Chapter 4

The AIPS Program

4.1 Overview

The AIPS program is the portion of the AIPS system with which the user normally interacts. The major functions of the AIPS program are: (1) prepare the parameters for and initiate the tasks which do most of the computations, (2) allow interactive use of TV and graphics devices, (3) provide limited direct analysis capability and (4) provide a high level of control logic to allow simple functions to be grouped into more complex functions (i.e., a programming language).

The basis of the AIPS program is the POPS (People Oriented Parsing Service) language processor. POPS is an interpretive language processor which can either accept statements for immediate execution or in the form of programs, called procedures, which are compiled and stored for later execution. Operations on data, images etc. are performed by means of "verbs" and "tasks". Verbs are operations which are done directly by the AIPS program and tasks are programs which are run asynchronously from AIPS. Both verbs and tasks are controlled by a set of global parameters called "adverbs". Verbs may change the values of adverbs whereas tasks cannot.

This chapter will attempt to describe the basic methods of the POPS processor and explain how to add new verbs and adverbs. The AIPS program does not know directly about tasks, so adding tasks requires no modifications to the AIPS program.

Other documentation about POPS processors may be found in a report by Jerome A. Hudson entitled "POPS People-Oriented Parsing Service Language Description and Program Documentation" and *POPS An Interactive Terminal Language with Applications in Radio Astronomy* by A. Sume, 1978, Internal Report no. 115, Research Laboratory of Electronics and Onsala Space Observatory, Chalmers University of Technology, Gothenburg, Sweden.

4.2 Structure of the AIPS Program

The basis of the AIPS program is a POPS processor which interprets user instructions and calls the relevant applications routines and spawns the desired tasks. Input to the POPS processor is in the form of statements which may do one of the following:

1. Modify an adverb value. This may be either by specifying a literal constant or an arithmetic, logical or character string expression.

2. Invoke an applications verb. These are the verbs which are specific to a given data analysis problem, such as displaying an image on the TV, rather than general control verbs such as loop control or sine functions etc.

3. Logic flow control. These statements control the execution of other statements, e.g., loop control, IF, THEN, ELSE etc.

4. Spawn tasks. Tasks are programs which take relatively long times to run and are executed asynchronously from AIPS. Communication between AIPS and tasks is primarily by disk files.

5. Prepare and edit procedures. POPS programs called procedures may be entered and compiled for later execution. These procedures may later be edited.

6. Prepare batch files. AIPS can run in a batch mode. To do this, the user enters and/or edits a list of commands in a batch file for later execution. This can be done either in the normal AIPS or a special batch version of AIPS named BATER.

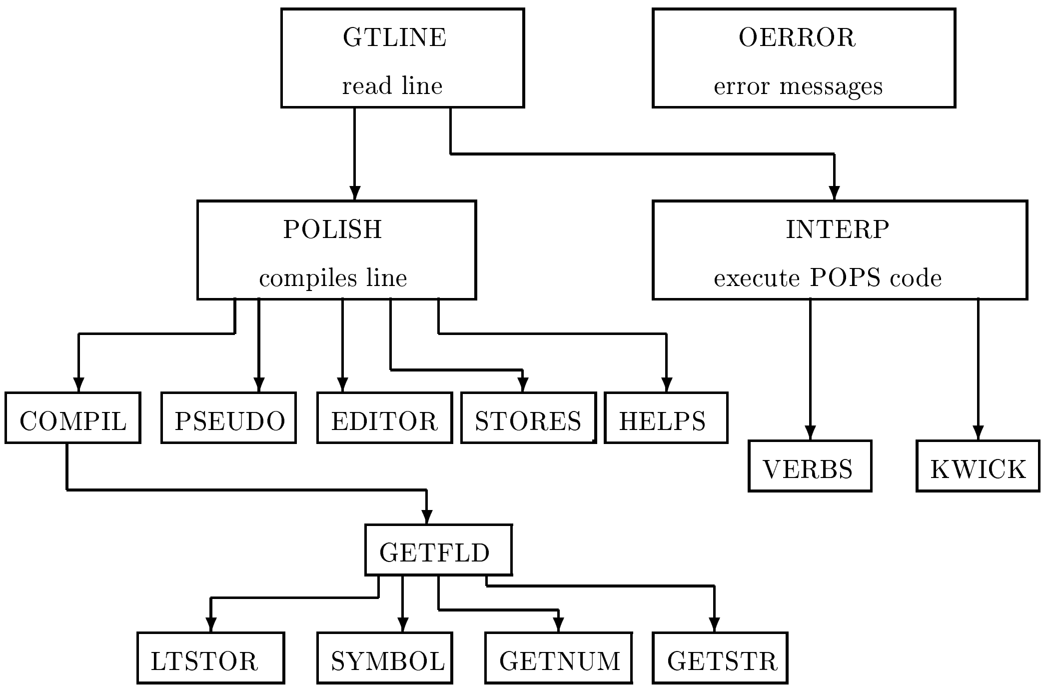
7. Store the current environment including all current procedures via the STORE or SAVE command. This environment is restored via the RESTORE or GET command.

4.2.1 The POPS processor

POPS uses an "inverse POLISH" stack to store operands and operation codes. Symbolics such as verb, adverb or procedure names are stored in a symbol table and each is identified by a type (TYPE) and a number (TAG). The initial entries in the symbol table and initial values of the adverbs are read from an external disk file which is prepared by the stand alone utility routine POPSGN. The various tables and stack pointers etc. are carried in common and the tables are equivalenced into an array known as the "K array".

Multiple statements, separated by semicolons, may be entered in a single line. There are a number of special verbs known as "pseudo" verbs which are executed as soon as they are encountered, causing any other instructions on the same line to be parsed in special fashions, ignored, or handled normally depending on the pseudoverb.

The basic structure of the AIPS program is very hierarchal. The main routine calls a startup routine, AIPBEG, a shutdown and error routine, AIPERR and a single routine GTLINE which controls the bulk of the processing. The structure of the basic routines in the POPS processor is shown in the following figure:



Structure of POPS

More details of each of these routines is given in the following:

- *GTLINE* is the main POPS routine. It causes lines to be read by PREAD, parsed and compiled or executed (in the case of pseudo verbs) by POLISH, and finally executed by INTERP. GTLINE returns only on error or requested termination of the program.
- *OERROR* displays an error message on the user terminal and resets POPS.
- *INTERP* causes POPS code to be executed by placing operands on the V and STACK stacks and calling VERBS and KWICK for verbs.
- *VERBS* calls the relevant applications verb routines based on the verb number. Functions are grouped together in routines named AUn. The appropriate routine is called with a branch code as an argument. This branch code in the verb number minus the first verb number in that AU routine plus one. The verb numbers are defined in an external file but VERBS must also know which verb numbers correspond to which AU routine.
- *KWICK* executes the basic POPS control verbs. These are the verbs which don't depend particularly on a given application but are frequently encountered.
- *POLISH* parses the character string entered by the user and translates it to Polish postfix notation. The result is a string of integers representing code for the POPS interpreter. Negative tokens are operand pointers while positive tokens are operator codes. The array A, which is equivalenced to STACK, holds the list of tokens; AP points to the most recent entry and SP points to the next entry. The operand pointers are to the location of the adverb or temporary variable in the K array.
- *COMPIL* does the actual interpretation of instructions and adds them to the stacks. COMPIL exits when a pseudo-verb or end-of-line is encountered.
- *PSEUDO* handles procedure and adverb declarations, sets up for the runtime operators IF, THEN, ELSE, WHILE (which require forward references and an additional cleanup pass) and the FINISH operator.
- *EDITOR* performs the operations required to begin and stop editing an existing procedure.
- *STORES* stores either the procedure source code, procedure object code, or handles the procedure source code.
- *HELPS* handles the user assistance facilities HELP, INPUT, EXPLAIN and RUN and other functions which require access to external text files. HELP lists symbols by type or lists a text file whose member name matches a user name. RUN sets the input to a specified member of a text file. This allows users to have personal strings of commands (e.g., procs, verbs, adverb settings). INPUTS lists the adverbs and their current values and brief descriptions on the terminal. Subroutine HELPS simply parses the user input in a more friendly fashion and places appropriate verb numbers and strings on the stacks.
- *GETFLD* finds the next non-blank character in the input buffer, KARBUF, and determines whether the token begun with that character is symbolic (1st char is A–Z), numeric (1st char is 0–9 or .), or hollerith (1st char is '). After the field length is found, appropriate calls are made to the symbol processing routine, number scanning routine, etc. Communication back to POLISH is via TYPE and TAG parameters determined by the processors SYMBOL, GETNUM, LTSTOR...
- *LTSTOR* searches the list of literals in the K array. If a matching literal is found, the TAG is returned. If not, a new one is generated and linked to the literal list. Note: a "literal" is a constant having either a numeric, character, or logical value.
- *SYMBOL* finds a symbol in the symbol list. The result is returned as TYPE and TAG through a common. If the routine is in the variable declaration mode, a new entry will be made in the symbol table if it does not already exist.
- *GETNUM* converts a character string into a DOUBLE PRECISION value.
- *GETSTR* obtains a character string from a buffer.

4.2.2 POPS commons

Most of the communication between POPS subroutines is by means of commons. As with most commons in the AIPS system, these commons are obtained by use of include files. The contents and uses of these commons are described in the following. The text of the include files is given at the end of this chapter.

