OS

File No. S360-26
Form C33-4000-0

IBM Systems Reference Library

IBM System/360 Operating System

ALGOL Programmer's Guide

Program Number 360S-AL-531....Compiler
360S-LM-532....Library Routines

This publication describes how to compile, linkage edit and execute a program written in the System/360 Operating System Algorithmic Language (ALGOL). It includes an introduction to the operating system and a description of the information listings that can be produced, the job control language, and the subroutine library.

PREFACE

This publication is intended for use by Application Programmers, Systems Programmers and IBM Systems Engineers. A knowledge of ALGOL is assumed, and the reader is expected to be familiar with the prerequisite publication:

IBM System/360 Operating System: ALGOL Language. Form C28-6615.

In Section 2, the description "IBM-Supplied Cataloged Procedures" provides sufficient information to process and execute an ALGOL program that can use the IBM-supplied cataloged procedures without modification.

The rest of Section 2, together with information in Section 1 and the Appendices, will be required for programs that cannot use the IBM-supplied cataloged procedures without modification.

The description of information listings in Section 3 and the list of diagnostic messages given in Appendix F will be helpful in interpreting system output, especially for debugging.

An extensive index has been provided to assist the reader in using the manual for reference purposes.

This publication contains most of the information required by the Applications Programmer.

The following publications are referred to within the text for information beyond the scope of this publication.

IBM System/360 Operating System: Assembler Language. Form C28-6514.

IBM System/360 Operating System: Data Management. Form C28-6537.

IBM System/360 Operating System: Linkage Editor. Form C28-6538.

IBM System/360 Operating System: Job Control Language. Form C28-6539.

IBM System/360 Operating System: Operator's Guide. Form C28-6540.

IBM System/360 Operating System: Control Program Services. Form C28-6541.

IBM System/360 Operating System: Utilities. Form C28-6586.

IBM System/360 Operating System: FORTRAN IV Library Subprograms. Form C28-6596.

IBM System/360 Operating System: Messages, Completion Codes, and Storage Dumps. Form C28-6631.

First Edition (April 1967)
Significant changes or additions to the specifications contained in this publication will be reported in subsequent revisions or Technical Newsletters.

CONTENTS

FIGURES

The primary constituent of a System/360 data processing operation is a job. This, basically, is the work that the user requires the computer to do. To carry out a job, a computer needs two types of information -- a program and data.

- A program (known in this context as a source program) is a sequence of instructions which specify the actions to be performed by the machine. These instructions are written in a symbolic language and are translated into machine language by a processing program contained in the operating system before they are performed.
- Data is the information to be processed by the program. The source program is regarded as data while it is being processed by operating system programs to make it suitable for execution.

From this brief introduction it can be seen that a job is affected by three separate factors -- the source program, the operating system and the machine configuration.

SOURCE PROGRAM

For jobs discussed in this publication, the source program will be written primarily in System/360 Operating System ALGOL (Algorithmic Language). This is defined in IBM System/360 Operating System: ALGOL Language. In addition the programmer must observe the restrictions, caused by internal capacity limitations, listed in Section 4.

An ALGOL source program may be written in freeform on any 80 column coding sheet. The program text is contained in columns 1 to 72. Columns 73 to 80 can be used by the programmer for program identification. To avoid confusion with job control statements (see "Operating System"), the character sequences // and /* must not be used in columns 1 and 2. It is possible to do this since these sequences are syntactically incorrect outside strings, and when they occur within strings, they may be shifted into non-critical columns by inserting a blank space before the opening string quotes '('. Two character sets are available for punching the source program into a card deck (see Appendix C).

For operations that require more precise control over the machine than can be provided by ALGOL, subprograms written in Assembler language can be included in the source program (see Section 4). Assembler language subprograms can also be used as a link to other languages, such as PL/I, COBOL and FORTRAN. The Assembler language is defined in IBM System/360 Operating System: Assembler Language.

OPERATING SYSTEM

The System/360 Operating System is a set of IBM-supplied, control and processing programs (supplemented if necessary by user-written programs) that assist the programmer to use the computer efficiently. The operating system selected for a particular installation is generated during the initial setting-up of the computer, by a process known as system generation.

Job Control

Operating system instructions (known as job control statements) must be added to the source program to control its handling within the operating system and to specify the data management facilities required.

These statements do not need to be specified until the program is ready to be executed. This means that the program can be prepared independent of installation considerations.

Six types of statements are available, which, in conjunction with associated parameters, can supply all information required by the operating system for job control. To save programming effort, commonly used sequences of control statements can be stored by the system for subsequent recall by identifying names. These sequences are known as cataloged procedures.

JOB is the first statement of each job. It indicates that a new job is beginning and, consequently, that the previous job has ended. A job can be divided into a number of job steps, which can be inter-related to improve processing efficiency. For example, the execution of one job step can be made dependent on the result of a previous one. This is an important feature of the operating system and users are recommended to exploit it as fully as possible.

EXEC (Execution) is the first statement in each job step. It specifies the program or cataloged procedure to be executed, and must be included even if the job consists of only one job step.

DD (Data Definition) is the statement used to describe a data set and to specify associated data control block information. It also specifies input/output (I/O) device assignment. One or more DD statements are usually required for each job step.

In addition the command statement is used to place operator commands into the input stream, the null statement indicates the end of the last job in the input stream, and the delimiter statement separates data from subsequent control statements when sequential scheduling is used. The command statement, when used, must immediately precede a JOB, EXEC or null statement.

The job control statements required for an ALGOL source program are described in Section 2.

Control Program

The control program is the primary program within the operating system and must be included with all installations. It is divided into a number of functions. Those affecting the applications programmer are described in the following text.

Job Scheduling

A job scheduler is included as part of the control program to control the flow of jobs and allocate the I/O devices required. Two forms of job scheduling are available.

With sequential scheduling the jobs are carried out in the order they are presented in the input stream to the computer.

With priority scheduling a summary of the input job stream is stored on a direct access device and jobs are carried out in order of priority (as specified in the JOB control statement). Any hold-up in the execution of a program, due, for example, to a delay in mounting a volume, will cause the job scheduler to select the next job available, in order of priority, and the revert back to the higher priority job when it is ready.

Supervisor

The supervisor is a set of subroutines, included in the control program, for transferring control of the central processing unit of the computer from one program to another and co-ordinating I/O operations. Initialization and termination of all programs described in this publication are achieved using the standard method given in IBM System/360 Operating System: Control Program Services.

Data Management

This sub-section is a summary of data management facilities. Full details are given in IBM System/360 Operating System: Data Management.

Data Sets: Data is usually stored on I/O devices and is only brought into main storage for processing. It is organized into data sets. These are collections of records that are logically related (for example, a set of test readings).

System/360 Operating System allows a data set to be identified and accessed by symbolic name only, without any reference to its location on the storage device. To do this the operating system builds a catalog of data set location against name. This catalog resides on one or more direct access volumes. A volume is one complete physical unit of storage such as a tape reel or a disk pack. It may contain a number of data sets, or alternatively one data set may stretch over a number of volumes. Data sets are created using DD statements.

Data Control Blocks: The operating system must be provided with information describing the characteristics of a data set before the data set can be processed. This information is assembled in the data control block associated with each data set. Data control blocks are automatically created for each data set that is to be processed by the program, and are completed from two sources:

1. Any information provided in the program is included first.

2. Information provided by the DD statement is then included, but this cannot over-ride any information stated in the program.

In the case of an existing data set, further information is taken from the data set label. Again, this cannot over-ride previously inserted information. Any DCB information provided by the programmer is checked by an appropriate routine to ensure its validity and to assign default values.

Data Set Labels: Data set labels, if requested by the programmer in the DD statement, are created by the operating system to store information relevant to the data set such as name and retention period. They can supplement information in the data control block and serve as identifiers during accessing. They are positioned at the beginning and end of the data set.

Records and Blocks: Records are the smallest items of data which can be read or written separately. Their length can be specified as fixed, variable or undefined. The unit of length is known as a byte, which is normally equivalent to one character. For mechanical reasons it is necessary to have a fixed length gap between each record. This means that the smaller the average length of the records so the smaller the amount of information that can be stored in a given area of storage. To conserve space a number of records can be grouped together to form a block, which is treated as a single record for I/O operations. The complete block is read into main storage and then unblocked for the required record to be processed. Record format and blocksize are defined in the data control block. For fixed length records blocksize must be a multiple of record length. This multiplication factor is known as the blocking factor.

A control character can be specified for inclusion in each record of a data set. This selects carriage control when the data set is printed, or stacker when the data set is punched.

Data Set Organization: According to how they are going to be used, records can be organized within the data set in a number of ways.

Sequential organization is a feature of I/O devices such as magnetic tapes. To access a particular record the data set must be read sequentially until the record is found. This is satisfactory for many applications where a large proportion of the records will be required on each run but could be time-consuming where data is being accessed randomly.

To avoid reading each record in turn the indexed sequential method is often employed, in which the location of the required record is found from an index at the beginning of its data set. On a disk pack the specification of a record location is broken down into two levels - cylinder and track. Each level has its own index. With large data sets up to three levels of master index can also be used. Overflow areas are provided for the primary storage area so that insertions can be made.

Alternatively, a data set can be partitioned into blocks of identical format called members. A directory is built up at the beginning of the data set so that each member can be accessed independently by specifying its name as a suffix to the data set name. This form of data set is described as a library.

Access Language: Two access languages are available to store and retrieve records. The queued access language provides a full range of buffering and blocking facilities to improve processing efficiency. It can only be used with sequential and indexed sequential data sets.

The basic access language gives the programmer more direct control over the I/O device but does not provide buffering and blocking facilities. These must be constructed by the user (see IBM System/360 Operating System: Control Program Services).

Access Methods: The data set organization and access language used are combined to fully describe the method of handling a data set, for example, Queued Sequential Access Method, Basic Partitioned Access Method, etc. The access method is specified in the data control block.

