
FOR GLINE .GLINE =GLINE
FOR HALREP,HALREP=HALREP
FOR HOPRN .HOPRN =HOPRN
FOR HOPTR1,HOPTR1=HOPTR1
FOR HOPTR3.HOPTR3=HOPTR3
FOR HPR1  .HPR1  =HPR1
FOR HPR2  .HPR2  =HPR2
FOR HPR3  .HPR3  =HPR3
FOR HPRNDS,HPRNDS=HPRNDS
FOR HSCAN .HSCAN =HSCAN
FOR HSHBLK,HSHBLK=HSHBLK
FOR IHASH .IHASH =IHASH
FOR IMPBLK.IMPBLK=IMPBLK
FOR INCLUD.INCLUD=INCLUD
FOR INITG .INITG =INITG
FOR INITM .INITM =INITM
FOR INITN .INITN =INITN
FOR INLBLK,INLBLK=INLBLK
FOR INPBLK.INPBLK=INPBLK
FOR INPUT .INPUT =INPUT
FOR INTGR4.INTGR4=INTGR4
FOR KEYBLK,KEYBLK=KEYBLK
```

Figure 6-2. SAP PDP-11/70 FORTRAN Compilation Command Procedure (1 of 3)

```
FOR KILLP .KILLP =KILLP
FOR LABEL .LABEL =LABEL
FOR LABLST,LABLST=LABLST
FOR LBLBLK,LBLBLK=LBLBLK
FOR LDTBLK.LDTBLK=LDTBLK
FOR LNKPOT,LNKPOT=LNKPOT
FOR LOADK ,LOADK =LOADK
FOR LOCCHR,LOCCHR=LOCCHR
FOR LOOKAH,LOOKAH=LOOKAH
FOR LOOKND,LOOKND=LOOKND
FOR LOOKK ,LOOKK =LOOKK
FOR LOOKP ,LOOKP =LOOKP
FOR LOOKS .LOOKS =LOOKS
FOR LUNBLK,LUNBLK=LUNBLK
FOR MCMPLX.MCMPLX=MCMPLX
FOR MDIRY ,MDIRY =MDIRY
FOR MODBLK.MODBLK=MODBLK
FOR NEWPOT.NEWPOT=NEWPOT
FOR NUMER .NUMER =NUMER
FOR OPBLK .OPBLK =OPBLK
FOR OPERAT.OPERAT=OPERAT
FOR PAGBLK.PAGBLK=PAGBLK
FOR PAGER .PAGER =PAGER
FOR POKEP .POKEP =POKEP
FOR POKES .POKES =POKES
FOR PRASGN.PRASGN=PRASGN
FOR PRASS .PRASS =PRASS
FOR PRCALL.PRCALL=PRCALL
FOR PRCNTL,PRCNTL=PRCNTL
FOR PRDOS .PRDOS =PRDOS
FOR PRGOTO.PRGOTO=PRGOTO
FOR PRIFS .PRIFS =PRIFS
FOR PRIMPL,PRIMPL=PRIMPL
FOR PRIO  .PRIO  =PRIO
FOR PRRET ,PRRET =PRRET
FOR PRSPEC,PRSPEC=PRSPEC
FOR PRSTRC.PRSTRC=PRSTRC
FOR PRSUBS.PRSUBS=PRSUBS
FOR PRTHAL,PRTHAL=PRTHAL
FOR PRTOKE,PRTOKE=PRTOKE
FOR PRTXFR.PRTXFR=PRTXFR
FOR PRTYPE,PRTYPE=PRTYPE
FOR READER.READER=READER
FOR REPHAL.REPHAL=REPHAL
FOR SAPMAIN.SAPMAIN=SAPMAIN
FOR SELBLK.SELBLK=SELBLK
FOR SKPCHR.SKPCHR=SKPCHR
FOR STATE .STATE =STATE
FOR STATG .STATG =STATG
FOR STATM .STATM =STATM
FOR STDUMP.STDUMP=STDUMP
FOR STEBLK,STEBLK=STEBLK
FOR SWIBLK.SWIBLK=SWIBLK
FOR SYMBLK.SYMBLK=SYMBLK
FOR TABCCC.TABCCC=TABCCC
```

Figure 6-2. SAP PDP-11/70 FORTRAN Compilation Command Procedure (2 of 3)

```
- 15-JUN-82                GENSAP.CMD                PAGE   3

   FOR TABLES,TABLES=TABLES
   FOR TESTK .TESTK =TESTK
   FOR TYPBLK.TYPBLK=TYPBLK
   FOR TYPE  .TYPE  =TYPE
   FOR UCPLX1.UCPLX1=UCPLX1
   FOR UCPLX2,UCPLX2=UCPLX2
   FOR USRWTS,USRWTS=USRWTS
   FOR WRTDB .WRTDB -WRTDB