DCON.INC

This common contains the basic POPS "memory" or K array, i.e., the symbol tables, adverb values, procedures etc. This common consists of equivalenced, INTEGER (K), REAL (C) and HOLLERITH (CH) arrays. Included in the latter part of this array are the adverb values. The variables used for the installed (predefined) adverbs are declared in the include DAPL.INC and follow a shortened declaration of the K array in DCON.INC. They specify the adverbs as equivalences to the K array beginning at K(KXORG+10).

User defined adverbs as well as as procedures and temporary literal values are stored beginning at K(301). The names of all symbolics (adverbs, verbs and procedures) are kept in a symbol table which is a linked list of symbol names containing the symbol type (TYPE), location in the K array (TAG) and the location of the array or string descriptor entries if appropriate. The first entry in the symbol table is pointed to by K(1) and a zero link indicates the last entry in the table. More details are given in later sections.

Literals (constants) are kept in a literal table which is also a linked list in the K array. The first entry is pointed to by K(4) and the last entry is pointed to by K(10). The literal table entry contains the type, length, and value of the literal.

The current compiled version of procedures is also kept in the K array. Each procedure may be divided into several blocks in the K array; the blocks are connected by forward links. A pointer is kept to the first location of the source version of the procedure in the LISTF array kept in the working memory file (kept on disk). The first block of a procedure is pointed to by the symbol table.

The different portions of the K array are used as follows:

```
K(1)     Symbol table link, points to first entry in the symbol table.
K(2)     Program link, points to first program (Procedure)
K(3)     Next free cell in K array to be allocated.
K(4)     Constants (literal) link, points to first entry in the
         literal table.
K(5)     Number of cells allocatable. Currently 14760.
K(6)     KTEMP, pointer to KKT (temporary value) area.
K(7)     Symbol protect limit.  Names with TAGs greater than this
         value may be changed.  This is used to protect
         procedures compiled by POPSGN.
K(8)     KXORG, pointer to KX array (data area). Currently 14761.

K(9)     Last symbol pointer.
K(10)    Last literal pointer
K(11-50) Not used

                    KKT area, temporary storage for MODE=0

K(51)    Not used
K(52)    Program link
K(53)    Next free cell
K(54)    Constants link
K(55)    Number of cells allocatable
K(56-59) not used
K(60)    Last constant pointer.

K(301...) Used for program storage, constants, symbols etc. for the
          remainder of the program position of the K array.
```

```
                     KX area, data storage

K(KXORG+0)    Not used
K(KXORG+1)    not used
K(KXORG+2)    Next free cell
K(KXORG+4)    Number of cells allocatable
K(KXORG+5)    not used
K(KXORG+6)    Highest adverb address in K not changeable by user.
K(KXORG+7->+9) not used
K(KXORG+10...) data storage.

                     Symbol table entries.

Word 1:   Link to next symbol table entry. Zero if end of list.
     2:   bits 2**0 to 2**3 = type.
          bits 2**4 to 2**15 = number of words in symbol
     3:   TAG (location in core where the data is kept)
     4:   Array data block counter if symbol is an array name,
          string, or procedure.
     5:   Bytes 1 and 2 of the name.
     6:   Bytes 3 and 4 of symbol name.
     7:   etc.

                      Array data blocks, define arrays.
                        (pointed to by symbol table)

Word 1:   Total array size
     2:   Number of dimensions
     3:   Initial index for first index
     4:   first dimension
     5:   Initial index for second dimension
     6:   etc.

                         Strings and string arrays
                        (pointed to by symbol table)

Word 1:  Total array size
     2:  Number of dimensions
     3:  1
     4:  no. floating point words in each element.
     5:  initial index for first subscript, if any
     6:  first subscript range, if any
     7:  etc.

                    Literal table entries

Word 1:   Pointer to next literal table entry, zero if last entry.
     2:   Bits 2**0 to 2**3 = type.  the types are 11=>floating point
               real (2 integer words), 14=>character string, 15=>
               logical constant (TRUE or FALSE)
          Bits 2**4 to 2**15 length of literal in integers.
     3:   First integer word in literal.
     4:   etc.
```

```
                  Procedure storage (compiled code)
                    (pointed to by symbol table)

Word 1:  Link to next program block, zero if last.
     2:  Pointer to text array for purposes of listing.
     3:  first interpreter instruction.
     4:  etc.
         ...
     N:   1  An opcode of 1 terminates a block.  If the link to the
         next block is zero the procedure terminates.
```

DPOP.INC

The common in this include carries the various stacks, stack pointers and other values. The contents of this common are described in the following:

```
 V(60)       R  Operand stack for REAL variables.
 XX          R  Intermediate REAL value
 KT          I  Starting location in K array of KKT (temporary) area.
 LPGM        I  Start address of an entry in the K array.  Used while
                allocating storage.
 LLIT        I  Not used
 LAST        I  Last token (opcode); if zero, finished with line.
                Used by COMPIL.
 IDEBUG      I  A debug flag used in various places.  If true
                (.GE.0) then debug info about POPS is given.
 MODE        I  The current mode of the POPS processor.
                   0 => immediate execution of an input line
                   1 => compile a procedure
                   2 => finishing a procedure
                   3 => editing a procedure
                   69 => adding a new symbol to symbol table
 IFFLAG      I  = 1 if an operator has been found in the current
                instruction; 0 otherwise.
 LINK        I  A link (pointer in K array)
 L           I  Another link (pointer in K array)
 NAMEP       I  Pointer in K array to a name in the symbol table.
 IP          I  Pointer in K array
 LP          I  Pointer in K array
 SLIM        I  Maximum allowed index in the stacks (currently 60)
 AP          I  Pointer to last entry in STACK
 BP          I  Pointer to last entry in CSTACK
 ONE         I  Pointer in C to value of 1.0
 ZERO        I  Pointer in C to value of 0.0
 TRUE        I  Pointer in C to value .TRUE.
 FALSE       I  Pointer in C to value .FALSE.
 STACK(60)   I  Instruction stack
 CSTACK(60)  I  Second (temporary) instruction stack
 SP          I  Pointer in STACK
 CP          I  Pointer in CSTACK
 SP0         I  Another pointer in STACK
 MPAGE       I  Number of pages (512 bytes) in the Memory file.
```

```
                 (LISTF + K array)
LPAGE       I  Number of pages (512 bytes) of the memory file
                 which contain LISTF (procedure source code)
```

DSMS.INC

This common contain various important values passed between routines. The contents of this common are described in the following.

```
KPAK(5)     H  Temporary array for storing a symbol name.
NKAR        I  The number of characters in KPAK
KBPTR       I  A character pointer in KARBUF, the input line buffer.
NEWCOD      I  Tag given by SYMBOL when allocating space for a new
               adverb.
TYPE        I  Symbol type.  See section on TAG and TYPE.
SKEL        R  Not used.
TAG         I  Symbol number.  See section on TAG and TYPE.
LEVEL       I  Precedence level bias.
LX          I  Number of integer words in character string X.
NEXTP       I  Precedence level of next item on A-stack.
X(25)       R  Temporary storage for character strings.
LOCSYM      I  Location in symbol or literal table of current
               symbol.
```

DIO.INC

This common contains short I/O buffers and related information. The contents of this common are described in the following.

```
IPT    C*1   Prompt character
NBYTES I     Number of valid characters in KARBUF, number of last
             non-blank character.
KARBUF C*80  A buffer containing the current input line.
JBUFF  C*80  Buffer used to read user input.
KARLIM I     Number of characters in KARBUF
IUNIT  I     Input unit number for PREAD; 1=> user terminal, 2=>
             text editor, 3=>batch input file 4=>text entered
             during screen hold.
HOLDUF C*80  Buffer for storing text entered during screen hold by
             SCHOLD.
```

4.2.3 TAG and TYPE

Adverbs, verbs, procedures etc. are all represented by symbolic names to the user. Internally, POPS identifies symbolics by TYPE and TAG. TYPE determines the type of symbolic (e.g., scalar, character string, verb etc.) and TAG is a label for the particular symbolic (e.g., a verb number). The TYPE of all symbols and the TAG of verbs are specified to POPSGN in the POPSDAT.HLP file. The TAG of an adverb is computed by POPS and is the start address of the value field.

The current list of symbolic types is given in the following list.

```
TYPE = 1   REAL scalar.
       2   REAL array.
```

```
 3   Procedure name.
 4   Verb name
 5   Pseudo verb name.
 6   Quit (used by POPSGN)
 7   Character string
 8   Element of character string
 9   substring of a character string
10   not used
11   Numeric constant
14   Character constant
15   Logical constant.
```

4.2.4 Error Handling

If a subroutine determines that an error condition exists, it sets the variable ERRNUM in INCLUDE DERR.INC to an error code known to the routine OERROR, increments ERRLEV in DERR.INC, and, if ERRLEV .LE. 5, copies the name of the subroutine into DERR.INC array PNAME. Following this, the subroutine returns. Thus, after each call to another AIPS subroutine, a subroutine should check ERRNUM and, if it is not zero, then that subroutine should increment ERRLEV, add its name to PNAME and exit. If GTLINE determines that an error has occurred, it returns to to the main AIPS routine which calls AIPERR which calls OERROR. This provides a traceback capability which can be exercised setting the AIPS adverb DEBUG to 1.0.