Input/Output Devices: Data can be stored on a number of input/output devices depending, among other things, on the method of data set organization required. The devices most commonly used in scientific and engineering installations are:

Card readers and punches Printers (output only) Paper tape devices Magnetic tape devices	All data handled by these devices is sequentially organized.
Disk storage devices Data cell storage devices Drum storage devices	These are known as direct access devices and can be used for sequential, indexed sequential or partitioned organization.

A console typewriter is used for direct two-way communication between the operator and the operating system.

Areas of main storage known as buffers are used to provide overlapping of reading, writing and processing operations. The transfer of data between main storage and I/O devices is controlled through units known as channels.

Processing Programs

In addition to the control program, a number of processing programs may be included in the operating system depending on the requirements of the installation. To carry out a job that contains a source program written in ALGOL the following processing programs are required:

1. ALGOL compiler

2. Linkage editor

The ALGOL compiler processes the source program to translate it into machine language. The translated source program (known as the object module) is then processed by a linkage editor to combine any routines required from the ALGOL library (see Appendix A). The result of these two operations (known as the load module) is then loaded into main storage and control is passed to the load module so that it can be executed. The basic flowchart for handling an ALGOL source program is shown in Figure 1.

ALGOL Compiler

This processing program is available for the F level of main storage size, and requires a minimum of 44K bytes. If extra storage capacity is provided it is used to increase compiler capacity (see Figure 6).

Initialization and Termination: The standard method is used for initialization and termination of the compiler (see "Supervisor"). At the end of the compilation one of the following return codes is generated:

0 meaning normal conclusion. Object module has been generated unless both the NODECK and NOLOAD options (see Appendix E) are specified in the invoking statement. No diagnostic messages have been listed.

4 meaning object module has been generated unless both the NODECK and NOLOAD options are specified. Only warning diagnostic messages (severity code W) have been listed.

12 meaning process has been completed but a complete object module could not be generated due to a serious error. Diagnostic messages (severity codes S and possibly W) have been listed.

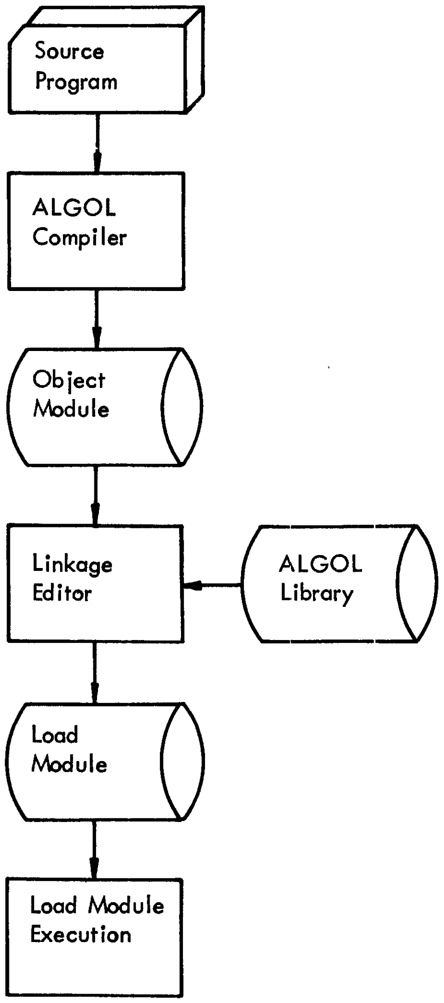


Figure 1. Basic flowchart for handling an ALGOL program.

16 meaning process has been terminated abnormally due to a terminating error. A complete object module could therefore not be generated. Diagnostic messages (severity codes T and possibly W and S) have been listed. The severity codes are described in Appendix F.

Output: A successful compilation of an ALGOL source program produces the following output:

- An object module (described in Appendix D) which can be:
  - Included in a data set for use as input to the linkage editor (optional).
  - Included in another data set to give some other form of output, such as a card deck (optional).

- Information listings (described in Section 3).

Linkage Editor

The linkage editor is a standard processing program used for all languages accepted by the System/360. For ALGOL, it is used to include routines from the ALGOL library. It also has a wide range of optional functions, and is available for two levels of main storage size - E level (where it requires 15K or 18K bytes) and F level (where it requires 44K or 88K bytes). A full description is contained in IBM System/360 Operating System: Linkage Editor.

Initialization and Termination: The standard method is used for initialization and termination of the linkage editor (see "Supervisor"). At the end of the linkage editing one of the following return codes is generated:

0 meaning normal conclusion. A load module has been produced.

4 meaning a load module has been produced but a severity 1 error, which may cause an error at execution time has been detected and listed.

8 meaning a load module has been produced but a severity 2 error, which may cause an abnormal termination at execution time, has been detected and listed.

12 meaning a load module has been produced but a severity 3 error, which will cause an abnormal termination at execution time, has been detected and listed.

16 meaning process has been terminate abnormally. A severity 4 error has been listed.

Output: The following output can be produced by the linkage editor:

- A load module data set, stored on the output library SYSLMOD.
- Information listings (described in Section 3).

Load Module Execution

The load module produced by the linkage editor is loaded into main storage by the supervisor. When the loading operation is complete, the supervisor passes control to the load module, which is then executed.

Initialization and Termination: The standard method is used for initialization and termination of the load module (see "Supervisor"). At the end of the execution, one of the following return codes is generated:

0 meaning normal execution has been performed.

4 meaning execution has been abnormally terminated due to an error. A diagnostic message has been listed.

Output: The following output is produced by a successful execution of a load module:

- Results, etc., as specified by the programmer.
- Information listings (described in Section 3).

MACHINE CONFIGURATION

To successfully carry out a job containing a source program written in ALGOL, a certain minimum machine configuration must be available. This is:

- A System/360 Model 30, 40, 50, 65, 75 or 91 with the scientific instruction set. Main storage size depends on the program being executed.
  - For compilation, at least 64K bytes.
  - For linkage editing, at least 32K bytes.
  - For load module execution, variable, depending on the size and arrangement of the source program.

  These figures include the space used by the control program of the operating system.
- In a minimum configuration, all data sets may use a single direct access I/O device, provided that the total size of the data sets which exist at any one time does not exceed the capacity of the device. A card reader and printer will also be needed, but these do not have to be part of the System/360 configuration.
- A console typewriter may be required for diagnostic messages if there is an error on the data set used for output listings, and also to allow direct two-way communication between the operator and the operating system.

SECTION 2: SOURCE PROGRAM HANDLING

This section explains the job control statements which must be provided with each source program. These statements can either be written for each job, or a standard job control procedure can be written and cataloged in the operating system for use with a range of jobs.

Using such a cataloged procedure minimizes the number of job control statements that must be supplied by the programmer with each job. Therefore IBM provides:

- Three basic cataloged procedures for use with ALGOL.
- The means to temporarily over-ride these procedures if the user requires different or additional system support to that provided.
- The means for the user to modify permanently the IBM-supplied cataloged procedures or to write his own procedures and catalog them for permanent reference.

In the statement formats used in this section upper-case words must be coded exactly as they appear; lower-case words are used to indicate where the programmer must supply information according to his own requirements.

IBM-SUPPLIED CATALOGED PROCEDURES

The three cataloged procedures for ALGOL which are supplied by IBM are:

ALGOFC	compilation only
ALGOFCL	compilation and linkage editing.
ALGOFCLG	compilation, linkage editing and execution.

To invoke these cataloged procedures, the programmer must supply the following job control statements:

1. A JOB statement to indicate the start of the job.
2. An EXEC statement indicating the name of the cataloged procedure to be used.
3. DD statements indicating the location of the source program and, for execution time, the data sets used or created by the load module.

The following text indicates the minimum contents of these statements. For requirements beyond this, reference should be made to Appendix E.

Compilation

The cataloged procedure to compile a source program is ALGOFC. The job control statements used in this cataloged procedure are shown in Appendix B. The following statements can be used to invoke the ALGOFC cataloged procedure:

```
//jobname    JOB
//           EXEC  ALGOFC
//SYSIN      DD    {* or parameters defining an
                      input data set containing
                      the source program }
```

where "jobname" is the name of the job. If DD * is used then the source program must follow immediately afterwards in the input stream. For sequential scheduling, the source program must then be followed by a delimiter statement (/*).

If more than one source program is to be compiled in the same job, all job control statements except the JOB statement must be repeated for each source program.

A sample deck of job control statements to compile an ALGOL source program is shown in Figure 2.

Compilation and Linkage Editing

The cataloged procedure to compile an ALGOL source program and linkage edit the resulting object module is ALGOFCL. The job control statements used in this cataloged procedure are shown in Appendix B. The following statements can be used to invoke the ALGOFCL cataloged procedure:

```
//jobname    JOB
//           EXEC  ALGOFCL
//SYSIN      DD    {* or parameters defining an
                     input data set containing
                     the source program }
```

where "jobname" is the name assigned to the job. If DD * is used then the source program must follow immediately afterwards in the input stream. For sequential scheduling, the source program must then be followed by a delimiter statement (/*).

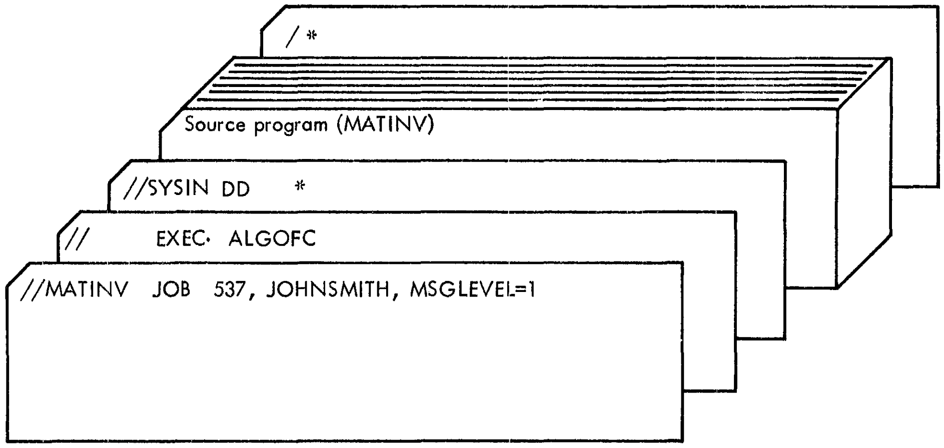


Figure 2. Sample deck for using ALGOFC cataloged procedure with a single source program. This job compiles the MATINV source program used in Example 1 of Appendix E.