   FOR WRTSEL.WRTSEL=WPTSEL
   FOR WTSBLK,WTSBLK=WTSBLK
   FOR XFRBLK.XFRBLK=XFRBLK
```

Figure 6-2. SAP PDP-11/70 FORTRAN Compilation Command Procedure (3 of 3)

```
15-JUN-82                    SAP.CMD                          PAGE    1

 : COMMAND FILE TO BUILD SAP TASK
 :
 SAP.SAP/SH/-SP=SAP/MP
 ACTFIL=6
 UNITS=12
 ASG=TI:5:6
 ASG=SY:1:2:3:4:7:8
 //
```

Figure 6-3. SAP PDP-11/70 Task Building Command Procedure

```
15-JUN-82                    SAP.ODL                          PAGE    1

;
; SAP (V2) OVERLAY
;
;  NOTE: READER AND GLINE CANNOT BE OVERLAID
;
        .ROOT ROOT-*(AA1,AA2,AA3,AA4,AA5,AA6,AA7)
ROOT:   .FCTR SAPMAIN-COMPAR-IHASH-LOOKP-ERAPOT-ADDPOT-READER-R1
R1:     .FCTR PAGER-POKEP-LOOKAH-LOOKND-GLINE-HSCAN-TABCCC-ESTIM-R3
R3:     .FCTR CT1BLK-CT2BLK-CT3BLK-CT4BLK-R4
R4:     .FCTR CT5BLK-DELBLK-DLIBLK-GLBBLK-R5
R5:     .FCTR HSHBLK-INLBLK-INFBLK-KEYBLK-R6
R6:     .FCTR LBLBLK-LDTBLK-LUNBLK-MODBLK-R7
R7:     .FCTR OPBLK-PAGBLK-STEBLK-SWIBLK-R8
R8:     .FCTR SYMBLK-TYPBLK-WTSBLK-IMPBLK-XFRBLK-SELBLK
AA1:    .FCTR  LOADK-CINPUT-(INCLUD-USRWTS-LOCCHR-INPUT-A1
A1:     .FCTR  SKPCHR-FNNAME-DEFSEL-DEFINE-FINDIT)
AA2:    .FCTR  INITG-INITM-INITN-NEWPOT-LNKPOT
AA3:    .FCTR  DSCAN-GARCOL-HOPRN-HOPTR1-B1
B1:     .FCTR  HOPTR3-INTGR4-KILLP-LABEL-B2
B2:     .FCTR  LABLST-LOOKK-LOOKS-NUMER-ERRMSG-B3
B3:     .FCTR  POKES-ASGNID-STATE-TYPE-B4
B4:     .FCTR  OPERAT-PRTOKE-TESTK-(CC1,CC2)
CC1:    .FCTR   PRASGN-PRCNTL-PRIO-PRSTRC-C2
C2:     .FCTR   (PRGOTO,PRRET,PRIFS,PRDOS,PRASS,PRCALL)
CC2:    .FCTR   FLVARI-PRSPEC-PRSUBS-PRTYPE-PRIMPL
AA4:    .FCTR  STATM-TABLES-STDUMP
AA5:    .FCTR  MCMPLX-CNTXFR-UCPLX1-UCPLX2-A4
A4:     .FCTR  HPRNDS-HPR1-HPR2-HPR3-COMPUT-A5
A5:     .FCTR  (PRTHAL-WRTDB-WRTSEL)
AA6:    .FCTR  COLGLB-STATG-HALREP-MDIRY-PRTXFR
AA7:    .FCTR  REPHAL-(COEF)
        .END
```

Figure 6-4. SAP PDP-11/70 Overlay Description

```
$ SET VERIFY
$ !  @GENFPPSAP
$ !
$ !  THIS COMMAND PROCEDURE WILL PREPROCESS
$ !  THE SAP FORTRAN ROUTINES
$ !
$ !  THERE WILL BE TWO DATA SETS GENERATED PER ROUTINE
$ !  A *.FLS (LISTING) AN A *.FTN (FORTRAN)
$ !
$ !  NOTE: BEFORE EXECUTING THIS COMMAND PROCEDURE
$ !        THE USER SHOULD EDT ROUTINE LOADK.FPP AND CHANGE DISK AND
$ !        UIC ON THE OPEN STATEMENT FOR THE KEYWORDS.SAP FILE
$ !        AND ON THE WEIGHTS.SAP FILE IN ROUTINE USRWTS.FPP
$ !