The messages displayed to the user corresponding to the defined values of ERRNUM are shown in the following:

```
ERRNUM      Message          ERRNUM      Message
   1      'BLEW CORE!  '       33      'CTLG PROBLEM'
   2      'SYMBOL?     '       34      'HISTORY FILE'
   3      'BAD ( OR )  '       35      'FIT FAILS   '
   4      'LINE SIZE   '       36      'NO PROC MODE'
   5      'SYMBOL SIZE!'       37      'TEKS IN USE '
   6      'ARRAY LIMITS'       38      'VERS TOO NEW'
   7      'STACK LIMITS'       39      'NOT YES / NO'
   8      'SYNTAX!     '       40      'BATCH ERROR '
   9      'CHARACTER?  '       41      'NO RET CODE '
  10      'PRINT       '       42      'TASK ACTIVE '
  11      'NO PROGRAM  '       43      'NOT TASK    '
  12      'ARG LIST?   '       44      'SYNC. FAILS '
  13      'STRING SIZE '       45      'FILE MISSING'
  14      'ALREADY DF  '       46      'NO DESTROY  '
  15      'CONTROL     '       47      'INVALID TAPE'
  16      'LOGIC EXP?  '       48      'TAPE PROBLEM'
  17      'FOR---END?  '       49      'TV PROBLEM  '
  18      'INF LOOP?   '       50      'DISK PROBLEM'
  19      'NO OPERATOR!'       51      'TV UNAVAILAB'
  20      'DIVIDE BY 0 '       52      'OPEN FILE?  '
  21      'IF OR LOOP! '       53      'NOT IN RUN  '
  22      'READ        '       54      'NOT INPUTS  '
  23      'DATA TYPE?  '   55      'CREATE FILE?'
  24      'USING WHAT? '   56      'CLOSE FILE? '
  25      'PROTECTED!  '   57      'PRINTER ERR.'
  26      'SQRT NEGTIVE'   58      'FILE NOT OPN'
  27      'NUMBER SIZE '   59      'TEXT READER '
  28      'RUN IN A RUN'   60      'NOT IN BATCH'
```

```
  29       'LOG NEGATIVE'   61       'DISK PROBLEM'
  30       'VERS TOO OLD'   62       'BAD EXPONENT'
  31       'UNAVAILABLE!'   70       'ONLY IN PROC'
  32       'BOUNDARY LIM'   71       'NOT IN PROC '
 100       'ABORT!!!    '
```

4.2.5 Memory Files

The contents of the K array and LISTF, the source code for procedures, are initially obtained by AIPS from a memory file (type "ME"). The user may save the contents of LISTF and the K array by the pseudo verbs STORE or SAVE. The contents of these arrays can be recovered by the pseudo verbs RESTORE and GET. The working version of LISTF is stored at the beginning of the memory file.

The structure of the memory file is illustrated in the following. The size of the LISTF is given in pages (512 bytes) by variable LPAGE in common /POPS/ and the combined number of pages used by the LISTF and the K array are given by MPAGE in the same common. The current values of LPAGE and MPAGE are 16 and 90, respectively.

```
| Lw | K0 | L0 | K1 | L1 | K2 | L2 | ...

          where Lw = working version of LISTF
                K0 = startup version of the K array
                     initialized by POPSGN.
                L0 = startup version of the LISTF
                     initialized by POPSGN.
                K1 = user STORE area 1 for K array.
                L1 = user STORE area 1 for LISTF.
                K2 = user STORE area 2 for K array.
                L2 = user STORE area 2 for LISTF.
                etc.
```

4.2.6 Special modes

In the normal mode in which AIPS operates, the user types in instructions which are executed immediately. There are several alternate modes in which AIPS can operate. These modes are described briefly in the following sections.

RUN files

AIPS can be directed to read input from a disk text file which can be prepared with the local source editor. The instructions in such a file will be treated in the same fashion as if they were typed in through the terminal. RUN files are used mostly for permanent storage of complex procedures or other fixed data processing schemes. In AIPS, if IUNIT=3 in common /IO/, instructions are read from the RUN file until an end-of-file or an error is encountered.

Batch

AIPS can also be made to run in batch mode at a lower priority. To run AIPS batch, the user edits a file of instructions which are the same as would be given to an interactive AIPS. The major difference is that all tasks are run with DOWAIT=TRUE. This causes AIPS to suspend itself until the task is finished. Another difference is that tape drives, TVs, and graphics devices are not allowed for batch jobs.

The batch file can be created either by an interactive AIPS or a special version of AIPS, called BATER, for this purpose. Once the file is created, the SUBMIT verb sends it to AIPSC which checks the syntax.

One of several possible AIPSBs, the batch AIPSs, is scheduled to execute the batch file. Each of the three versions of AIPS (AIPS, the interactive program; AIPSC, the batch checker; and AIPSB, the batch AIPS) has a separate version of the subroutine VERBS called VERBS, VERBSC and VERBSB, respectively.

Procedures

POPS programs, called procedures, can be entered into the K array or edited by the user with the editor in the POPS processor. Alternately, procedures can be entered by POPSGN when creating the POPS memory files. As a procedure is entered, it is compiled line by line and the final compiled code is stored in the K array. Editing or modifying a procedure will cause the procedure to be recompiled and replaced in the K array.

The source version of the procedures is stored in an array called LISTF which is kept on disk in the current working memory file. All access to the source code causes this file to be read and/or written.

When procedures are recompiled and stored in the K array, the space for the old instructions is not recovered. The verb, COMPRESS, which was to recover this unused space, has never been implemented.

4.3 Example of the POPS processor

The following discussion of the POPS compiler and an example of its action is lifted (with some updates) from the 1978 Sume report.

4.3.1 The Compiler

POPS compiles expressions into reverse polish stacks, which can then be executed by the interpreter. Operators are translated into integers 1, 2, 3,... and operands into negative integers. The magnitudes of the negative integers are the addresses within the K array of the operands. Arithmetic operators carry a precedence which is used in converting expressions into polish sequences. Some operators, such as (and ; are used only at compile time to signal the elevation of precedence of operators, the end of a statement, etc.

The following table lists POPS operators and their precedence level.

Symbol	Meaning	Precedence
=	Store	1
!	Or	2
&	And	2
^	Not	2
=	Equal (as opposed to store)	3
>	Greater than	3
<	Less than	3
<=	Greater or equal	3
>=	Less or equal	3
<>	Not equal	3
TO	Loop control	4
:	Loop control	4
BY	Loop control	4
!!	String concatenation	4
+	Add	5
-	Subtract	5
SUBSTR	String extraction, insertion	5
*	Multiply	6
/	Divide	6
**	Exponentiate	7
-	Unary -	8
+	Unary +	0
Verbs ;,FOR,END,READ,TYPE,PRINT, RETURN, AND DUMP		0
All other verbs		9

Translation to polish form takes place in the routines POLISH and COMPIL as follows: Three push-down stacks, A, B, and BPR, hold operands, operators, and operator precedents respectively, while an expression is scanned from left to right. The expression is contained in the array KARBUF and the tokens are obtained from KARBUF by the subroutine GETFLD (in POLISH) called from COMPIL. Operands are placed on the A stack in order of appearance. Operators are placed on the B stack if their precedence (NEXTP) exceeds the precedence of the last operator on the stack, or if the B stack is empty. Using the BCLEAN subroutine, operators are taken off the B stack and pushed onto A if their precedence is equal to or greater than the precedence of the operator currently being scanned. This takes place until the top operator on the B stack has precedence lower than the one being scanned, or the B stack is emptied, whence the new operator is pushed onto the B stack, and its precedence onto the BPR stack at the corresponding position. If the "(" operator is encountered, the precedence of every subsequent operator is raised by an amount MAXLEV (=10) while ")" lowers the level by MAXLEV. The end of a statement "operator", the ";" operator, and others with which arithmetic expressions may be associated, such as TO, BY, THEN, ELSE, etc. are taken to have lowest possible precedence, so that they have the effect of emptying the B stack. We are then left with the polish sequence of operators and operands in the A stack. For example, the expression.

```
Y = A*(B*X + C);
```

would be translated with the following steps:

```
Step     Token   Prec(token)   A-stack   B-stack  BPR-stack
----     -----   -----------   -------   -------  ---------

(1)        Y       ...         (empty)   (empty)   (empty)

(2)        =        3            Y       (empty)   (empty)

(3)        A       ...           Y          =          3

(4)        *        6            Y          =          3
                                 A

(5)        (      raise level    Y          =          3
                                 A          *          6

(6)        B        ...         --------- SAME ----------

(7)        *      6+MAXLEV       Y          =          3
                                 A          *          6
                                 B

(8)        X        ...          Y          =          3
                                 A          *          6
                                 B          *          6+MAXLEV

(9)        +      5+MAXLEV       Y          =          3
                                 A          *          6
                                 B          *          6+MAXLEV
                                 X

(10)       C         ...         Y          =          3
                                 A          *          6
                                 B          +          5+MAXLEV
                                 X
                                 *

(11)       )       decrement     Y          =          3
                                 A          *          6
                                 B          +          5+MAXLEV
                                 X
                                 *
                                 C

(12)       ;          0           ------------ SAME ------------

(13)   Final result              Y       (empty)      (empty)
                                 A
                                 B
                                 X
                                 *
                                 C
                                 +
                                 *
                                 =
```

4.3.2 The Interpreter

The POPS interpreter executes polish postfix code left by the POPS compiler. To do so requires 3 run-time stacks: the main stack (STACK), the control stack (CSTACK) and a value stack (V).

The main stack holds operand addresses (tags). Corresponding to each operand, the appropriate position in the value stack is loaded with a floating point number, found in core at the stack address. This number may or may not be meaningful, depending on the type of data kept at that address. Operators will make use of the address or value depending on which is appropriate.