If more than one source program is to be processed in the same job, then all job control statements except the JOB statement must be repeated for each source program.

If it is required to keep a load module for use in a later job (as in the case when the load module is a precompiled procedure), then the SYSLMOD DD statement in the cataloged procedure must be over-ridden to specify a permanent data set. This has to be done for each load module that is kept. The over-riding statement is placed at the end of the job step to which it applies, and has the form:

```
//LKED.SYSLMOD  DD  DSNAME=dsname(member),
                    DISP=(MOD,KEEP)
```

where "dsname" is the name of a partitioned data set and "member" is the member name assigned to the load module on the partitioned data set.

A sample deck of job control statements to compile and linkage edit two source programs is shown in Figure 3.

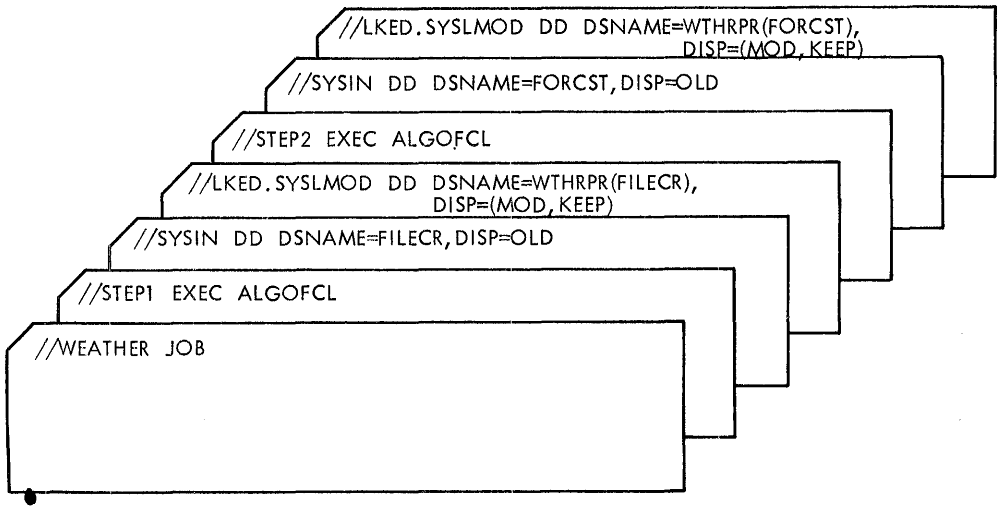

//LKED.SYSLMOD DD DSNAME=WTHRPR(FORCST),
DISP=(MOD,KEEP)
//SYSIN DD DSNAME=FORCST,DISP=OLD
//STEP2 EXEC ALGOFCL
//LKED.SYSLMOD DD DSNAME=WTHRPR(FILECR),
DISP=(MOD,KEEP)
//SYSIN DD DSNAME=FILECR,DISP=OLD
//STEP1 EXEC ALGOFCL
//WEATHER JOB

Figure 3. Sample deck for using ALGOFCL cataloged procedure with two source programs. These two job steps compile and linkage edit the two source programs used in Example 3 of Appendix E. Both source programs have been previously stored on intermediate I/O devices.

Compilation, Linkage Editing and Execution

The cataloged procedure used to compile an ALGOL source program, linkage edit the resulting object module, and execute the load module produced by the linkage editor is ALGOFCLG.

The statements used in this cataloged procedure are shown in Appendix B. The following statements can be used to invoke the ALGOFCLG cataloged procedure:

```
//jobname    JOB
//JOBLIB     DD  DSNAME=dsname1, DISP=OLD
//           EXEC  ALGOFCLG
//SYSIN      DD   {* or parameters defining an
                     input data set containing
                     the source program }
//GO.ALGLDD02  DD  DSNAME=dsname2
                                  .
                                  .
                                  .
//GO.ALGLDD15  DD  DSNAME=dsname15
```

where "jobname" is the name assigned to the job. "dsname1" is the name of a data set that contains a precompiled procedure (see Section 4) which is called by the load module being executed. The DD statement containing dsname1 need not be used if no precompiled procedure is used.

For a description of the correct use of the JOBLIB DD statement when more than one precompiled procedure is used in a job, or when a precompiled procedure resides on more than one data set, see "Data Set Concatenation" in Appendix E.

"dsname2"... "dsname15" are the names of input data sets required by the load module at execution time and output data sets to be created at execution time. In addition, a data set for printed output (ddname SYSPRINT) is supplied by the cataloged procedure, and a data set for input only can be specified by using the following statement after the invoking sequence just given.

```
//GO.SYSIN  DD  {* or parameters defining an
                   input data set }
```

If DD * is used then the source program must follow immediately afterwards in the input stream. For sequential scheduling, the source program must be followed by a delimiter statement (/*).

If more than one source program is to be processed and executed in the same job, then all job control statements except the JOB statement and the JOBLIB DD statement must be repeated for each source program.

A sample deck of job control statements required to compile, linkage edit and execute three source programs is shown in Figure 29.

Over-riding Cataloged Procedures

The programmer can change any of the statements in a cataloged procedure, except the name of the program in an EXEC statement.

These over-riding conditions are temporary, and will be in effect only until the next job step is started. The following text describes methods of temporarily modifying existing parameters and adding new parameters to the EXEC and DD statements used in the cataloged procedures. The full list of parameters available to the ALGOL programmer for these statements, and detailed explanations of the parameters, is given in Appendix E. The EXEC and DD statements used in the IBM-supplied cataloged procedures are shown in Appendix B.

Over-riding EXEC Statements

In the EXEC statement, the programmer can change or add any of the keyword parameters by using the following format:

keyword. procstep=option

where:

"keyword" is the parameter to be changed in, or added to, the specified procedure job step: either TIME, COND, PARM or ACCT.

"procstep" is the procedure job step in which the change or addition is to occur: either ALGOL, LKED or GO.

"option" is the new option required.

For example, if the EXEC statement used to invoke the ALGOFCLG cataloged procedure was written as:

```
// EXEC  ALGOFCLG,PARM.ALGOL=DECK,
//                PARM.LKED=XREF,
//                COND.GO=(3,LT,ALGOL)
```

then the following changes would be made to the ALGOFCLG cataloged procedure:

1. In the PARM parameter of the job step ALGOL, the option DECK would be used instead of the default option NODECK (assuming that the standard default NODECK was not changed at system generation). Over-riding this option will not affect the other default options assumed for this parameter.
2. In the job step LKED, the option XREF is specified for the PARM parameter. Since the options specified in the cataloged procedure were XREF, LIST and LET, this statement has the effect of deleting the options LIST and LET since they were not default options.
3. In the job step GO, the COND parameter code is changed from 5, as it appears in the cataloged procedure, to 3. In this example, the code 3 causes the job step GO to be bypassed if a warning message is generated during the job step ALGOL. Note that although the other options (LT and ALGOL) are not to be altered, the entire parameter being modified must be respecified.

If "procstep" is not specified when over-riding a multi-step cataloged procedure, the operating system makes the following assumptions:

- COND and ACCT parameters apply to all procedure job steps.
- A PARM parameter applies to the first procedure job step and any options already specified in the PARM parameters for the remaining procedure job steps are cancelled.
- A TIME parameter specifies the computing time for the entire job and any options already specified in the TIME parameters for individual procedure job steps are cancelled.

Over-riding DD Statements

An additional DD statement is used in the invoking sequence for each DD statement in the cataloged procedure that is to be over-ridden. The following format is used:

```
//procstep.ddname  DD  parameter-list
```

where:

"procstep" is the procedure job step containing the DD statement to be over-ridden: either ALGOL, LKED or GO. If "procstep" is omitted then the first procedure job step is assumed.

"ddname" is the name of the DD statement to be over-ridden.

"parameter-list" is the list of parameters that are being added or changed. In both cases the whole parameter must be specified. Unchanged parameters in the original statement need not be specified. For example, the statement:

```
//ALGOL.SYSLIN  DD  SPACE=(400,(80,10))
```

will change the SPACE parameter of the SYSLIN DD statement in the ALGOL job step so that space will be allocated for 80 physical records instead of 40.

DD statements that are used to over-ride other DD statements in the cataloged procedures must be placed immediately after the EXEC statement invoking the cataloged procedure, and must be in the same order as their corresponding DD statements in the cataloged procedures.

Adding DD Statements

Complete, new DD statements that are to be added to the cataloged procedure use the same format as over-riding DD statements. The "ddname" specified must not exist in the job step specified by "procstep". These new DD statements must follow immediately after the over-riding DD statements which apply to the same procedure job step.

USER-WRITTEN PROCEDURES

The information required by the programmer to write his own job control procedures is given in the following text, and in Appendix E. Cataloging user-written procedures, or permanently modifying the IBM-supplied cataloged procedures, is accomplished using the IEBUPDAT utility program, described in <u>IBM System/360 Operating System: Utilities</u>. The statements required in user-written procedures are:

- An EXEC statement to invoke the program.
- DD statements to define the data sets used by the program.

<u>Compilation</u>

Invoking Statement

The ALGOL compiler consists of ten load modules contained in the link library, SYS1.LINKLIB, of the operating system. The compiler is activated

by invoking its first load module, named ALGOL, which then internally invokes the other load modules of the compiler.