$ !
$ RUN FPP
ADDPOT
ASGNID
CINPUT
CNTXFR
COEF
COLGLB
COMPAR
COMPWT
CT1BLK
CT2BLK
CT3BLK
CT4BLK
CT5BLK
DEFINE
DEFSEL
DELBLK
DUIBLK
DSCAN
ERAPOT
ERRMSG
ESTIM
FINDIT
FLVARI
FNNAME
GARCOL
GLBBLK
GLINE
HALREP
HOPRN
HOPTR1
HOPTR3
HPRNDS
HPR1
HPR2
```

Figure 6-5. SAP VAX-11/780 Preprocessing Command Procedure (1 of 3)

```
HPR3
HSCAN
HSHBLK
IHASH
IMPBLK
INCLUD
INITG
INITM
INITN
INLBLK
INPBLK
INPUT
INTGR4
KEYBLK
KILLP
LABEL
LABLST
LBLBLK
LDTBLK
LNKPOT
LOADK
LOCCHR
LOOKAH
LOOKK
LOOKND
LOOKP
LOOKS
LUNBLK
MCMPLX
MDIRY
MODBLK
NEWPOT
NUMER
OPBLK
OPERAT
PAGBLK
PAGER
POKEP
POKES
PRASGN
PRASS
PRCALL
PRCNTL
PRDOS
PRGOTO
PRIFS
PRIMPL
PRIO
PRRET
PRSPEC
```

Figure 6-5. SAP VAX-11/780 Preprocessing Command Procedure (2 of 3)

```
PRSTRC
PRSUBS
PRTHAL
PRTOKE
PRTXFR
PRTYPE
READER
REPHAL
SAPMAIN
SELBLK
SKPCHR
STATE
STATG
STATM
STDUMP
STEBLK
SWIBLK
SYMBLK
TABCCC
TABLES
TESTK
TYPBLK
TYPE
UCPLX1
UCPLX2
USRWTS
WRTDB
WRTSEL
WTSBLK
XFRBLK
```

Figure 6-5. SAP VAX-11/780 Preprocessing Command Procedure (3 of 3)

```
$ SET VERIFY
$ !
$ !  @GENSAP
$ !
$ !  THIS COMMAND PROCEDURE WILL COMPILE AND LINK
$ !  THE SAP.EXE LOAD MODULE
$ !
$ FOR/NOI4 ADDPOT.FTN
$ FOR/NOI4 ASGNID.FTN
$ FOR/NOI4 CINPUT.FTN
$ FOR/NOI4 CNTXFR.FTN
$ FOR/NOI4 COEF.FTN
$ FOR/NOI4 COLGLB.FTN
$ FOR/NOI4 COMPAR.FTN
$ FOR/NOI4 COMPWT.FTN
$ FOR/NOI4 CT1BLK.FTN
$ FOR/NOI4 CT2BLK.FTN
$ FOR/NOI4 CT3BLK.FTN
$ FOR/NOI4 CT4BLK.FTN
$ FOR/NOI4 CT5BLK.FTN
$ FOR/NOI4 DEFINE.FTN
$ FOR/NOI4 DEFSEL.FTN
$ FOR/NOI4 DELBLK.FTN
$ FOR/NOI4 DLIBLK.FTN
$ FOR/NOI4 DSCAN.FTN
$ FOR/NOI4 ERAPOT.FTN
$ FOR/NOI4 ERRMSG.FTN
$ FOR/NOI4 ESTIM.FTN
$ FOR/NOI4 FINDIT.FTN
$ FOR/NOI4 FNNAME.FTN
$ FOR/NOI4 FLVARI.FTN
$ FOR/NOI4 GARCOL.FTN
$ FOR/NOI4 GLBBLK.FTN
$ FOR/NOI4 GLINE.FTN
$ FOR/NOI4 HALREP.FTN
$ FOR/NOI4 HOPRN.FTN
$ FOR/NOI4 HOPTR1.FTN
$ FOR/NOI4 HOPTR3.FTN
$ FOR/NOI4 HPRNDS.FTN
$ FOR/NOI4 HPR1.FTN
$ FOR/NOI4 HPR2.FTN
$ FOR/NOI4 HPR3.FTN
$ FOR/NOI4 HSCAN.FTN