The control stack is used to save the run-time location counter (L) and the program chunk link (LINK), together with saved stack pointers, etc. While the main stack could be so used, it was felt that greater reliability would ensue if the control stack were kept separate, guarding from user-caused stack errors (such as leaving garbage on the main stack). Operations using the control stack require an authentication code to appear on the top of the stack before they are activated.

The interpreter expects all operands to be negative integers; all operators, save 0 to be positive (0 is considered a legitimate operand). Operands will be pushed onto the main stack. The value stack, described above, holds intermediate results of computations, as well as the contents of memory when the stack was loaded.

An example, using the arithmetic expression described in the polish compile segment:

```
  Source code:  Y = A * (B * X + C)

           Compiled code
           ---------------
            (1)  -addr. of   Y
            (2)  -addr. of   A
            (3)  -addr. of   B
            (4)  -addr. of   X
            (5)  +TAG of     * operator
            (6)  -addr. of   C
            (7)  +TAG of     + operator
            (8)  +TAG of     * operator
            (9)  +TAG of     = operator

 Execution: Suppose A = 1.5, B = 2.5, C = 3.5, X = 10.0

           Token being
 Step      executed          stack          V
-----      -----------      -------       -------


(1)             Y           (empty)       (empty)

(2)             A             Y           ********

(3)             B             Y           ********
                              A             1.5

(4)             X             Y           ********
                              A             1.5
                              B             2.5

(5)             *             Y           ********
```

		A	1.5
		B	2.5
		X	10.0
(6)	C	Y	********
		A	1.5
		*********	25.0
(7)	+	Y	********
		A	1.5
		*********	25.0
		C	3.5
(8)	*	Y	*********
		A	1.5
		**********	28.5
(9)	=	Y	*********
		**********	42.75
(10)	finish	(empty)	(empty)

4.4 Installing new VERBS

To install a new verb in AIPS several actions are required.

1. Enter the new verb in POPSDAT.HLP and run POPSGN. The new verb will probably be TYPE 4 and should be assigned a verb number (TAG) greater than 100; making sure the verb number is not already used. It should be noted that contiguous groups of verb numbers will use the same AU routine. If the new verb is similar to existing verbs it should be put in the same AU routine if possible.

2. Create or modify an AU routine to perform the desired function. If there are available verb numbers in the range available to the relevant AU routine, then the function can be added to that AU routine. If not, then a new AU routine is required. Note that the branch code sent to the AU routine is the verb number (one) relative to the first verb number in that AU routine. If the verb requires more than a few lines of Fortran, the AU routine should call a subroutine to do the work.

3. Modify VERBS, if necessary, to call the necessary AU routine when it is given the new verb number (J in VERBS). The range of verb numbers in each routine is defined in the arrays IAB and IAE. If new AU routines are added the dimensions of IAB and IAE should be changed and the upper limit on the DO loop index for the loop terminating at statement label 5 should be changed. The computed GO TO in this loop should be modified to include the new AU routine. New AU routines should be added at the end of the list for simplicity. Note that there are three versions of VERBS (VERBS, VERBSC, and VERBSB) for the interactive AIPS, the batch AIPS checker program, and batch AIPS respectively. All three must have corresponding changes although an error return may be desired for the two batch versions in the implementation of a new verb.

4. Compile the necessary subroutines and add them to the AIPS program subroutine library.

5. Recompile and link edit AIPS.

6. Create a HELP file for the verb in the same manner as for a task. Verbs will work without a HELP file, but it is much friendlier to write one.

7. When you start AIPS again, be sure to type "RESTORE 0" in case a pre-existing save/get environment like LASTEXIT gets loaded.

As a convenience for developing new verbs, nine temporary verbs are available, T1VERB, T2VERB, ..., and T9VERB (verb numbers 900–908) These are accessible through the routine AUT. To use one of these verbs all that is necessary is to modify AUT, recompile it, replace it in the AIPS program subroutine library (COMRPL), and recompile AIPS and relink it. Once verbs are tested, they should be moved to a more permanent AU routine if they meet AIPS' portability standards.

The branch code sent to the AU routine is (one) relative to the first verb number in that AU routine. If the verb has one or more arguments, they will be found in the value stack V in common /POPS/ in the reverse of the order in which they were specified. Real values can then be obtained as in the following example:

```
      SUBROUTINE TESTXX
C-----------------------------------------------------------------------
C   Routine to average the top two numbers on the V stack.
C   This routine is designed to be run from VERBS rather than KWICK,
C   that is, it should be called from an AU routine.
C-----------------------------------------------------------------------
      REAL      V1, V2, RESULT
      INTEGER   POTERR
      CHARACTER PRGNAM*6
      PARAMETER ( PRGNAM = 'TESTXX')
      INCLUDE 'INCS:DPOP.INC'
      INCLUDE 'INCS:DERR.INC'
C-----------------------------------------------------------------------
C                                       Set potential error number,
C                                       7 = 'STACK LIMIT'
      POTERR = 7
C                                       Check that stack not
C                                       exhausted.
      IF (SP.GE.2) THEN
C                                       Get values from stack.
         V1 = V(SP-1)
         V2 = V(SP)
C                                       Average.
         RESULT = (V1 + V2) / 2.0
C                                       For two operands change SP and,
C                                       STACK, for one don't change
C                                       SP or STACK.
         SP = SP - 1
         STACK(SP) = 0
C                                       If the verb returns a value,
C                                       RESULT, do the following.
         V(SP) = RESULT
C                                       Set error code
      ELSE
         ERRNUM = POTERR
C                                       Fill in /ERRORS/.
         ERRLEV = ERRLEV + 1
         IF (ERRLEV.LE.5) PNAME(ERRLEV) = PRGNAM
         END IF
C                                       Return
 999  RETURN
      END
```

The stack contents are as follows when a verb is called with an immediate argument:

1. For a real scalar including a subscripted real array adverb,

```
SP = 1     STACK(SP) = TAG     V(SP) = C(TAG) (=value)
```

2. For an array adverb,

```
  SP = 1     STACK(SP) = TYPE     V(SP) may be ignored
       2                 N
       3                 TAG
       4                 2
where for TYPE = 2,7  N = K array pointer to array
                          descriptor block,
                 14    = number of characters,
                  9    = 100 * character offset +
                         # characters
```

Adverbs may be accessed by name using the name as defined in the include DAPL.INC. Note that the order of adverbs is really defined in the POPSDAT.HLP file and the order in DAPL.INC must correspond exactly. Also, all adverbs are of Fortran data type REAL, although they may contain character strings. Note that character strings are stored as HOLLERITH which are allocated storage space as (NCHAR+3)/4 REAL locations.

4.5 Installing new ADVERBS

New, temporary, adverbs can be created in an executing AIPS task by SCALAR, ARRAY or STRING statements in a procedure. Permanent installation of an adverb requires entering it in POPSDAT.HLP, running POPSGN to update the memory files, and adding a variable into the declarations in common /CORE/ in the include DAPL.INC. The new adverbs should be entered in the same relative location among the other adverbs in common as in the POPSDAT file. At the end is, of course, best. The adverb value will be kept in this variable and is, therefore, directly available to verbs. A HELP file should be written for any permanent adverb.

4.6 POPSGN

The initial contents of the POPS memory files, and hence the LISTF and K arrays, are set by the stand alone utility program POPSGN. This program takes as input the file POPSDAT.HLP.

4.6.1 Function

The function of POPSGN is to initialize the contents of LISTF (the source code for procedures) and the K array when AIPS starts up by storing the contents in the POPS memory ("ME") files. This program is normally found in the same place as the AIPS program itself and asks for instructions directly from the key board. When the program begins it asks:

```
ENTER NPOPS1,NPOPS2,IDEBUG,MNAME,VERSION
```

The response is in free format (blanks between entries) and should be as follows:

```
NPOPS1  The lowest POPS number for this run of POPSGN, this
        is normally 1.

NPOPS2  The highest POPS number for this run of POPSGN, this
        is normally the highest POPS number run = No.
        interactive POPS + number of batch queues + 1.

IDEBUG  If not 0, POPSGN will give lots of debug messages.
        Use 0.

MNAME   The name of the file in the HELP area that contains
        the input file for POPSGN.  This is normally
        POPSDAT.HLP; type only 'POPSDAT'.

VERSION  This specifies the version of AIPS to have the
         memory files updated.  Normally this is blank
         which will update the 'TST' area; 'NEW' and 'OLD'
         are also understood by POPSGN.
```

After POPSGN has digested POPSDAT.HLP it will return a ">" prompt. Type a blank line to terminate the input and POPSGN will update the memory files.

4.6.2 POPSDAT.HLP

The bulk of the definitions of verbs, adverbs, and standard procedures are defined in the POPSDAT file. A "C-" in columns one and two indicate a comment line. A "/" character conventionally indicates the beginning of an end-of-line comment which must begin after column 44. The names of symbols begin in column 1 with no embedded blanks and may have no more than 8 characters. The POPSDAT file is read with a (A8,1X,I3,1X,I3,1X,I4,1X,I4,2(1X,F7.2)) format.

The first portion of the POPSDAT file defines the POPS verbs. Most of these verbs and pseudo verbs with verb numbers (TAG) less than 100 reside in the AIPS routine KWICK. Verb numbers greater than 100 are all in AU routines called by VERBS. The values following the symbol name are (1) the number of characters in the symbol name, (2) the symbol type (4 or 5 for verbs and pseudo verbs) and (3) the TAG, in this case the verb number. The end-of-line comments for verbs with numbers (TAG) greater than 100 tell the AU routine in which that verb is found.