The usual method of invoking the compiler is by means of an EXEC statement of the form:

```
//stepname  EXEC  PGM=ALGOL
```

where "stepname" is the name assigned to the job step (optional).

Other EXEC statement parameters may be included if required (see Appendix E).

(A method of dynamically invoking the compiler within a job step, by means of the CALL, LINK, XCTL or ATTACH macro-instructions, is described in Section 4.)

Data Sets Used

The data sets used in the compilation process are illustrated in Figure 4, and described in Figure 5. These data sets must be specified by the programmer with suitable DD statements.

Blocksize DCB information may be specified by the user for SYSIN, SYSLIN, SYSPRINT and SYSPUNCH. The maximum blocking factor depends on the main storage size available (see Figure 6). Record length is fixed at 80 bytes for SYSIN, SYSLIN and SYSPUNCH, and 91 bytes for SYSPRINT.

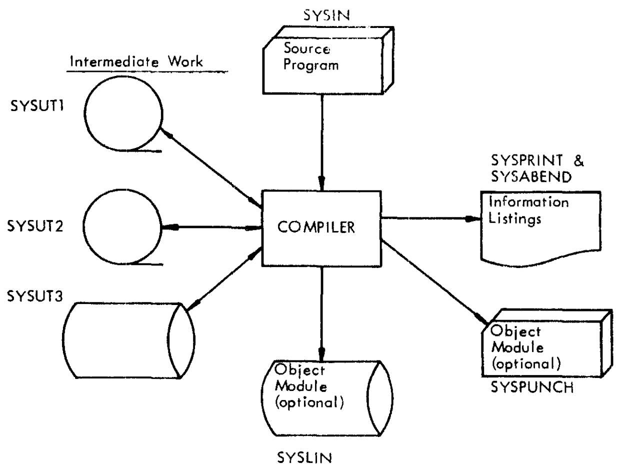


Figure 4. Flowchart showing data sets used by the compiler.

The space required for the compiler data sets depends on the size and structure of the source program, however it can be assumed that only in rare cases will the object module exceed four times the source program and usually much less will be required.

Purpose	Standard ddname	Devices required
For ALGOL source program	SYSIN	Card reader*
For object module to be used by linkage editor	SYSLIN	Direct access or magnetic tape
For compilation listings	SYSPRINT	Printer*
For object module (copied from SYSLIN)	SYSPUNCH	Card punch*
For the control program dump	SYSABEND	Printer*
For intermediate compiler working	SYSUT1	Direct access or magnetic tape
For intermediate compiler working	SYSUT2	Direct access or magnetic tape
For intermediate compiler working	SYSUT3	Direct access

* Some form of intermediate storage, such as magnetic tape, may be used to reduce I/O delay for the central processing unit.

Figure 5. Data sets used by the ALGOL compiler.

Also, as a rough estimate, SYSUT1, 2 and 3 must each be large enough to contain the number of valid characters in the source program.

SYSABEND is used for control program listings (see Section 3).

Processing of all data sets by the compiler is independent of the I/O device used except for the intermediate work data sets. These require magnetic tape or direct access devices.

Linkage Editing

Invoking Statement

The linkage editor is usually invoked with an EXEC statement of the form:

```
//stepname  EXEC  PGM=IEWL
```

where "stepname" is the name assigned to the job step (optional).

Other EXEC statement parameters may be included if required (see Appendix E). IEWL specifies the highest-level linkage editor in the installation's operating system.

(A method of dynamically invoking the linkage editor within a job step, by means of the CALL, LINK, XCTL or ATTACH instructions, is described in Section 4.)

Main storage size in bytes at which changes occur	Maximum blocking factor			
	SYSIN	SYSPRINT	SYSLIN	SYSPUNCH
45056 (44K)	5	5	5	1
51200 (50K)	5	5	5	5
59392 (58K)	5	5	5	5
67584 (66K)	5	5	5	5
77824 (76K)	5	5	5	5
90112 (88K)	20	20	40	20
104448 (102K)	20	20	40	20
120832 (118K)	20	20	40	20
139264 (136K)	20	20	40	20
159744 (156K)	20	20	40	20
184320 (180K)	40	40	40	40
212992 (208K)	40	40	40	40

Figure 6. Effect on compiler data sets if more than 44K bytes of main storage is available. The capacity of internal tables in the compiler is increased at each of the main storage sizes listed in this table, allowing, for example, a larger number of identifiers to be included in the source program. Therefore to get optimum performance, the user is recommended to use this list when specifying main storage size available to the compiler.

Data Sets Used

The data sets used by the linkage editor (see Figures 7 and 8) must be defined by the programmer with suitable DD statements.

Blocksize DCB information may be specified by the user for SYSLIN and SYSPRINT if the F level linkage editor is being used. Maximum blocking factor is 5 when 44K bytes of main storage size is available, and 40 when 88K bytes is available. Record length is fixed at 80 bytes for SYSLIN and 120 bytes for SYSPRINT.

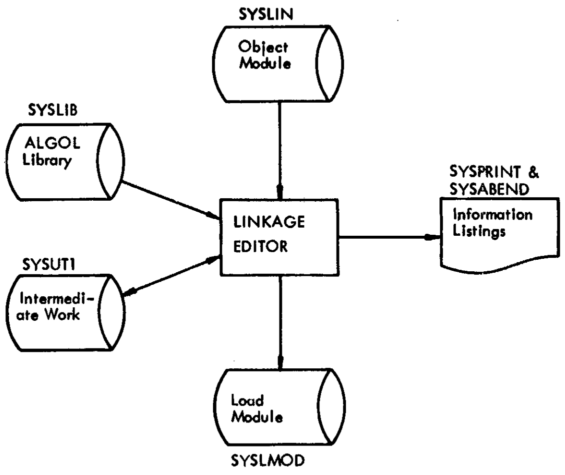


Figure 7. Flowchart showing data sets used by the linkage editor.

SYSABEND is used for control program listings (see Section 3).

Load Module Execution

Invoking Statement

The usual method of invoking the load module generated by the linkage editor is with an EXEC statement of the form:

```
//stepname EXEC PGM=member-name
```

Purpose	Standard ddname	Devices used
For object module input	SYSLIN	Direct access or magnetic tape
For load module output, stored as a member of a partitioned data set	SYSLMOD	Direct access
For ALGOL library, SYS1.ALGLIB. A partitioned data set containing routines in load module form	SYSLIB	Direct access
For linkage editing listings	SYSPRINT	Printer*
For intermediate linkage editor working	SYSUT1	Direct access or magnetic tape
For the control program dump	SYSABEND	Printer*

* Some form of intermediate storage, such as magnetic tape, may be used to reduce output delay for the central processing unit.

Figure 8. Data sets used by the linkage editor.

where "stepname" is the name assigned to the job step (optional).

"member-name" indicates the name of the partitioned data set member which contains the load module. This name is specified by the programmer in the SYSLMOD DD statement for the linkage editor. Other EXEC statement parameters may be included if required (see Appendix E).

(A method of dynamically invoking the load module within a job step, by means of the CALL, LINK, XCTL or ATTACH macro-instructions is described in Section 4.)

Data Sets Used

Up to 16 data sets for use at execution time may be specified by the programmer in the ALGOL source program by using the appropriate data set number. The numbers used and the corresponding names of their DD statements are listed below.

Data set number used in ALGOL source program	Corresponding ddname
0	SYSIN
1	SYSPRINT
2	ALGLDD02
3	ALGLDD03
4	ALGLDD04
5	ALGLDD05
6	ALGLDD06
7	ALGLDD07
8	ALGLDD08
9	ALGLDD09
10	ALGLDD10
11	ALGLDD11
12	ALGLDD12
13	ALGLDD13
14	ALGLDD14
15	ALGLDD15

Any reference to a data set number by an I/O procedure within an ALGOL source program is translated into a reference to a data control block using the corresponding ddname. It is the responsibility of the programmer to supply the DD statements which correspond to the data set numbers used in the ALGOL source program.

The execution time data sets are illustrated in Figure 9 and described in Figure 10. For ALGLDD02 to ALGLDD15, case 1 in the column showing device used, applies if the source program contains any of the following:

- A backward repositioning specification by the procedures SYSACT4 or SYSACT13 for this data set.
- Both input and output procedure statements for this data set.
- Procedure statements which prevent the compiler from recognizing whether either of these applies; for example, if the data set number or SYSACT function number is not an integer constant or if a precompiled procedure is used.

If the source program has already been compiled and linkage edited in a previous job, then the data set on which it has been stored (in load module form) must be concatenated to SYS1.LINKLIB. Data sets containing precompiled procedures called by the source program (see Section 4) must also be concatenated to SYS1.LINKLIB.

If the programmer specifies a TRACE, TRBEG or TREND option in the EXEC statement of the execution job step, the semicolon count (see Section 3) is stored intermediately on a data set with the ddname SYSUT1. The programmer must supply a corresponding DD statement if he uses this option. The semicolon count is converted to external form and transferred to the SYSPRINT data set as soon as the execution ends either by reaching the logical end of the source program or due to an error.