$ FOR/NOI4 HSHBLK.FTN
$ FOR/NOI4 IHASH.FTN
$ FOR/NOI4 IMPBLK.FTN
$ FOR/NOI4 INCLUD.FTN
$ FOR/NOI4 INITG.FTN
$ FOR/NOI4 INITM.FTN
$ FOR/NOI4 INITN.FTN
```

Figure 6-6. SAP VAX-11/780 FORTRAN Compilation and Linking Command Procedure (1 of 3)

```
$ FOR/NOI4 INLBLK.FTN
$ FOR/NOI4 INPBLK.FTN
$ FOR/NOI4 INPUT.FTN
$ FOR/NOI4 INTGR4.FTN
$ FOR/NOI4 KEYBLK.FTN
$ FOR/NOI4 KILLP.FTN
$ FOR/NOI4 LABEL.FTN
$ FOR/NOI4 LABLST.FTN
$ FOR/NOI4 LBLBLK.FTN
$ FOR/NOI4 LDTBLK.FTN
$ FOR/NOI4 LNKPOT.FTN
$ FOR/NOI4 LOADK.FTN
$ FOR/NOI4 LOCCHR.FTN
$ FOR/NOI4 LOOKAH.FTN
$ FOR/NOI4 LOOKK.FTN
$ FOR/NOI4 LOOKND.FTN
$ FOR/NOI4 LOOKP.FTN
$ FOR/NOI4 LOOKS.FTN
$ FOR/NOI4 LUNBLK.FTN
$ FOR/NOI4 MCMPLX.FTN
$ FOR/NOI4 MDIRY.FTN
$ FOR/NOI4 MODBLK.FTN
$ FOR/NOI4 NEWPOT.FTN
$ FOR/NOI4 NUMER.FTN
$ FOR/NOI4 OPBLK.FTN
$ FOR/NOI4 OPERAT.FTN
$ FOR/NOI4 PAGBLK.FTN
$ FOR/NOI4 PAGER.FTN
$ FOR/NOI4 POKEP.FTN
$ FOR/NOI4 POKES.FTN
$ FOR/NOI4 PRASGN.FTN
$ FOR/NOI4 PRASS.FTN
$ FOR/NOI4 PRCALL.FTN
$ FOR/NOI4 PRCNTL.FTN
$ FOR/NOI4 PRDOS.FTN
$ FOR/NOI4 PRGOTO.FTN
$ FOR/NOI4 PRIFS.FTN
$ FOR/NOI4 PRIMPL.FTN
$ FOR/NOI4 PRIO.FTN
$ FOR/NOI4 PRRET.FTN
$ FOR/NOI4 PRSPEC.FTN
$ FOR/NOI4 PRSTRC.FTN
$ FOR/NOI4 PRSUBS.FTN
$ FOR/NOI4 PRTHAL.FTN
$ FOR/NOI4 PRTOKE.FTN
$ FOR/NOI4 PRTXFR.FTN
$ FOR/NOI4 PRTYPE.FTN
$ FOR/NOI4 READER.FTN
$ FOR/NOI4 REPHAL.FTN
$ FOR/NOI4 SAPMAIN.FTN
```

Figure 6-6. SAP VAX-11/780 FORTRAN Compilation and Linking Command Procedure (2 of 3)

```
$ FOR/NOI4 SELBLK.FTN
$ FOR/NOI4 SKPCHR.FTN
$ FOR/NOI4 STATE.FTN
$ FOR/NOI4 STATG.FTN
$ FOR/NOI4 STATM.FTN
$ FOR/NOI4 STDUMP.FTN
$ FOR/NOI4 STEBLK.FTN
$ FOR/NOI4 SWIBLK.FTN
$ FOR/NOI4 SYMBLK.FTN
$ FOR/NOI4 TABCCC.FTN
$ FOR/NOI4 TABLES.FTN
$ FOR/NOI4 TESTK.FTN
$ FOR/NOI4 TYPBLK.FTN
$ FOR/NOI4 TYPE.FTN
$ FOR/NOI4 UCPLX1.FTN
$ FOR/NOI4 UCPLX2.FTN
$ FOR/NOI4 USRWTS.FTN
$ FOR/NOI4 WRTDB.FTN
$ FOR/NOI4 WRTSEL.FTN
$ FOR/NOI4 WTSBLK.FTN
$ FOR/NOI4 XFRBLK.FTN
$ !
$ !
$ !  GENERATE THE LOAD MODULE
$ !