Following the verbs come the adverb definitions. The values following the symbol name are: (1) the number of characters in the symbol name, (2) the symbol type (see the section of TYPEs and TAGs). For scalar, real adverbs (TYPE 1) the next two integer fields are blank and the following REAL field (F7.0) is taken to be the initial value of that scalar.

For real arrays (TYPE 2), the first value past the TYPE field is the number of dimensions (1 or 2), the next integer field is blank and the following one or two REAL (F7.1) fields give the number of positions in each of the one or two dimensions.

For character string variables (TYPE 7) the first integer field past the TYPE is the number of dimensions; where the characters in the each string constitute the first dimension. Thus there can only be single dimensional character string array adverbs. The next integer field is blank and the next REAL (F7.0) field is the number of characters in the string. For character string arrays the following REAL (F7.0) field is the second dimension, i.e., the number of elements in the array.

An adverb named QUIT with TYPE = 6 tells POPSGN that all verb and adverb definitions have been read. Following this, normal POPS commands may be entered and the definitions of the standard procedures are normally entered here. A "*" in column 1 indicates a POPS comment line. The end of file terminates the input.

The current contents of POPSDAT is shown in the following:

```
; POPSDAT
;---------------------------------------------------------------
;! lists all POPS symbols, used to create them in MEmory files
;# List POPS
;-----------------------------------------------------------------------
;;  Copyright (C) 1995
;;  Associated Universities, Inc. Washington DC, USA.
;;
;;  This program is free software; you can redistribute it and/or
;;  modify it under the terms of the GNU General Public License as
;;  published by the Free Software Foundation; either version 2 of
;;  the License, or (at your option) any later version.
;;
;;  This program is distributed in the hope that it will be useful,
;;  but WITHOUT ANY WARRANTY; without even the implied warranty of
;;  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;;  GNU General Public License for more details.
;;
;;  You should have received a copy of the GNU General Public
;;  License along with this program; if not, write to the Free
;;  Software Foundation, Inc., 675 Massachusetts Ave, Cambridge,
;;  MA 02139, USA.
;;
;;  Correspondence concerning AIPS should be addressed as follows:
;;         Internet email: aipsmail@nrao.edu.
;;         Postal address: AIPS Project Office
;;                         National Radio Astronomy Observatory
;;                         520 Edgemont Road
;;                         Charlottesville, VA 22903-2475 USA
;-----------------------------------------------------------------------
;---------------------------------------------------------------
POPSDAT   LLLLLLLLLLLLUUUUUUUUUUUU CCCCCCCCCCCCCCCCCCCCCCCCCCCCC
---------------------------------------------------------------
C-                                   This module is POPSDAT.
,            1   4   1                         --|
(            1   4   2                           |
)            1   4   3                           |
=            1   4   4                           |
+            1   4   5                           |
-            1   4   6                           | subtract
*            1   4   7                           |
/            1   4   8                           |
**           2   4   9                           |
>            1   4  10                           |
<            1   4  11                           |
+            1   4  12                           |
-            1   4  13                           | unary
^            1   4  14                           |
TO           2   4  15                           |
:            2   4  15                           |
BY           2   4  16                           |
=            1   4  17                           | logical
!            1   4  18                           |
&            1   4  19                           |
```

```
;            1   4   20                    |
FOR          3   4   21                    |
END          3   4   22                    |
READ         4   4   23                    |
TYPE         4   4   24
PRINT        5   4   24                    |
RETURN       6   4   25
LENGTH       6   4   26                    |
C-                   27             | res array equates
C-RUN        3   4   28                    |
C-EXIT       4   4   29                    |
C-RESTART    7   4   30                    |
LOG          3   4   31
LN           2   4   32
MOD          3   4   33
MODULUS      7   4   34
ATAN2        5   4   35
SIN          3   4   36
COS          3   4   37
TAN          3   4   38
ATAN         4   4   39
SQRT         4   4   40
DUMP         4   4   41                    |
<=           2   4   42                    |
>=           2   4   43                    |
<>           2   4   44                    |
EXP          3   4   45
SUBSTR       6   4   46
!!           2   4   47
CHAR         4   4   48
VALUE        5   4   49
MSGKILL      7   5   50                  --| PSEUDO
PROCEDURE    9   5   51                  --|
PROC         4   5   51                    |
ARRAY        5   5   52                    |
ELSE         4   5   53                    |
THEN         4   5   54                    |
FINISH       6   5   55                    |
DEBUG        5   5   56                    |
IF           2   5   57                    |
STRING       6   5   58
WHILE        5   5   59
SCALAR       6   5   60

EDIT         4   5   61                    | EDITOR
ENDEDIT      7   5   62                  --|
MODIFY       6   5   63
C-storecode          64                  --| reserved
STORE        5   5   65                    |
RESTORE      7   5   66                    |
SAVE         4   5   67                    |
GET          3   5   68                    |
LIST         4   5   69                    | -- STORES
CORE         4   5   70                    |
```

```
SCRATCH      7   5    71                                      |
COMPRESS     8   5    72                                      |
C-endmodify           73                                    --| reserved
SHOW         4   5    76                                      | -- HELPS
TELL         4   5    77                                      | -- HELPS
ISBATCH      7   5    78                                      | -- PSEUDO
ERASE        5   5    79                                      | -- EDITOR
RUN          3   5    80                                      | -- HELPS
HELP         4   5    81                                    --|
INP          3   5    82                                    --|
INPUTS       6   5    83
GO           2   5    84
TGET         4   5    85
SGDESTR      7   5    86
ABORTASK     8   5    87
TPUT         4   5    88
WAITTASK     8   5    89
EXPLAIN      7   5    90
CEIL         4   4    91
FLOOR        5   4    92
ABS          3   4    93
MAX          3   4    94
MIN          3   4    95
C-                    96                                    --| res: END
C-                    97                                        res: WHILE
C-                    98                                        res: SUBS
C-                    99                                        res: NOP

PRTMSG       6   4   100                                      | AU1
EXIT         4   4   101
RESTART      7   4   102
CLRMSG       6   4   103
C-HELP               110                                      | AU1A
C-INP                111
C-INPUTS             112
C-EXPLAIN            113
C-SHOW               114
C-GO         2   4   120                                      | AU2
SPY          3   4   121
C-WAITTASK           122
C-ABORTASK   8   4   123
C-TPUT       4   4   124
C-TELL       4   4   125
STQUEUE      7   4   126
SETDEBUG     8   4   127
C-TGET               130                                      | AU2A
C-SGDESTR            131
TGINDEX      7   4   132
SGINDEX      7   4   133
CATALOG      7   4   150                                      | AU3
MCAT         4   4   151
IMHEADER     8   4   152
ZAP          3   4   153
UCAT         4   4   154
```