The space required for the semicolon count is:

For the main heading	6 bytes
For each semicolon	2 bytes
For each call of a precompiled procedure	12 bytes
For each physical record on SYSUT1	4 - 6 bytes

System/360 ALGOL permits data to be temporarily stored on and retrieved from external devices without conversion, using the ALGOL I/O procedures PUT and GET. If the programmer uses this facility in his source program, then he must supply a DD statement with the ddname SYSUT2. The device specified by this statement for storing such intermediate data should be a direct access device to guarantee reasonable performance, though programming is performed independently between magnetic tape and direct access devices. All data passed by a single PUT is

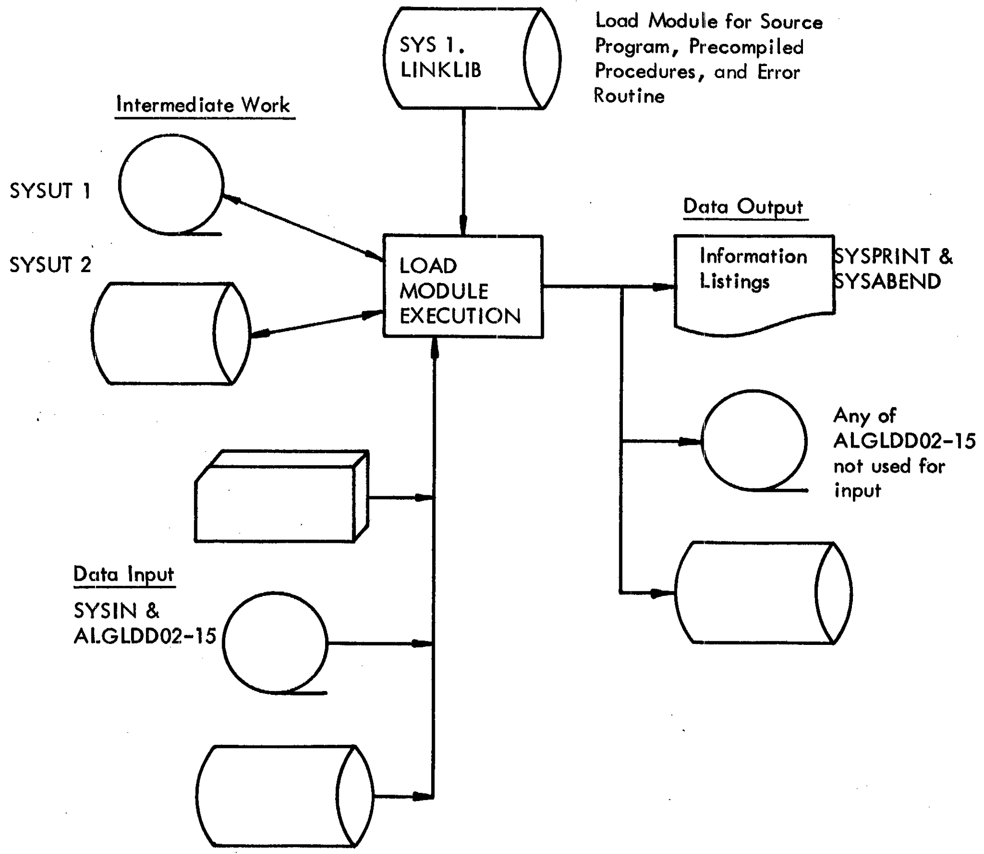


Figure 9. Flowchart showing data sets used at load module execution. The data input and output requirements are variable.

stored as one record. This record will be as long as the data passed, plus 8 bytes. The maximum record length accepted is 2048 bytes.

The DCB information which may be specified by the user for execution time data sets is blocksize, record format and record length, except for the trace and PUT/GET data sets (ddnames SYSUT1 and SYSUT2) for which only blocksize may be specified (up to a maximum of 2048 bytes).

For information not provided, default values will be inserted by a routine in the ALGOL library. In particular, blocksize is assumed as 2048 bytes for SYSUT1 and SYSUT2 if none is specified.

SYSABEND is used for control program listings (see Section 3).

	Standard ddname	Device Used
For data input to load module	SYSIN	Any input device
For execution time listings and data output	SYSPRINT	Printer*
For data input or output	ALGLDD02 : ALGLDD15	1. Direct access or magnetic tape 2. Any
For intermediate storage of semicolon counter when TRACE is specified	SYSUT1	Direct access or magnetic tape
For temporary storage when PUT is specified	SYSUT2	Direct access or magnetic tape
For the control program dump	SYSABEND	Printer*

* Some form of intermediate storage, such as magnetic tape, may be used to reduce I/O delay for the central processing unit.

Figure 10. Data sets used at execution time.

SECTION 3: INFORMATION LISTINGS

To assist the programmer to find the cause of any faults in the processing or execution of his program, various forms of information listings are produced for the compilation, linkage editing and execution operations. Some of these listings are optional. Examples are illustrated in Figures 11 to 16.

CONTROL PROGRAM LISTINGS

All three operations may produce listings generated by the control program. These are described in IBM System/360 Operating System: Messages, Completion Codes, and Storage Dumps. The ABEND macro-instruction for specifying the main storage dump is described in IBM System/360 Operating System: Control Program Services.

COMPILATION LISTINGS

A successful compilation of an ALGOL source program produces the following information listings:

- Job control statement information according to which MSGLEVEL option was specified in the JOB statement.
- The source program supplemented by a count of the semicolons occurring in the program (optional).
- A table giving details of all identifiers used in the program (optional).
- Any warning diagnostic messages.
- Information on main storage requirements at execution time.

If a serious diagnostic message is produced (meaning that object module generation has ended) then the source program and identifier table listings will be printed in full if they have been requested, but the information on main storage requirements will not be printed. If a terminating diagnostic message is produced then the source program and identifier table listings can be printed only as far as they have been produced.

Source Program

If the SOURCE option has been specified, the source program is transferred by the compiler to an output data set in order to be listed by a printer. Unless NOTEST has been specified, this source program is supplemented by a semicolon count, which is referred to in the diagnostic messages to help localize errors.

The compiler generates this semicolon count when scanning the source program, by counting all semicolons occurring in the source program outside strings, except those following the delimiter 'COMMENT'. The value of this semicolon count at the beginning of each record of the source program is printed at the left of that record. It is assigned by the compiler in order to have a clear, problem-oriented reference. Any reference to a particular semicolon number refers to the segment of source program following the specified semicolon, for example, the semicolon number 5 refers to the program segment between the fifth and sixth semicolons.

Identifier Table

If the SOURCE option has been specified, a list of all identifiers declared or specified within the source program is transferred by the compiler to the output data set for printing after the source program listing. This identifier table gives information about the characteristics and internal representation of all identifiers. The identifiers are grouped together within the identifier table according to their scopes.

All blocks and procedure declarations within the source program are numbered according to the order of occurrence of their opening delimiters 'BEGIN' or 'PROCEDURE'. Therefore, if the body of a procedure declaration is a block, then usually this block has the same number as the procedure declaration itself. These numbers are called program block numbers (even if they belong to a procedure declaration and not to a block).

Each line in the table contains entries for up to three identifiers and the line begins with the number of the program block in which the identifiers were declared or specified, the value of the semicolon count at the commencement of the program block, and the number of the immediately surrounding program block. Each identifier entry contains:

1. The external name of the identifier as appearing in the source program. Space for six characters is provided and, if necessary, the identifier is truncated.
2. The type key, as described below.
3. The number of dimensions (for array identifiers), components (for switch identifiers) or parameters (for procedure identifiers). This position is blank for all other types of identifiers.
4. The displacemert for the quantity denoted by the identifier, as explained below.

The type key consists of five characters denoting the type characteristics of the identifier. These characters are as follows (b represents blank):

In first position:	R when real I when integer B when Boolean b when anything else
In second position:	L when label S when switch T when string (text) b when anything else
In third position:	A when array P when procedure b when anything else
In fourth position:	N when formal parameter called by name V when formal parameter called by value b when declared identifier (not formal parameter)
In fifth position:	C when precompiled (code) procedure b when anything else

Examples of these are:

For a real variable	Rbbbb
For a Boolean array	BbAbb
For a formal parameter specified integer procedure called by name	IbPNb
For a precompiled procedure	bbPbC

The displacement is in hexadecimal form and has the following meaning:

- For all identifiers denoting simple variables, arrays and formal parameters, it is the relative position of their values in the data storage area, as described below.

- For all identifiers denoting labels, procedures and switches (if not specified as formal parameters), it is the relative position of the corresponding entry in the label address table, as described below. This position is known as the label number (LN).

The space allocated to each identifier is as follows:

For formal parameters: 8 bytes

For Boolean identifiers: 1 byte

For integer identifiers: 4 bytes

For real identifiers: 4 bytes when SHORT is specified; 8 bytes when LONG is specified.

For arrays: see storage mapping function below.

At execution time, for each program block, a data storage area (DSA) is created dynamically at each entry of the program block and is released when leaving it. The lengths of the data storage area and the relative positions of all data contained in them are determined by the compiler. These relative positions, together with the program block numbers, uniquely identify the quantities of an ALGOL program. Two forms are used according to whether the SHORT or LONG option was specified in the invoking statement.

The data storage area of a program block contains locations for:

1. The values of simple variables
2. The storage mapping functions of arrays (see below)
3. In the case of formal parameters, the type characteristics and addresses of the actual parameters
4. Intermediate results, addresses, etc.

A label address table is created by the compiler and transferred to the object module. In general it is used at execution time to load a branch register before any branch is performed. It contains addresses corresponding to:

1. Library modules required
2. Labels
3. Procedure declarations
4. Switch declarations
5. Internal branches ('IF', 'FOR', etc.)

The storage mapping function (SMF) describes the storage layout of an array. The storage that the SMF requires in the DSA can be calculated from:

$$s = 4(d + 5) + X$$

where, s = number of bytes in storage mapping function

d = number of dimensions in array

$X = \begin{cases} 4 \text{ if LONG is specified and } d \\ \text{is an even number, } 0 \text{ otherwise} \end{cases}$

Diagnostic Messages

During the compilation as many programming errors as possible are detected and appropriate diagnostic messages are produced to help the programmer to identify them. Diagnostic messages are caused by:

1. Programming errors. These are detected and reported by the compiler as far as they do not depend on the dynamic flow of the program. Programming errors depending on the dynamic flow of the program are detected and reported by the load module.

2. Violations of capacity limitations. Such violations are detected and reported by the compiler, where possible. Those which cannot be detected at compile time are detected and reported by the load module at execution time.

3. I/O errors caused by malfunction of channels or external devices, are reported when they occur.

4. Control card errors not detected by the job scheduler.

5. Program interrupts

The diagnostic messages are transferred to the output data set to be listed by a printer. Appendix F contains a list of the messages that may be produced by the ALGOL compiler.