$ LIBRARY/CREATE SAP
$ LIBRARY/INSERT SAP ADDPOT,ASGNID,CINPUT,CNTXFR,COEF,COLGLB,COMPAR
$ LIBRARY/INSERT SAP COMPWT,CT1BLK,CT2BLK,CT3BLK,CT4BLK,CT5BLK,DEFINE
$ LIBRARY/INSERT SAP DEFSEL,DELBLK,DLIBLK,DSCAN,ERAPOT,ERRMSG,ESTIM
$ LIBRARY/INSERT SAP FINDIT,FLVARI,FNNAME,GARCOL,GLBBLK,GLINE,HALREP
$ LIBRARY/INSERT SAP HOPRN,HOPTR1,HOPTR3,HPRNDS,HPR1,HPR2,HPR3,HSCAN
$ LIBRARY/INSERT SAP HSHBLK,IHASH,IMPBLK,INCLUD,INITG,INITM,INITN
$ LIBRARY/INSERT SAP INLBLK,INPBLK,INPUT,INTGR4,KEYBLK,KILLP,LABEL
$ LIBRARY/INSERT SAP LABLST,LBLBLK,LDTBLK,LNKPOT,LOADK,LOCCHR,LOOKAH
$ LIBRARY/INSERT SAP LOOKK,LOOKND,LOOKP,LOOKS,LUNBLK,MCMPLX,MDIRY
$ LIBRARY/INSERT SAP MODBLK,NEWPOT,NUMER,OPBLK,OPERAT,PAGBLK,PAGER
$ LIBRARY/INSERT SAP POKEP,POKES,PRASGN,PRASS,PRCALL,PRCNTL,PRDOS
$ LIBRARY/INSERT SAP PRGOTO,PRIFS,PRIMPL,PRIO,PRRET,PRSPEC,PRSTRC
$ LIBRARY/INSERT SAP PRSUBS,PRTHAL,PRTOKE,PRTXFR,READER,REPHAL
$ LIBRARY/INSERT SAP SAPMAIN,SKPCHR,STATE,STATG,STATM,STDUMP
$ LIBRARY/INSERT SAP STFBLK,SWIBLK,SYMBLK,TABCCC,TABLES,TESTK,TYPBLK
$ LIBRARY/INSERT SAP TYPE,UCPLX1,UCPLX2,USRWTS,WRTDB,WRTSEL
$ LIBRARY/INSERT SAP WTSBLK,XFRBLK,SELBLK
$ !
$ LINK/EXEC=SAP SAPMAIN,SAP/LIBRARY/INCLUDE=(CT1BLK,-
CT2BLK,CT3BLK,CT4BLK,CT5BLK,DELBLK,DLIBLK,GLBBLK,HSHBLK,-
IMPBLK,INLBLK,INPBLK,KEYBLK,LBLBLK,LDTBLK,LUNBLK,MODBLK,-
OPBLK,PAGBLK,SELBLK,STEBLK,SWIBLK,SYMBLK,TYPBLK,WTSBLK,XFRBLK)
$ !
$ !
```

Figure 6-6. SAP VAX/11/780 FORTRAN Compilation and Linking Command Procedure (3 of 3)

SECTION 7 - MOVING SAP TO ANOTHER COMPUTER

The entire SAP system is available on distribution tapes created for either the PDP-11/70 or VAX-11/780 computers. Programmers installing SAP on these computers are referred to the first file on the distribution tape, the installation guide, for an explanation of the tape contents and instructions for generating the executable program.

The following discussion is directed to programmers who wish to install SAP on a machine other than the DEC PDP-11/70 or the DEC VAX-11/780.

Moving SAP to another model of a DEC computer that has a FORTRAN compiler available is a straight-forward operation. The system generation procedures described in Section 6 will require major modification only if the operating system is not RSX-11M for a PDP-11 model or VMS for a VAX-11 model.

When planning the installation of SAP on a non-DEC computer, three areas should be considered: reading the distribution tape, compatability of SAP data structures with the target computer's word size, and the language extensions used in the SAP source code. These areas are discussed in the following sections.

7.1 THE SAP DISTRIBUTION TAPES

The SAP distribution tape is available for either the PDP-11/70 or the VAX-11/780. Each tape consists of text files that include an installation guide, command procedures to compile and link the source code (on the respective computer), source code, and required data files. There are no binary files on these tapes.

The SAP distribution tape is a 9-track, 1600 bit-per-inch, ASCII, unlabeled tape. The tape is written by the DEC FLX utility (Reference 12).

The distribution tape also contains either the PDP-11/70 or VAX-11/780 distribution tape files for the structured FORTRAN preprocessor (SFORT) (References 8 and 13) since SAP is written in structured FORTRAN. This document does not, however, discuss SFORT, except to note that the discussion in Sections 7.2 and 7.3 also applies to that program.