QHEADER	7	4	155	
PCAT	4	4	156	
FREESPAC	8	4	160	| AU3A
ALLDEST	7	4	161	
TIMDEST	7	4	162	
SAVDEST	7	4	163	
SCRDEST	7	4	164	
RENUMBER	8	4	170	| AU3B
RECAT	5	4	171	
TPHEAD	6	4	180	| AU4
AVFILE	6	4	181	
AVMAP	5	4	182	
REWIND	6	4	183	
AVEOT	5	4	184	
MOUNT	5	4	185	
DISMOUNT	8	4	186	
TVINIT	6	4	200	| AU5
TVCLEAR	7	4	201	
GRCLEAR	7	4	202	
TVON	4	4	203	
TVOFF	5	4	204	
GRON	4	4	205	
GROFF	5	4	206	
TV3COLOR	8	4	207	
TVPOS	5	4	208	
IMXY	4	4	209	
IMPOS	5	4	210	
TVNAME	6	4	211	
CURBLINK	8	4	212	
TVLOD	5	4	220	| AU5A
TVROAM	6	4	221	
SETROAM	7	4	222	
REROAM	6	4	223	
TVLABEL	7	4	240	| AU5B
TVWLABEL	8	4	241	
TVANOT	6	4	242	
TVWEDGE	7	4	250	| AU5C
IMWEDGE	7	4	251	
WEDERASE	8	4	252	
IMERASE	7	4	253	
TVWINDOW	8	4	254	
TVBOX	5	4	255	
TVSLICE	7	4	256	
REBOX	5	4	257	
TVMOVIE	7	4	260	| AU5D
REMOVIE	7	4	261	
TVCUBE	6	4	262	
OFFPSEUD	8	4	280	| AU6
OFFZOOM	7	4	281	
OFFSCROL	8	4	282	
TVZOOM	6	4	283	
TVSCROL	7	4	284	
TVPSEUDO	8	4	285	

```
TVHUEINT     8   4  286
OFFTRAN      7   4  290                            | AU6A
TVTRANSF     8   4  291
TVBLINK      7   4  292
TVMBLINK     8   4  293
TVLUT        5   4  294
TVMLUT       6   4  295
TVSPLIT      7   4  296
CURVALUE     8   4  300                            | AU6B
C-TVALL        5   4  305                          | AU6C
TVFIDDLE     8   4  306
TVSTAT       6   4  310                            | AU6D
IMSTAT       6   4  311
PRTHI        5   4  330                            | AU7
RENAME       6   4  331
RESCALE      7   4  332
CLRSTAT      7   4  333
AXDEFINE     8   4  334
ALTDEF       6   4  335
ALTSWTCH     8   4  336
CELGAL       6   4  337
ADDBEAM      7   4  340                            | AU7A
PUTHEAD      7   4  341
GETHEAD      7   4  342
PUTVALUE     8   4  343
CLRNAME      7   4  360                            | AU8
GETNAME      7   4  361
GET2NAME     8   4  362
GET3NAME     8   4  363
EXTDEST      7   4  364
CLR2NAME     8   4  365
CLR3NAME     8   4  366
EGETNAME     8   4  367
GETONAME     8   4  368
CLRONAME     8   4  369
EXTLIST      7   4  370                            | AU8A
MAXFIT       6   4  390                            | AU9
IMVAL        5   4  391
QIMVAL       6   4  392
TKPOS        5   4  400                            | AU9A
TKVAL        8   4  401
TKXY         4   4  402
TKSLICE      7   4  410                            | AU9B
TKASLICE     8   4  411
TKMODEL      7   4  412
TKAMODEL     8   4  413
TKRESID      7   4  414
TKARESID     8   4  415
TKGUESS      7   4  416
TKAGUESS     8   4  417
TKSET        5   4  420                            | AU9C
TK1SET       6   4  421
SUBMIT       6   4  440                            | AUA
BATCH        5   4  441                            | AUB
```

```
BATEDIT      7   4  442
UNQUE        5   4  443
BATCLEAR     8   4  444
BATLIST      7   4  445
QUEUES       6   4  446
JOBLIST      7   4  447
BAMODIFY     8   4  448
GRIPE        5   4  460                            | AUC
GRINDEX      7   4  461
GRLIST       6   4  462
PASSWORD     8   4  463
GRDROP       6   4  464
T1VERB       6   4  900                            | AUT
T2VERB       6   4  901
T3VERB       6   4  902
T4VERB       6   4  903
T5VERB       6   4  904
T6VERB       6   4  905
T7VERB       6   4  906
T8VERB       6   4  907
T9VERB       6   4  908

USERID       6   1                0.00
INNAME       6   7    1          12.00
INCLASS      7   7    1           6.00
INSEQ        5   1                0.00
INDISK       6   1                0.00
INTYPE       6   7    1           2.00
IN2NAME      7   7    1          12.00
IN2CLASS     8   7    1           6.00
IN2SEQ       6   1                0.00
IN2DISK      7   1                0.00
IN2TYPE      7   7    1           2.00
IN3NAME      7   7    1          12.00
IN3CLASS     8   7    1           6.00
IN3SEQ       6   1                0.00
IN3DISK      7   1                0.00
IN3TYPE      7   7    1           2.00
OUTNAME      7   7    1          12.00
OUTCLASS     8   7    1           6.00
OUTSEQ       6   1                0.00
OUTDISK      7   1                1.00
OUT2NAME     8   7    1          12.00
OUT2CLAS     8   7    1           6.00
OUT2SEQ      7   1                0.00
OUT2DISK     8   1                1.00
INEXT        5   7    1           2.00
IN2EXT       6   7    1           2.00
IN3EXT       6   7    1           2.00
INVERS       6   1                0.00
IN2VERS      7   1                0.00
IN3VERS      7   1                0.00
BADDISK      7   2    1          10.00
INTAPE       6   1                1.00
```

```
OUTTAPE      7   1           1.00
NFILES       6   1           0.00
NMAPS        5   1           0.00
TASK         4   7   1       8.00
DOWAIT       6   1          -1.00
PRIORITY     8   1           0.00
BLC          3   2   1       7.00
TRC          3   2   1       7.00
XINC         4   1           1.00
YINC         4   1           1.00
PIXXY        5   2   1       7.00
PIXVAL       6   1           0.00
PIXRANGE     8   2   1       2.00
FACTOR       6   1           0.00
OFFSET       6   1           0.00
TVBUT        5   1           0.00
XTYPE        5   1           5.00

XPARM        5   2   1      10.00
YTYPE        5   1           5.00
YPARM        5   2   1      10.00
OPCODE       6   7   1       4.00
FUNCTYPE     8   7   1       2.00
ROTATE       6   1           0.00
GAIN         4   1           0.10
NITER        5   1           0.00
FLUX         4   1           0.00
OBJECT       6   7   1       8.00
QUAL         4   1          -1.00
STOKES       6   7   1       4.00
BAND         4   7   1       4.00
TVCHAN       6   1           1.00
GRCHAN       6   1           0.00
TVLEVS       6   1         256.00
TVCORN       6   2   1       2.00
COLORS       6   1           0.00
TVXY         4   2   1       2.00
DOTV         4   1          -1.00
BATQUE       6   1           2.00
BATFLINE     8   1           0.00
BATNLINE     8   1           0.00
JOBNUM       6   1           0.00
LTYPE        5   1           3.00
PLEV         4   1           0.00
CLEV         4   1           0.00
LEVS         4   2   1      30.00
XYRATIO      7   1           0.00
DOINVERS     8   1          -1.00
DOCENTER     8   1           1.00
ZXRATIO      7   1           0.25
SKEW         4   1          45.00
DOCONT       6   1           1.00
DOVECT       6   1           1.00
ICUT         4   1           0.10
```

PCUT	4	1		0.10	
DIST	4	1		3.00	
IMSIZE	6	2	1	2.00	
CELLSIZE	8	2	1	2.00	
SHIFT	5	2	1	2.00	
SORT	4	7	1	2.00	
UVTAPER	7	2	1	2.00	
UVRANGE	7	2	1	2.00	
UVWTFN	6	7	1	2.00	
UVBOX	5	1		0.00	
DOGRIDCR	8	1		1.00	
ZEROSP	6	2	1	5.00	
BITER	5	1		0.00	
BMAJ	4	1		0.00	
BMIN	4	1		0.00	
BPA	3	1		0.00	
NBOXES	6	1		0.00	
BOX	3	2	2	4.00	10.00
DOEOF	5	1		1.00	
NDIG	4	1		0.00	
DOCAT	5	1		1.00	
DOHIST	6	1		-1.00	
BDROP	5	1		0.00	
EDROP	5	1		0.00	
ASPMM	5	1		0.00	
MINPATCH	8	1		51.00	
APARM	5	2	1	10.00	
BPARM	5	2	1	10.00	
GPOS	4	2	2	2.00	4.00
GMAX	4	2	1	4.00	
GWIDTH	6	2	2	3.00	4.00

DOPOS	5	2	2	2.00	4.00
DOMAX	5	2	1	4.00	
DOWIDTH	7	2	2	3.00	4.00
NGAUSS	6	1		0.00	
TRANSCOD	8	7	1	14.00	
AXREF	5	1		1.00	
NAXIS	5	1		3.00	
AXINC	5	1		0.00	
AXVAL	5	2	1	2.00	
AXTYPE	6	7	1	8.00	
DOSLICE	7	1		1.00	
DOMODEL	7	1		-1.00	
DORESID	7	1		-1.00	
ROMODE	6	1		0.00	
DETIME	6	1		0.00	
DOCRT	5	1		-1.00	
CHANNEL	7	1		0.00	
CPARM	5	2	1	10.00	
DPARM	5	2	1	10.00	
DOALIGN	7	1		1.00	
NPOINTS	7	1		1.00	
AX2REF	6	1		0.00	

```
DOALL       5   1          -1.00
TXINC       5   1           1.00
TYINC       5   1           1.00
TBLC        4   2   1       7.00
TTRC        4   2   1       7.00
VERSION     7   7   1      48.00
DOEOT       5   1           1.00
DOSTOKES    8   1          -1.00
PRTLEV      6   1           0.00
DOARRAY     7   1          -1.00
ZINC        4   1           1.00
TZINC       5   1           1.00
BCHAN       5   1           1.00
ECHAN       5   1           0.00


RESTFREQ    8   2   1       2.00
INFILE      6   7   1      48.00
IN2FILE     7   7   1      48.00
OUTFILE     7   7   1      48.00
DENSITY     7   1        1600.00
KEYWORD     7   7   1       8.00
KEYVALUE    8   2   1       2.00
KEYSTRNG    8   7   1      16.00
BCOUNT      6   1           1.00
ECOUNT      6   1           0.00
NCOUNT      6   1           0.00
DOTABLE     7   1           1.00
DOTWO       5   1          -1.00
COPIES      6   1           1.00
PRNUMBER    8   1           0.00
PRTIME      6   1           0.00
PRTASK      6   7   1       5.00
CTYPE       5   2   1       4.00
PIXAVG      6   1           0.00
PIXSTD      6   1           0.00
DOCIRCLE    8   1          -1.00
CHINC       5   1           1.00
NFIELD      6   1           1.00
FLDSIZE     7   2   2       2.00   16.00
RASHIFT     7   2   1      16.00
DECSHIFT    8   2   1      16.00
PHAT        4   1           0.00
GAINERR     7   2   1      30.00
TIMSMO      6   2   1      30.00
DOOUTPUT    8   1          -1.00
DOCONCAT    8   1          -1.00
DONEWTAB    8   1           1.00
DOCONFRM    8   1          -1.00
DOALPHA     7   1          -1.00
ERROR       5   1          -1.00
GRNAME      6   7   1      20.00
GRADDRES    8   7   1      48.00
GRPHONE     7   7   1      16.00
SLOT        4   1           1.00
```