Storage Requirements

Following the diagnostic messages, the compiler transfers information about the execution time storage requirements to the output data set if the compilation finished successfully. This information gives no exact storage estimate of the object module execution because the storage allocation for data is performed dynamically at execution time and depends on the flow of control through the object module and on the amount of data at execution time.

For example, the data storage area belonging to a program block is allocated only as long as that program block is active. In the case of recursive procedures more than one generation of the corresponding data storage area may be required. The storage needed for the array is not contained in a data storage area and depends on the execution time values of the bounds of the array.

Nevertheless, a programmer knowing the structure of his program may gain rough storage estimates from the following information given by the compiler.

1. Main storage required by the object module, including tables and constant pool.

2. A list of the main storage requirements of all data storage areas. This list consists of one entry for each program block, containing the program block number, and the number of bytes required for the corresponding data storage area.

LINKAGE EDITING LISTINGS

A successful linkage editing can produce the following information listings:

- Job control statement information according to which MSGLEVEL option was specified in the JOB statement.

- Disposition data, listing the options specified and the status of the load module in the output library.

- Diagnostic messages (severity code 1).

- A cross-reference table of the load module, or alternatively, a module map (both optional).

If a diagnostic message of severity code 2 or 3 is produced then the other information listings might not be produced. If a diagnostic message of severity code 4 is produced then the other information listings will not be produced.

Diagnostic Messages

A description of the diagnostic messages that may be produced by the linkage editor is contained in Appendix F.

Module Map

If MAP is specified in the invoking statement for the linkage editor, then a module map is transferred to the output data set to be listed by a printer. The module map shows all control sections (the smallest separately relocatable units of a program) in the load module and all entry names (to routines in the ALGOL library) in each control section. The control sections are arranged in ascending order according to their origins (which are temporary addresses assigned by the linkage editor prior to loading for execution). The entry names are listed below the control section in which they are defined. The origins and lengths (in bytes) of the control sections, and the location of the entry names are listed in hexadecimal form. Unnamed control sections are identified by $PRIVATE in the list.

At the end of the module map is the entry address of the instructions with which processing of the module begins. It is followed by the total length of the module, in bytes. Both values are in hexadecimal form.

Cross-Reference Table

If XREF is specified in the invoking statement for the linkage editor, the cross-reference table is transferred to the output data set to be listed by a printer.

The cross-reference table consists of a module map and a list of cross-references for each control section. In the list of cross-references, each address constant that refers to a symbol defined in another control section is listed with its assigned location (in hexadecimal form), the symbol referred to, and the name of the control section in which the symbol is defined.

If a symbol is unresolved after processing by the linkage editor, it is identified by $UNRESOLVED in the list. However, if an unresolved symbol is marked by the never call function, it is identified by $NEVER-CALL.

The entry address and total length are listed after the list of cross-references.

EXECUTION TIME LISTINGS

A successful execution of the load module produces the following information listings:

- Job control statement information according to which MSGLEVEL option was specified in the JOB statement.

- The ALGOL program trace, which is a list of the semicolon numbers assigned by the compiler (optional).

If an error is detected during execution of the load module, additional information listings are printed before the trace: these are;

- A diagnostic message

- The contents of the data storage areas (optional)

Diagnostic Messages

Any error detected at execution time causes abnormal termination. A diagnostic message is produced which is transferred to an output data set to be listed by a printer. The diagnostic messages which may be produced during load module execution are listed in Appendix F.

Data Storage Areas

If DUMP is specified in the invoking statement for the execution operation, the data storage areas (DSA) in main storage are transferred to the output data set to be listed by a printer. They are listed in the reverse order to which they were created.

A DSA is created for each call of a program block (see "Compilation Listings") and exists in main storage as long as the call is effective. The DSA contains:

1. All execution time values of variables declared or specified in the program block except for arrays. The array values are stored separately but are included in the listing because they are referenced by the SMF which is contained within the DSA.

2. Intermediate results (known as the object time stack).

The information listed for each DSA consists of:

- Name of load module
- Program block number
- Description of program block; either BLOCK, PROCEDURE or TYPE PROCEDURE
- The values in the DSA, in batches according to their category, that is, formal parameters, declared identifiers and object time stack, arrays called by value, and declared arrays.

The values are those which exist at the time the error was detected (in hexadecimal form). The displacement in the DSA of the first value in each line is printed at the beginning of each line. This is a six digit hexadecimal number.

For formal parameters, each entry has 16 digits, and in the case of parameters called by name the entry contains an address constant pointing indirectly to the value.

For declared identifiers and the object time stack, the identifier entries are listed first and they can be located using the identifier table if it was listed by the compiler. In the case of a type procedure, the function value is stored at the location of the procedure identifier within the program block defined by the procedure. The object time stack contains various intermediate results and addresses which are not directly related to the identifiers in the source program.

For arrays the length depends on the SMF. The displacement of the SMF in the DSA is given for each array.

In the listings, real values have a length of 8 hexadecimal digits when SHORT is specified and 16 digits when LONG is specified. They are in standard floating point representation. Integer values have a length of 8 hexadecimal digits and are in standard fixed point representation. Boolean values have a length of 2 hexadecimal digits which appear as 00 for 'FALSE' and 01 for 'TRUE'.

An editing routine inserts blanks between each set of 8 digits to improve readability.

ALGOL Program Trace

A program trace, listing the semicolon numbers assigned by the compiler (see "Compilation Listings") in the order the corresponding semicolons were encountered during execution, is transferred to an output data set to be listed by a printer if TRACE, TRBEG or TREND is specified in the invoking statement for the execution. The completeness of the trace depends on the option or options specified (see Appendix E). Only the semicolons actually passed through at execution time are included in the trace.

If a precompiled procedure is used in the program and TRACE is specified, then the semicolon numbers for the procedure are included in the correct position within the program. The appropriate load module name (first four characters only) is inserted at the beginning of the listings and each time a change occurs in the first four characters of the module name.

```
                                          SOURCE PROGRAM

 SC      SOURCE STATEMENT

00000    'BEGIN' 'INTEGER' I; 'REAL' A; 'BOOLEAN' B;'INTEGER' 'ARRAY' IA(/1:5/);
00004    'ARRAY' AR(/0:3,2:8/); 'BOOLEAN' 'ARRAY' BA(/0:1,1:3,3:7/);
00006    'INTEGER' 'PROCEDURE' IP; IP:= I+5;
00008    'REAL' 'PROCEDURE' RP(A); 'VALUE' A; 'INTEGER' A; RP:=A*A;
00012    'PROCEDURE' P(A,B,C); 'BOOLEAN' A; 'REAL' B; 'INTEGER' C;
00016    A:=B<C
00017    I:=1; A:=2.6;
00019    AR(/1,1/):=IP;
00020    AR(/1,2/):=RP(AR(/1,1/));
00021    P(BA(/0,1,3/),A,I);
00022    P(B,AR(/1,2/),IP);
00023    SYSACT(1,8,50); OUTREAL(1,AR(/1,1/));
00025    OUTBOOLEAN(1,BA(/0,1,3/));
00026    OUTBOOLEAN(1,B)   ;
00027    A:=A/0;
00028    'END'
```

Figure 11. Example of Source Program Listing.

```
                                          IDENTIFIER TABLE

PBN SC     PBN      NAME     TYPE   DM DSP      NAME     TYPE    DM DSP       NAME    TYPE   DM DSP
           SURR                     PR LN                        PR LN                       PR LN


001 00000 000       A        R         01C      AR       R A     02 03C       B       B         020
                    BA       B A    03 058      I        I          018       IA      I A    01 024
                    IP       I P    00 070      P          P     03 078       RP      R P    01 074

002 00006 001       IP       I P    00 070

003 00008 001       A        I   V     020      RP       R P     01 074

004 00012 001       A        B   N     018      B        R N        020       C       I   N     028
```

Figure 12. Example of Identifier Table Listing. This corresponds to the program in Figure 11.

```
                                  STORAGE REQUIREMENTS (DECIMAL)

OBJECT MODULE SIZE 1840 BYTES.

DATA STORAGE AREA SIZES

  PBN    BYTES         PBN    BYTES       PBN    BYTES        PBN   BYTES          PBN   BYTES

  001     136          002       32       003       40        004      60
```

Figure 13. Example of Storage Requirements Listing. This corresponds to the program in Figure 11.

```
                                        ----  CROSS REFERENCE TABLE  ----
   CONTROL SECTION                          ENTRY

     NAME     ORIGIN  LENGTH                  NAME    LOCATION      NAME   LOCATION      NAME   LOCATION      NAME   LOCATION

                  00     730
                                            IHIDSTAB      6D8     IHIENTIF      724
   IHISYSCT*     730     5EC
   IHISOREA*     D20     328
                                            IHISORAR      D20     IHISOREL      D30

   IHIIORTN*    2580     B58

                                            IHIIOROQ     2580     IHIIOROP     25AC     IHIIORNX     28C4     IHIIORCL     2B0C
                                            IHIIORCP     2C72     IHIIORGP     2D38     IHIIORCN     2D3C     IHIIOREN     2D76
                                            IHIIOREV     2DCE     IHIIORED     2E40     IHIIORCI     2F44     IHIIORER     2FCC

   LOCATION   REFERS TO SYMBOL   IN CONTROL SECTION

       61C              IHISYSCT         IHISYSCT
       658              IHISOREL         IHISOREA
       660              IHIOBOOL         IHIOBOOL
       D08              IHIIORCL         IHIIORTN

      1F48              IHIFSARB         IHIFSARB
      1F5C              IHIIORCP         IHIIORTN
      1F81              IHIFSARA         IHIFSARA
  ENTRY ADDRESS       IF24
  TOTAL LENGTH        30D8
```