7.2 SAP DEPENDENCE UPON COMPUTER WORD SIZE

SAP is written with an implicit assumption of running on a computer with a 16-bit integer word size and addressability to the (8-bit) byte level.

Most mathematical calculations performed by SAP use either 16-bit integers or 32-bit floating point variables. In some instances, integer variables have been declared to be 32 bits in length because their value frequently exceeds 32767.

Character manipulation within SAP is performed with LOGICAL*1 (or BYTE) variables, each of which contains one character. The structure of the SAP software that examines source code is based upon the ability to manipulate a single character at a time. Integer variables are equivalenced to LOGICAL*1 or BYTE arrays containing character data to permit efficient transfer of this data as a block; however, no character manipulation or mathematical calculations are performed with these integers.

Reference 3, Appendix A, presents a description of the internal representation of integer, floating point, and byte data types on the PDP computers. Reference 5, Appendix A, presents similar information for the VAX computers. It should be noted that both computers require 16-bit and 32-bit variables to be aligned with 16-bit word addresses. Other computers may have more stringent requirements for variables appearing in COMMON or EQUIVALENCE statements.

7.3 ENVIRONMENTAL CONSIDERATIONS

The environment in which SAP operates has features that may not be available at other installations. This section discusses the features most likely to be unavailable.

SAP references four routines supplied by DEC as support for FORTRAN systems. These routines are shown in Table 7.1, along with references to the appropriate documentation.

DEC file naming conventions are discussed in References 3 and 5. In some instances in SAP, the file name extension of '.DAT' is appended to the file name if an extension is not supplied by the user.

The symbol table used by SAP (Section 2.2.1) contains variables in which individual bits are set and read. Setting and reading these bits is accomplished with the nonstandard use of the logical operators .OR. and .AND., respectively.

Other nonstandard FORTRAN usage is presented in Table 7.2. An explanation of the SFORT constructs (.IF-ELSE-ENDIF and DOWHILE-ENDDO) is given in Reference 8.

The PDP-11/70 version of SAP is overlayed to execute within 65K bytes of memory. The PDP task builder manual (Reference 14) and the SAP overlay description (Figure 6-4) can be used as a starting point in designing an overlay for other installations with memory restrictions.

Table 7-1. System Routines Used by SAP

System Routine	Reference 3 (PDP)	Reference 4 (VAX)
ERRSET	Section D.6	*
ISHFT	Section 4.1	Section C.3
DATE	Section D.4	Section C.4.1
TIME	Section D.16	Section C.4.6

*The VAX implementation of ERRSET is discussed in Reference 5, Section D.3.3

Table 7-2. Language Extensions Used in SAP

Language Extension	Reference 2 (PDP)	Reference 4 (VAX)
ENCODE Statement	Section 7.6	Section A.1
DECODE Statement	Section 7.6	Section A.1
INCLUDE Statement	Section 1.5	Section 1.5
D-Lines Debug Feature	Section 1.3.3.2	Section 1.3.3.2
OPEN Statement Keywords		
TYPE	Section 9.1.20	Section 9.1.25
RECORDSIZE	Section 9.1.17	Section 9.1.21
MAXREC	Section 9.1.12	Section 9.1.15
NAME	Section 9.1.13	Section 9.1.16
READONLY	Section 9.1.16	Section 9.1.19
Direct Access Record Number Specifier	Section 7.4	Section 7.2.1.4
Octal Constants	Section 2.3.1	Section 2.3.7
FORMAT Edit Descriptors		
Q	Section 8.1.12	Section 8.1.20
$	Section 8.1.13	Section 8.1.21
<n>	Section 8.2	Section 8.1.26
Type Specifications	Section 2.2	Section 2.2
BYTE		
LOGICAL*1		
LOGICAL*2		
INTEGER*2		
INTEGER*4		
REAL*4		
REAL*8		

REFERENCES

1. American National Standards Institute, ANSI X3.9-1978, American National Standard Programming Language FORTRAN, April 1978

2. Digital Equipment Corporation, AA-1855D-TC, PDP-11 FORTRAN Language Reference Manual, December 1979

3. --, AA-1884C-TC, FORTRAN IV-PLUS User's Guide, December 1979

4. --, AA-D034B-TE, VAX-11 FORTRAN Language Reference Manual, April 1980

5. --, AA-D035B-TE, VAX-11 FORTRAN User's Guide, April 1980

6. International Business Machines Corporation, SC28-6852, IBM OS FORTRAN IV (H Extended) Compiler Programmer's Guide, November 1974

7. -- GC28-6515, IBM System /360 and System /370 FORTRAN IV Language, May 1974

8. Software Engineering Laboratory, SEL-77-003, Structured FORTRAN Preprocessor (SFORT), B. Chu, D. S. Wilson, and R. Beard, September 1977

9. --, SEL-78-102, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 1), W. J. Decker and W. A. Taylor, September 1982