VLAOBS	6	7	1	6.00	
VLAMODE	7	7	1	2.00	
CMETHOD	7	7	1	4.00	
CMODEL	6	7	1	4.00	
BCOMP	5	2	1	16.00	
NCOMP	5	2	1	16.00	
LPEN	4	1		3.00	
PRSTART	7	1		1.00	
OPTYPE	6	7	1	4.00	
DOWEDGE	7	1		1.00	
SOURCES	7	7	2	16.00	30.00
CALSOUR	7	7	2	16.00	30.00
TIMERANG	8	2	1	8.00	
SUBARRAY	8	1		1.00	
BIF	3	1		0.00	
EIF	3	1		0.00	
ANTENNAS	8	2	1	50.00	
BASELINE	8	2	1	50.00	
DOCALIB	7	1		1.00	
INTERPOL	8	7	1	4.00	
SMOTYPE	7	7	1	4.00	
INTPARM	7	2	1	3.00	
FLAGVER	7	1		0.00	
GAINVER	7	1		0.00	
GAINUSE	7	1		0.00	
REASON	6	7	1	24.00	
SAMPTYPE	8	7	1	4.00	
CODETYPE	8	7	1	4.00	
NOISE	5	2	1	64.00	
PBSIZE	6	2	1	64.00	
OUTVERS	7	1		0.00	
DOCELL	6	1		1.00	
PIX2XY	6	2	1	7.00	
PIX2VAL	7	1		0.00	
STFACTOR	8	1		0.00	
CUTOFF	6	1		0.00	
OPTELL	6	7	1	4.00	
FORMAT	6	1		0.00	
BLVER	5	1		-1.00	
BPVER	5	1		-1.00	
ANTWT	5	2	1	30.00	
SOLINT	6	1		0.00	
CALCODE	7	7	1	4.00	
REFANT	6	1		0.00	
SMODEL	6	2	1	7.00	
SOLTYPE	7	7	1	4.00	
SOLMODE	7	7	1	4.00	
SOLCON	6	1		0.00	
WTUV	4	1		0.00	
DODELAY	7	1		-1.00	
SYSVEL	6	1		0.00	
VELDEF	6	7	1	8.00	
VELTYP	6	7	1	8.00	

```
RESTFREQ     8   2    1          2.00
DOHMS        5   1              -1.00
BLOCKING     8   1               1.00
PMODEL       6   2    1          7.00
DOPOL        5   1              -1.00
DOBAND       6   1              -1.00
SMOOTH       6   2    1          3.00
DOUVCOMP     8   1              -1.00
REWEIGHT     8   2    1          2.00
REFDATE      7   7    1          8.00
SELBAND      7   1              -1.00
SELFREQ      7   1              -1.00
FREQID       6   1              -1.00
CHANSEL      7   2    2          3.00   10.00
FQTOL        5   1              -1.00
C-   Adverbs below are dummys for testing.
STRA1        5   7    1          4.00
STRA2        5   7    1          8.00
STRA3        5   7    1         12.00
STRB1        5   7    1          4.00
STRB2        5   7    1          8.00
STRB3        5   7    1         12.00
STRC1        5   7    1          4.00
STRC2        5   7    1          8.00
STRC3        5   7    1         12.00
ARRAY1       6   2    1         10.00
ARRAY2       6   2    2         20.00    2.00
ARRAY3       6   2    1          3.00
SCALR1       6   1               1.00
SCALR2       6   1               0.00
SCALR3       6   1               0.00
C-  Quit tells POPSGN 'end of adverbs'.
QUIT         4   6


*
VERSION = ' '; OPTELL = 'CHAN'
DOPOS = 1 ;  DOMAX = 1 ;  DOWIDTH = 1 ;
*
PROC TSTDUM
SCALAR X, Y, I , J , DELTAX , DELTAY
FINISH
*
PROC SETXWIN(DELTAX,DELTAY);IMXY;BLC(1)=PIXXY(1)-DELTAX/2
TRC(1)=BLC(1)+DELTAX;BLC(2)=PIXXY(2)-DELTAY/2;
TRC(2)=BLC(2)+DELTAY;RETURN;FINISH
*
PROC OFFROAM;I=TVCHAN;J=GRCHAN;TVCHAN=1234;GRCHAN=1234;
OFFSCROL;TVOFF;GRCHAN=J;TVCHAN=I;TVON;RETURN;FINISH
*
PROC OFFHUINT; I=ABS(TVCHAN); IF I < 12  THEN I=12; END
J=MOD(I/10,10);I=MOD(I,10);TVOFF(1234);OFFPS;TVCH=I;OFFTR;
TVCH=J;OFFTR;TVON;RETURN
FINISH
*
```

```
PROC TKWIN;TKXY;BLC=PIXXY;TKXY;TRC=PIXXY;
RETURN;FINISH
*
PROC TKBOX(I); TKXY;BOX(1,I)=PIXXY(1);BOX(2,I)=PIXXY(2)
TKXY;BOX(3,I)=PIXXY(1);BOX(4,I)=PIXXY(2);RETURN;FINISH
*
PROC TKNBOXS(NBOXES); FOR J=1:NBOXES;
TYPE 'SET BOX NUMBER',J,' :';TKBOX(J);END;RETURN
FINISH
*
PROC TVRESET; COLOR=0;TVOFF(12345);TVON(TVCH); OFFZ; OFFSC;
OFFPS; GRCH=0;GRCLEAR; OFFTR;RETURN; FINISH
*
PROC TVALL; TVOFF(1234);OFFZOOM;TVON(TVCH);GROFF(1234);J=GRCH;
GRCH=24;GRCL;GRCH=J;TVCL;TVLOD;TVWED(16);TVWLAB;TVFID;RETURN
FINISH
*-----------------------------------------------------------------
```