Figure 14. Example of Cross-Reference Table Listing. This is part of the table produced from the program in Figure 11. A Module Map Listing would contain only the list of Control Sections and Entry Names, plus the Entry Address and Total Length Information. Control Sections marked with an asterisk were included from a library during automatic library call.

```
IHI031I  SC=00027  PSW= FF05000F 48005E22    DIVISION BY ZERO, FLOATING POINT

MODULE = GO         PROGRAM BLOCK NUMBER = 001     (BLOCK)

         DECLARED IDENTIFIERS AND OBJECT TIME STACK
000018   00000001  4129999A  0001FF2C  01000000  0001E49C   0001F4A0   0001E4B4   00000014
000038   00000004  02000024  0001E428  0001E430  0001E4A0   00000070   0000001C   00000004
000058   0300003C  0001E408  0001E410  0001E42E  0000001E   0000000F   00000005   00000001
000078   0001E44C  0000581C  0001F560  400058C

         SMF DISPLACEMENT IN DSA =  000058        DECLARED ARRAY
000000   00000000  00000000  00000000  00000000  00000000   00000000   00000000   00000000

         SMF DISPLACEMENT IN DSA =  00003C        DECLARED ARRAY
000000   00000000  00000000  00000000  00000000  00000000   00000000   41600000   42240000
000020   00000000  00000000  00000000  00000000  00000000   00000000   00000000   00000000
000040   00000000  00000000  00000000  00000000  00000000   00000000   00000000   00000000
000060   00000000  00000000  00000000  00000000

         SMF DISPLACEMENT IN DSA =  000024        DECLARED ARRAY
000000   00000000  00000000  00000000  00000000  00000000
```

Figure 15. Example of Error Message and Data Storage Area Listing. This is the listing produced from the program in Figure 11 when the division by zero was encountered.

```
  ALGOL PROGRAM TRACE

MODULE     SEMICOLON NUMBERS

  GO       00001 00002 00003 00004 00005 00006 00008 00012 00017 00018 00019 00007 00020
           00009 00010 00011 00021 00013 00014 00015 00016 00022 00013 00014 00015 00016
           00007 00023 00024 00025 00026 00027
END OF ALGOL PROGRAM EXECUTION
```

Figure 16. Example of Program Trace Listing. This was produced from the program in Figure 11.

CAPACITY LIMITATIONS

In addition to those given in IBM System/360 Operating System: ALGOL Language, the following restrictions must be observed when writing an ALGOL source program:

Number of blocks and procedure declarations (NPB)	≤255
Number of for statements	≤255
Number of identifiers declared or specified in one block or procedure. F is at most twice the number of for statements occurring in that block	≤179-F for type procedures ≤180-F otherwise
Length of letter string serving as parameter delimiter	≤1024 letters when main storage size available is less than 50K, ≤2000 letters otherwise
Length of label identifer	≤1024 characters when main storage size available is less than 50K, ≤2000 characters otherwise
Number of valid characters	≤255K
Number of semicolons in the whole program	≤65535
Number of nested blocks, compound statements, for statements and procedure declarations	≤999
Number of labels declared or additionally generated by the compiler	≤1024

The compiler generates the following additional labels:

For each switch declaration	2
For each procedure declaration	2
For each procedure activation (including function designators)	1
For each 'THEN' and each 'ELSE'	1
For each for statement	at most L + 3 where L is the number of for list elements

Length of constant pool	≤(256 - NPB) x 4096 bytes

The requirements of components within the pool are

Integer constant	4 bytes
Real constant (SHORT)	4 bytes.
Real constant (LONG)	8 bytes
String (in bytes)	2 + number of symbols of open string between the outermost string quotes

The constant pool is divided into blocks of 4096 bytes each. The first block contains the integer constants 0 to 15 (64 bytes). All strings together are restricted to fill not more than the rest of this block (4096 - 64 - 2S bytes, where S = number of strings).

No constant occurring more than once in the source program is stored twice in the same block; however, it may possibly be stored more than once in different blocks. Up to seven bytes may be left unused.

Length of data storage area for each block or procedure declaration	≤4096 bytes
Number of blank spaces serving as delimiters on I/O data sets	≤255

Number of records per section	≤255
Number of entries in the Note Table	≤127

(The Note Table stores information to retrieve records which may be required again later. An entry for a record is made each time the ALGOL I/O procedures PUT and SYSACT13 are executed, and each time an input operation, with backward repositioning, follows an output operation on the same data set.)

Identification number (N) used by PUT or GET	0≤N≤65535

INVOKING A PROGRAM WITHIN A JOB STEP

Any one of the four macro-instructions, CALL, LINK, XCTL or ATTACH, may be used to dynamically invoke the compiler, linkage editor and load module within a job step. This is an alternative to the more usual method of invoking a program by starting a job step with an EXEC statement. Full details of the four macro-instructions are given in IBM System/360 Operating System: Control Program Services.

To invoke a program with the CALL macro-instruction, the program must first be loaded into main storage, using the LOAD macro-instruction. This returns, in general register 15, the entry address which is used by the CALL macro-instruction. The instructions used could be:

```
LOAD     EP=member-name

LR       15,0

CALL     (15), (option-address), VL
```

To invoke a program with one of the LINK, XCTL or ATTACH macro-instructions would need instructions such as:

```
LINK     EP=member-name,

         PARAM=(option-address), VL=1

XCTL     EP=member-name

ATTACH   EP=member-name,

         PARAM=(option-address), VL=1
```

"member-name" specifies the name of the member of a partitioned data set which contains the program required.

For the compiler, member-name=ALGOL

For the linkage editor, member-name=IEWL

For the load module, member-name is specified by the programmer in the SYSLMOD DD statement for the linkage editor.

"option-address" specifies the address of a list containing the options required by the user. An address must be given even if no options are specified. The list must begin on a half-word boundary. The first two bytes contain a number giving the number of bytes in the remainder of the list. (If no options are specified this number must be zero). The list itself contains any of the options available to the PARM parameter in an EXEC statement (see Appendix E).

When using CALL, LINK or ATTACH to invoke the compiler, other ddnames may be used in place of the standard ddnames given in Section 2 for the data sets (except for SYSABEND), and an alternative page number (instead of the normal 001) may be specified for the start of output listings.

If alternative ddnames are used, then in the statement invoking the compiler, "option-address" must be followed by "ddname-address" giving the address of a list containing the alternative ddnames. If alternative page numbers are used, then "page-address" giving the address of a location containing the alternative page number must be placed after "ddname-address"; though if alternative ddnames are not required "ddname-address" may be replaced by a comma.

The ddname list must begin on a half-word boundary. The first two bytes contain a number giving the number of bytes in the remainder of the list. The list itself contains up to ten 8-byte fields, separated by commas, for specifying alternative ddnames for the data sets. As only seven data sets are used by the compiler, three of the fields are left blank. The alternative ddnames must be listed in the following order:

Purpose of data set	Standard ddname
Output of object module for linkage editor	SYSLIN

-- Three blank fields --	
Source program input	SYSIN
Information listings	SYSPRINT
Output of object module for card deck	SYSPUNCH
Intermediate work	SYSUT1
Intermediate work	SYSUT2
Intermediate work	SYSUT3

The field for a data set which does not use an alternative ddname must be left blank if there is an alternative ddname following. Otherwise the field is omitted.

The location containing the page number must begin on a half-word boundary. The first two bytes contain a number giving the number of bytes in the remainder of the location (namely, four). These four bytes contain the number for the first page of the output listings, and on return to the invoking program they will contain the number of the last page.

An example of an invoking statement and the associated lists, for the compiler, is:

```
COMPILE   LINK   EP=ALGOL,PARAM=
                 (OPTIONS,DDNAMES,PAGE),
                 VL=1

OPTIONS   DC     H'25',C'PROCEDURE,DECK,
                 SIZE=90112'

DDNAMES   DC     H'35',C'OUTPUTbb,3CL8'b',
                 C'INPUTbbb',CL8'b',
                 C'CARDDECK'

PAGE      DC     H'04',F'62'

          b = blank
```

In this case, the PROCEDURE and DECK options are specified and 88K bytes of main storage are made available. Alternative ddnames are specified for SYSLIN, SYSIN and SYSPUNCH, and 62 is specified as the first page number for the output listings.

PRECOMPILED PROCEDURES

Subprograms, written in either ALGOL or Assembler language, may be compiled or assembled, linkage edited, and stored in load module form on a partitioned data set for subsequent call by a load module produced from an ALGOL source program when this latter module is being executed. The compiler will recognize a subprogram if the PROCEDURE option is specified in the invoking statement. A subprogram of this type when stored in load module form is known as a precompiled procedure and is specified in the calling ALGOL program by using the 'CODE' delimiter as the body of a procedure.

A precompiled procedure is loaded into main storage when control passes to the program block in which the precompiled procedure is declared, and is deleted when control leaves that block. Thus, precompiled procedures declared in disjoint program blocks will overlay each other.

The REUS option (see Appendix E) must not be specified for the precompiled procedure load module, in the statement invoking the linkage editor, if the installation allows multiprogramming.

The module name specified to the linkage editor for a precompiled procedure must be the procedure name used in the declaration of the procedure in the calling ALGOL program. The precision of real values must be the same, SHORT or LONG, in the calling ALGOL program and the precompiled procedure. If this rule is not observed then undefined results may occur.

ALGOL Procedures

The requirements for a precompiled procedure written in ALGOL are given in IBM System/360 Operating System: ALGOL Language.

Assembler Language Procedures

The following requirements must be observed when writing a precompiled procedure in Assembler language.

In the instructions given below, the programmer may specify any valid names in the name fields, provided that any cross-referencing of names is observed. To avoid erroneous results, other instructions should not be included in the following sequences.