10. --, SEL-78-001, FORTRAN Static Source Code Analyzer (SAP) Design and Module Descriptions, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, January 1978

11. M. Halstead, Elements of Software Science. New York: Elsevier Publishing Co., 1977

12. Digital Equipment Corporation, AA-5567B-TC, RSX-11 Utilities Procedures Manual, December 1977

13. Software Engineering Laboratory, SEL-78-004, Structured FORTRAN Preprocessor (SFORT) PDP-11/70 User's Guide, D. S. Wilson, B. Chu, and G. Page, September 1978

14. Digital Equipment Corporation, AA-H266A-TC, RSX-11M/M-PLUS Task Builder Manual, June 1979

BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-Originated Documents

Software Engineering Laboratory, SEL-76-001, Proceedings From the First Summer Software Engineering Workshop, August 1976

SEL-77-001, The Software Engineering Laboratory, V. R. Basili, M. V. Zelkowitz, F. E. McGarry, et al., May 1977

SEL-77-002, Proceedings From the Second Summer Software Engineering Workshop, September 1977

SEL-77-003, Structured FORTRAN Preprocessor (SFORT), B. Chu, D. S. Wilson, and R. Beard, September 1977

SEL-77-004, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-001, FORTRAN Static Source Code Analyzer (SAP) Design and Module Descriptions, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, January 1978

†SEL-78-002, FORTRAN Static Source Code Analyzer (SAP) User's Guide, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

SEL-78-102, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 1), W. J. Decker and W. A. Taylor, May 1982 (preliminary)

SEL-78-003, Evaluation of Draper NAVPAK Software Design, K. Tasaki and F. E. McGarry, June 1978

†This document superseded by revised document.

SEL-78-004, Structured FORTRAN Preprocessor (SFORT) PDP-11/70 User's Guide, D. S. Wilson, B. Chu, and G. Page, September 1978

SEL-78-005, Proceedings From the Third Summer Software Engineering Workshop, September 1978

SEL-78-006, GSFC Software Engineering Research Requirements Analysis Study, P. A. Scheffer, November 1978

SEL-78-007, Applicability of the Rayleigh Curve to the SEL Environment, T. E. Mapp, December 1978

SEL-79-001, SIMPL-D Data Base Reference Manual, M. V. Zelkowitz, July 1979

SEL-79-002, The Software Engineering Laboratory: Relationship Equations, K. Freburger and V. R. Basili, May 1979

SEL-79-003, Common Software Module Repository (CSMR) System Description and User's Guide, C. E. Goorevich, S. R. Waligora, and A. L. Green, August 1979

SEL-79-004, Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment, C. E. Goorevich, A. L. Green, and F. E. McGarry, September 1979

SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

SEL-80-001, Functional Requirements/Specifications for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, C. E. Goorevich, and A. L. Green, February 1980

SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker, C. E. Goorevich, and A. L. Green, May 1980

SEL-80-003, Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study, T. Welden, M. McClellan, P. Liebertz, et al., May 1980

SEL-80-004, System Description and User's Guide for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, W. J. Decker, J. G. Garrahan, et al., October 1980

SEL-80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook and F. E. McGarry, December 1980

SEL-81-001, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., September 1981

SEL-81-002, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide, D. C. Wyckoff, G. Page, F. E. McGarry, et al., September 1981

SEL-81-003, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, D. N. Card, D. C. Wyckoff, G. Page, et al., September 1981

†SEL-81-004, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., September 1981

SEL-81-104, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

†SEL-81-005, Standard Approach to Software Development, V. E. Church, F. E. McGarry, G. Page, et al., September 1981

SEL-81-105, Recommended Approach to Software Development, S. Eslinger, F. E. McGarry, V. E. Church, et al., May 1982

SEL-81-006, Software Engineering Laboratory (SEL) Document Library (DOCLIB) System Description and User's Guide, W. Taylor and W. J. Decker, December 1981

†SEL-81-007, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, E. J. Smith, A. L. Green, et al., February 1981

SEL-81-107, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, E. J. Smith, W. A. Taylor, et al., February 1982

SEL-81-008, Cost and Reliability Estimation Models (CAREM) User's Guide, J. F. Cook and E. Edwards, February 1981

†This document superseded by revised document.