4.7 INCLUDEs

4.7.1 DAPL.INC

```
C                                                           Include DAPL.
C                                      length before adverb values
      INTEGER   K(14770)
C                                      full length
      REAL      C(18944)
      HOLLERITH CH(18944)
C                                      character strings
      HOLLERITH INNAM(3), INCLS(2), INTYP, IN2NAM(3), IN2CLS(2), IN2TYP,
     *   IN3NAM(3), IN3CLS(2), IN3TYP, OUTNAM(3), OUTCLS(2), OU2NAM(3),
     *   OU2CLS(2), INEXT, IN2EXT, IN3EXT, TASK(2), OPCODE, FUNTYP,
     *   OBJECT(2), STOKES, BAND, SORT, UVWTFN, TRANSC(4), AXTYPE(2),
     *   VERSON(12), INFLL(12), IN2FLL(12), OUTFLL(12), KEYWRD(2),
     *   KEYSTR(4), PRTASK(2), GRNAME(5), GRADDR(12), GRPHON(4),
     *   VLAOBS(2), VLAMOD, CMETHX, CMODXX, OPTYPE, CSOURS(4,30),
     *   CCALS(4,30), CINTPL, CSMOTY, CREASO(6), SMPTYP, CDETYP, OPTELL,
     *   VELDEF(2), VELTYP(2), XREFDA(2),
     *         STRA1, STRA2(2), STRA3(3), STRB1, STRB2(2), STRB3(3),
     *         STRC1, STRC2(2), STRC3(3)
C                                      numeric variables
      REAL   XTRUE, XFALSE, USERID, INSEQ, INDSK, IN2SEQ, IN2DSK,
     *   IN3SEQ, IN3DSK, OUTSEQ, OUTDSK, OU2SEQ, OU2DSK, INVER, IN2VER,
     *   IN3VER, BADDSK(10), INTAPE, OUTTAP, NFILES, NMAPS, DOWAIT,
     *   PRIOTY, BLCORN(7), TRCORN(7), XINC, YINC, PIXXY(7), PIXVAL,
     *   PXRANG(2), FACTOR, OFFSET, TVBUTT, XTYPE, XPARM(10), YTYPE,
     *   YPARM(10), ROTATE, GAIN, NITER, FLUX, QUAL, TVCHAN, GRCHAN,
     *   TVLEVS, TVCORN(2), COLORS, TVXY(2), DOTV, BATQUE, BTFLIN,
     *   BTNLIN, JOBNUM, LTYPE, PLEV, CLEV, LEVS(30), XYRATO, DOINVR,
     *   DOCENT, ZXRATO, SKEW, DOCONT, DOVECT, ICUT, PCUT, DIST,
     *   IMSIZE(2)
      REAL   CELSIZ(2), SHIFT(2), UVTAPR(2), UVRANG(2), UVBOX, DOGRDC,
     *   ZEROSP(5), BITER, CBMAJ, CBMIN, CBPA, NBOXES, BOX(4,10), DOEOF,
     *   NDIG, DOCAT, DOHIST, BDROP, EDROP, ASPMM, MPTCH, APARMS(10),
```

```
     *   BPARMS(10), GPOS(2,4), GMAX(4), GWIDTH(3,4), ERRPOS(2,4),
     *   ERRMAX(4), ERRWTH(3,4), NGAUSS, AXREF, NAXIS, RAXINC, AXVAL(2),
     *   DOSLIC, DOMODL, DORESI, ROMODE, DETIME, DOCRT, CHANNL,
     *   CPARM(10), DPARM(10), DOALIN, NPONTS, AX2REF, DOALL, TVXINC,
     *   TVYINC, TVBLCO(7), TVTRCO(7), DOEOT, DOSTOK, LEVPRT, DORRAY,
     *   ZINC, TVZINC, BECHAN, ENCHAN, RESTFR(2), DENSTY, KEYVAL(2)
      REAL   BEGCNT, ENDCNT, NUMCNT, DOTABL, DOTWO, COPIES, PRNUMB,
     *   PRTIME, CTYPES(4), PIXAVG, PIXRMS, DOCIRC, XCHINC, XNFIEL,
     *   XFLDSZ(2,16), XRASHF(16), XDCSHF(16), XPHAT, XGNERR(30),
     *   XTMSMO(30), DOOUTP, DOCNCT, DONEW, DOCONF, DOALPH, ERRORA,
     *   SLOTAD, XBCOMP(16), XNCOMP(16), QMSPEN, PRSTRT, DOWDGE,
     *   XTIMR(8), XSUBAR, XBIF, XEIF, XANTS(50), XBASLN(50), XDOCAL,
     *   XINTPR(3), XFLGVE, XGAVER, XGAUSE, ANOISE(64), PBSIZE(64),
     *   OUTVER, DOCELL, PIX2XY(7), PIX2VL, STFCTR, CUTOFF, TAMROF,
     *   XBLV, XBPV, XANTWT(30), XSOLIN, XCALCO, XREFA, XSMODE(7)
      REAL   XSOLTY, XSOLMO, XSOLCO, XWTUV, XDODEL, XSYSVL, DOHMS,
     *   BLOCKD, XPMDL(7), XDOPOL, XDOBND, XSMOTH(3), XDOUVC, XXREWT(2),
     *   XSELBN, XSELFQ, XFQID, CHNSEL(3,10), XFQTOL,
     *         ARRAY1(10), ARRAY2(20,2), ARRAY3(3), SCALR1, SCALR2,
     *         SCALR3
      COMMON /CORE/ K, XTRUE, XFALSE, USERID, INNAM, INCLS, INSEQ,
     *   INDSK, INTYP, IN2NAM, IN2CLS, IN2SEQ, IN2DSK, IN2TYP, IN3NAM,
     *   IN3CLS, IN3SEQ, IN3DSK, IN3TYP, OUTNAM, OUTCLS, OUTSEQ, OUTDSK,
     *   OU2NAM, OU2CLS, OU2SEQ, OU2DSK, INEXT, IN2EXT, IN3EXT, INVER,
     *   IN2VER, IN3VER, BADDSK, INTAPE, OUTTAP, NFILES, NMAPS, TASK,
     *   DOWAIT, PRIOTY, BLCORN, TRCORN, XINC, YINC, PIXXY, PIXVAL,
     *   PXRANG, FACTOR, OFFSET, TVBUTT, XTYPE, XPARM, YTYPE, YPARM,
     *   OPCODE, FUNTYP, ROTATE, GAIN, NITER, FLUX, OBJECT, QUAL,
     *   STOKES, BAND, TVCHAN, GRCHAN, TVLEVS, TVCORN, COLORS, TVXY,
     *   DOTV, BATQUE, BTFLIN, BTNLIN, JOBNUM, LTYPE, PLEV, CLEV, LEVS,
     *   XYRATO, DOINVR, DOCENT, ZXRATO, SKEW, DOCONT, DOVECT, ICUT,
     *   PCUT, DIST, IMSIZE
      COMMON /CORE/ CELSIZ, SHIFT, SORT, UVTAPR, UVRANG, UVWTFN, UVBOX,
     *   DOGRDC, ZEROSP, BITER, CBMAJ, CBMIN, CBPA, NBOXES, BOX, DOEOF,
     *   NDIG, DOCAT, DOHIST, BDROP, EDROP, ASPMM, MPTCH, APARMS,
     *   BPARMS, GPOS, GMAX, GWIDTH, ERRPOS, ERRMAX, ERRWTH, NGAUSS,
     *   TRANSC, AXREF, NAXIS, RAXINC, AXVAL, AXTYPE, DOSLIC, DOMODL,
     *   DORESI, ROMODE, DETIME, DOCRT, CHANNL, CPARM, DPARM, DOALIN,
     *   NPONTS, AX2REF, DOALL, TVXINC, TVYINC, TVBLCO, TVTRCO, VERSON,
     *   DOEOT, DOSTOK, LEVPRT, DORRAY, ZINC, TVZINC, BECHAN, ENCHAN,
     *   RESTFR, INFLL, IN2FLL, OUTFLL, DENSTY, KEYWRD, KEYVAL, KEYSTR,
     *   BEGCNT, ENDCNT, NUMCNT, DOTABL, DOTWO, COPIES, PRNUMB, PRTIME,
     *   PRTASK, CTYPES, PIXAVG, PIXRMS, DOCIRC, XCHINC, XNFIEL, XFLDSZ
      COMMON /CORE/ XRASHF, XDCSHF, XPHAT, XGNERR, XTMSMO, DOOUTP,
     *   DOCNCT, DONEW, DOCONF, DOALPH, ERRORA, GRNAME, GRADDR, GRPHON,
     *   SLOTAD, VLAOBS, VLAMOD, CMETHX, CMODXX, XBCOMP, XNCOMP, QMSPEN,
     *   PRSTRT, OPTYPE, DOWDGE, CSOURS, CCALS, XTIMR, XSUBAR, XBIF,
     *   XEIF, XANTS, XBASLN, XDOCAL, CINTPL, CSMOTY, XINTPR, XFLGVE,
     *   XGAVER, XGAUSE, CREASO, SMPTYP, CDETYP, ANOISE, PBSIZE, OUTVER,
     *   DOCELL, PIX2XY, PIX2VL, STFCTR, CUTOFF, OPTELL, TAMROF, XBLV,
     *   XBPV, XANTWT, XSOLIN, XCALCO, XREFA, XSMODE, XSOLTY, XSOLMO,
     *   XSOLCO, XWTUV, XDODEL, XSYSVL, VELDEF, VELTYP, DOHMS, BLOCKD,
     *   XPMDL, XDOPOL, XDOBND, XSMOTH, XDOUVC, XXREWT, XREFDA, XSELBN,
     *   XSELFQ, XFQID, CHNSEL, XFQTOL,
```

```
     *        STRA1, STRA2, STRA3, STRB1, STRB2, STRB3, STRC1, STRC2,
     *        STRC3, ARRAY1, ARRAY2, ARRAY3, SCALR1, SCALR2, SCALR3
      EQUIVALENCE (K(1), C(1), CH(1))
C                                                           End DAPL.
```

4.7.2 DBAT.INC

```
C                                                           Include DBAT.
      INTEGER   BATLUN, BATIND, BATREC, BATDUM, BATDAT(256)
      COMMON /BATCH/ BATLUN, BATIND, BATREC, BATDUM, BATDAT
C                                                           End DBAT.
```

4.7.3 DBWT.INC

```
C                                                           Include DBWT.
      INTEGER   BWTNUM, BWTLUN, BWTIND, BWTREC, BWTDAT(256)
      LOGICAL   WASERR
      CHARACTER BWTNAM*48
      COMMON /BWTCHC/ BWTNAM
      COMMON /BWTCH/ BWTDAT, BWTNUM, BWTLUN, BWTIND, BWTREC, WASERR
C                                                           End DBWT.
```

4.7.4 DCON.INC

```
C                                                           Include DCON.
      INTEGER   K(18944), KXORG
      REAL      C(18944)
      HOLLERITH CH(18944)
      COMMON /CORE/ K
      EQUIVALENCE (K(1), C(1), CH(1)),     (K(8), KXORG)
C                                                           End DCON.
```

4.7.5 DERR.INC

```
C                                                           Include DERR.
C                                      POPS error handling for OERROR
      INTEGER   ERRNUM, IERROR(10), ERRLEV
      CHARACTER PNAME(10)*6
      COMMON /ERRORS/ ERRNUM, IERROR, ERRLEV
      COMMON /ERRORC/ PNAME
C                                                           End DERR.
```

4.7.6 DIO.INC

```
C                                                           Include DIO.
      INTEGER   NBYTES, KARLIM, IUNIT
      CHARACTER KARBUF*80, JBUFF*80, HOLDUF*80, IPT*1
      COMMON /POPSIO/ NBYTES, KARLIM, IUNIT
      COMMON /POPCIO/ KARBUF, JBUFF, HOLDUF, IPT
C                                                           End DIO.
```

4.7.7 DPOP.INC

```
C                                                           Include DPOP.
      INTEGER   KT, LPGM, LLIT, LAST, IDEBUG, MODE, IFFLAG, LINK,
     *   L, NAMEP, IP, LP, SLIM, AP, BP, ONE, ZERO, TRUE, FALSE,
     *   STACK(100), CSTACK(100), SP, CP, SP0, MPAGE, LPAGE
      REAL   V(100), XX
      COMMON /POPS/ V, XX, KT, LPGM, LLIT, LAST, IDEBUG, MODE, IFFLAG,
     *   LINK, L, NAMEP, IP, LP, SLIM, AP, BP, ONE, ZERO, TRUE, FALSE,
     *   STACK, CSTACK, SP, CP, SP0, MPAGE, LPAGE
C                                                           End DPOP.
```

4.7.8 DSMS.INC

```
C                                                           Include DSMS.
      INTEGER   NKAR, KBPTR, NEWCOD, TYPE, TAG, LEVEL, LX, NEXTP, LOCSYM
      REAL      SKEL, X(25)
      HOLLERITH KPAK(5)
      COMMON /SMSTUF/ KPAK, X, SKEL, NKAR, KBPTR, NEWCOD, TYPE,
     *   TAG, LEVEL, LX, NEXTP, LOCSYM
C                                                           End DSMS.
```