SEL-81-009, Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation, W. J. Decker, A. L. Green, and F. E. McGarry, March 1981

SEL-81-010, Performance and Evaluation of an Independent Software Verification and Integration Process, G. Page and F. E. McGarry, May 1981

SEL-81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

SEL-81-012, The Rayleigh Curve As a Model for Effort Distribution Over the Life of Medium Scale Software Systems, G. O. Picasso, December 1981 (also published as University of Maryland Technical Report TR-1186, July 1982)

SEL-81-013, Proceedings From the Sixth Annual Software Engineering Workshop, December 1981

SEL-81-014, Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-82-001, Evaluation and Application of Software Development Measures, D. N. Card, G. Page, and F. E. McGarry, July 1982

SEL-82-002, FORTRAN Static Source Code Analyzer Program (SAP) System Description, W. Taylor and W. Decker, August 1982

SEL-82-003, Software Engineering Laboratory (SEL) Data Base Reporting Software User's Guide and System Description, P. Lo and S. Eslinger, September 1982

SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982

SEL-Related Literature

Anderson, L., "SEL Library Software User's Guide," Computer Sciences-Technicolor Associates, Technical Memorandum, June 1980

††Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Banks, F. K., "Configuration Analysis Tool (CAT) Design," Computer Sciences Corporation, Technical Memorandum, March 1980

††Basili, V. R., "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

††Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

Basili, V. R., "SEL Relationships for Programming Measurement and Estimation," University of Maryland, Technical Memorandum, October 1980

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: Computer Societies Press, 1980 (also designated SEL-80-008)

††Basili, V. R., and J. Beane, "Can the Parr Curve Help with Manpower Distribution and Resource Estimation Problems?", Journal of Systems and Software, February 1981, vol. 2, no. 1

††Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

††Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

Basili, V. R., and T. Phillips, "Validating Metrics on Project Data," University of Maryland, Technical Memorandum, December 1981

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity and Cost, October 1979

---

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

††Basili, V. R., and M. V. Zelkowitz, "Operation of the Soft ware Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

††Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: Computer Societies Press, 1978

Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982

††Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis to Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Church, V. E., "User's Guides for SEL PDP-11/70 Programs," Computer Sciences Corporation, Technical Memorandum, March 1980

Freburger, K., "A Model of the Software Life Cycle" (paper prepared for the University of Maryland, December 1978)

Higher Order Software, Inc., TR-9, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977 (also designated SEL-77-005)

Hislop, G., "Some Tests of Halstead Measures" (paper prepared for the University of Maryland, December 1978)

---

††This article also appears in SEL-82-004, Collected Softwar Engineering Papers: Volume 1, July 1982

Lange, S. F., "A Child's Garden of Complexity Measures" (paper prepared for the University of Maryland, December 1978)

Miller, A. M., "A Survey of Several Reliability Models" (paper prepared for the University of Maryland, December 1978)

National Aeronautics and Space Administration (NASA), NASA Software Research Technology Workshop (proceedings), March 1980

Page, G., "Software Engineering Course Evaluation," Computer Sciences Corporation, Technical Memorandum, December 1977

Parr, F., and D. Weiss, "Concepts Used in the Change Report Form," NASA, Goddard Space Flight Center, Technical Memorandum, May 1978

Perricone, B. T., "Relationships Between Computer Software and Associated Errors: Empirical Investigation" (paper prepared for the University of Maryland, December 1981)

Reiter, R. W., "The Nature, Organization, Measurement, and Management of Software Complexity" (paper prepared for the University of Maryland, December 1976)

Scheffer, P. A., and C. E. Velez, "GSFC NAVPAK Design Higher Order Languages Study: Addendum," Martin Marietta Corporation, Technical Memorandum, September 1977

Turner, C., G. Caron, and G. Brement, "NASA/SEL Data Compendium," Data and Analysis Center for Software, Special Publication, April 1981

Turner, C., and G. Caron, "A Comparison of RADC and NASA/SEL Software Development Data," Data and Analysis Center for Software, Special Publication, May 1981

Weiss, D. M., "Error and Change Analysis," Naval Research Laboratory, Technical Memorandum, December 1977

Williamson, I. M., "Resource Model Testing and Information," Naval Research Laboratory, Technical Memorandum, July 1979

††Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York: Computer Societies Press, 1979

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977