Initialization Instructions

ASSTART	DC	XL2'	' Characteristic of first formal parameter (see Figure 17)
	DC	XL2'	' Characteristic of second formal parameter
	.		
	.		
	.		
	DC	XL2'	' Characteristic of last formal parameter
			First instruction executed

Termination Instructions

Reset CDSA, PBT and FSA registers for a type procedure, store value of type procedure at displacement 24 in the data storage area (DSA)

B EPILOGP (FSA)	Return to calling program

Definition Instructions

The following storage and constants must be defined:

PBTAB	DS	F	Space
	DC	CL4'	' Name of procedure (first 4 characters)
	DS	F	Address of DSA (set by FSA routine)
	DC	H'	' Length of DSA. At least 24 (+8 if type procedure) +8 x number of formal parameters
	DC {	X'08'	If type procedure
		X'04'	If procedure
	DC	X'0p'	where p is number of formal parameters in one hexadecimal digit.

Entry Point

At the entry point of the module there must be an address constant:

```
        DC      A(PBTAB,DUMMY,ASSTART)
```

Register Definitions

The following registers must be defined if used in the program:

ADR	EQU	8	Used in communications with calling ALGOL program.
GDSA	EQU	9	Used in communications with calling ALGOL program.
CDSA	EQU	10	Address of DSA. Must be reset before communication with calling ALGOL program.
PBT	EQU	11	Address of PBTAB. Must be reset before communication with calling ALGOL program.
FSA	EQU	13	Must be reset before communication with calling ALGOL program.
STH	EQU	14	Used in communications with calling ALGOL program.
BRR	EQU	15	Used in communications with calling ALGOL program.

Fixed Storage Area Displacements

The following displacements must be specified for routines in the fixed storage area which are used in the program.

CAP1	EQU	X'0D4'
CAP2	EQU	X'0D8'
PROLOGFP	EQU	X'0DC'
RETPROG	EQU	X'0E4'
EPILOGP	EQU	X'0E8'
CSWE1	EQU	X'0F4'
VALUCALL	EQU	X'118'

Parameter Handling

In the following instructions DISPL is the displacement of the formal parameter in the DSA. For example, the displacement of the nth formal parameter is:

$24 + 8(n-1)$, except in the case of type procedures where it is $32 + 8(n-1)$

Formal Parameters Called by Name:-

1. If the formal parameter is a string, an array, or of type real, integer or Boolean, the following method can be used to call the actual parameter.

   Save all registers.

   ```
   Reset CDSA, PBT and FSA registers
           BAL     BRR,CAP1(FSA)
           DC      H'8'
           DS      H
           L       ADR,DISPL(CDSA)
   ```

2. If the formal parameter is a procedure (containing j formal parameters), the following method can be used to call the actual procedure.

   Save all registers.

   ```
   Reset CDSA, PBT and FSA registers
           BAL     BRR,CAP1(FSA)
           DC      H'8'
           DS      H
           L       ADR,DISPL(CDSA)
           BAL     BRR,PROLOGFP(FSA)
           DC      A(THUNK1)
           DC      XL2'        '  Characteristic of
                                  first parameter
                                  (see Figure 17)
           DC      H'          '  Number of param-
                                  eters, j
           DC      A(THUNK2)
           DC      XL2'        '  Characteristic of
                                  second parameter
           DS      H
           .
           .
           .
           DC      A(THUNKj)
           DC      XL2'        '  Characteristic of
                                  last parameter
           DS      H
                                  Return after call
   ```

   A "thunk" is a sequence of instructions that loads register ADR with the address of the actual parameter. The following instructions must therefore be included in the precompiled procedure when the above sequence is used.

   ```
   THUNK1  LA  ADR,ACTPR1 Address of first
                          actual parameter *
           B   CAP2(FSA)
           .
           .
           .
   THUNKj  LA  ADR,ACTPRj Address of last
                          actual parameter
           B   CAP2(FSA)
   ```

   * In the case of a string the first 2 bytes should contain the length of the string.

3. If the formal parameter is a label, the following method can be used to call the actual parameter.

   Save all registers.

   ```
   Reset CDSA, PBT and FSA registers
           BAL     BRR,CAP1(FSA)
           DC      H'8'
           DS      H
           L       ADR,DISPL(CDSA)
           B       RETPROG(FSA)
   ```

4. If the formal parameter is a switch, the following method can be used to call the actual parameter.

   Save all registers.

   ```
   Reset CDSA, PBT and FSA registers
           BAL     BRR,CAP1(FSA)
           DC      H'8'
           DS      H
           L       ADR,DISPL(CDSA)
           LA      BRR,i          i=element number
           BAL     STH,CSWE1(FSA)
           B       RETPROG(FSA)
   ```

Formal Parameters Called by Value:-

If the formal parameter is an array, or of type real, integer or Boolean, the following method can be used to call the actual parameter.

Save all registers.

```
Reset CDSA, PBT and FSA registers
        BAL     BRR,CAP1(FSA)
        DC      H'8'
        DS      H
        L       ADR,DISPL(CDSA)
        BAL     BRR,VALUCALL(FSA)
        DC      H'DISPL'
        DC      XL2'        '  Characteristic of
                               formal parameter
                               (see Figure 17)
```

Type of Parameter	Characteristic Halfword (in hexadecimal form)		Result after call of actual parameter
	When called by name	When called by value	
STRING	CB10		ADR contains address of string
REAL	C212		ADR contains address of real value
REAL		C222	DISPL in CDSA contains real value
INTEGER	C211		ADR contains address of integer value
INTEGER		C221	DISPL in CDSA contains integer value
BOOLEAN	C213		ADR contains address of Boolean value
BOOLEAN		C223	DISPL in CDSA contains Boolean value
ARRAY or REAL ARRAY	CA16		ADR contains address of SMF (see below)
ARRAY or REAL ARRAY		CA26	DISPL in CDSA contains address of SMF
INTEGER ARRAY	CA15		ADR contains address of SMF
INTEGER ARRAY		CA25	DISPL in CDSA contains address of SMF
BOOLEAN ARRAY	CA17		ADR contains address of SMF
BOOLEAN ARRAY		CA27	DISPL in CDSA contains address of SMF
LABEL	CA18		ADR contains address of label
LABEL		CA28	ADR contains address of label
SWITCH	CA1C		ADR contains address of switch
PROCEDURE	CAD0		If the actual procedure is parameter-less then procedure is called, otherwise ADR contains address of procedure
REAL PROCEDURE	CAD2		If the actual procedure is parameter-less then procedure is called, and ADR contains address of real value, otherwise ADR contains address of procedure
REAL PROCEDURE		C2E2	DISPL in CDSA contains real value
INTEGER PRO-CEDURE	CAD1		If the actual procedure is parameter-less then procedure is called, and ADR contains address of integer value, otherwise ADR contains address of procedure
INTEGER PRO-CEDURE		C2E1	DISPL in CDSA contains integer value
BOOLEAN PRO-CEDURE	CAD3		If the actual procedure is parameter-less then procedure is called, and ADR contains address of Boolean value, otherwise ADR contains address of procedure
BOOLEAN PRO-CEDURE		C2E3	DISPL in CDSA contains Boolean value

Figure 17. Table of parameter characteristics for an Assembler language precompiled procedure. The SMF describes the storage layout of an array. Byte 0 contains a value denoting the number of subscripts in the array. Bytes 8 to 11 contain the address of the first element in the array. Bytes 16 to 19 contain a value denoting the size of the array.

When processing the source program, the compiler detects and specifies any routines that need to be combined with the generated object module before it can be executed. These routines are contained in the System/360 Operating System ALGOL library - a partitioned data set with the external name SYS1.ALGLIB. The routines are in load module form and the linkage editor combines them with the object module to produce an executable load module. There are three types of routines - fixed storage area routines, mathematical routines and input/output routines. Additionally, an error routine, stored on the operating system link library, SYS1. LINKLIB, is called at execution time if an error occurs.

Initialization and termination of the library routines is performed using the standard method (see "Supervisor" in Section 1).

FIXED STORAGE AREA

General routines required to some degree by all object modules are combined into a single load module known as the fixed storage area (IHIFSA). These routines are used to initialize and terminate execution of the ALGOL program, to handle the DSA when entering or leaving a program block or procedure, to produce the program trace, to load precompiled procedures, to get main storage for arrays, to convert values from real to integer and integer to real, to call actual parameters, to handle branches in the program, to handle program interrupts, etc....

MATHEMATICAL ROUTINES

Standard mathematical functions contained in ALGOL have corresponding mathematical routines in the library, except for ABS, SIGN and LENGTH which are handled by the compiler, and ENTIER which is contained in the fixed storage area. Routines exist in each case for both long and short precision of real numbers.

These mathematical routines are taken from the System/360 Operating System FORTRAN IV library and modified to conform to the ALGOL language requirements without affecting the mathematical methods used. Full details of these routines are contained in IBM System/360 Operating System: FORTRAN IV Library Sub-programs.

INPUT/OUTPUT ROUTINES

Data transfer between the load module and external data sets is performed by input/output routines. These routines correspond to the ALGOL I/O procedures and are mostly contained on separate load modules (see Figure 18). In addition there is a single load module, IHIIOR, which contains a number of commonly-used subroutines.

ERROR ROUTINE

If an error is detected during execution of the load module, an error routine (in SYS1. LINKLIB) is invoked. Its main purpose is to construct the error message and produce the data storage area listing before passing to the termination routine in the FSA. If a second error occurs while the first is being handled (due, for example, to an I/O error or because the object module has overwritten part of the ALGOL library or control program), then termination takes place immediately and incomplete information listings may be produced.

Module Name		When used	Storage estimate (bytes
ALGOL	FORTRAN IV		
IHIERR		When an error is detected at execution time	4270
IHIFDD	IHCFDXPD	For an exponentiation (** or 'POWER') using long precision base and long precision exponent	200
IHIFDI	IHCFDXPI	For an exponentiation (** or 'POWER') using long precision base and integer exponent	140
IHIFII	IHCFIXPI	For an exponentiation (** or 'POWER') using integer base and integer exponent	170
IHIFRI	IHCFRXPI	For an exponentiation (** or 'POWER') using short precision base and integer exponent	140
IHIFRR	IHCFRXPR	For an exponentiation (** or 'POWER') using short precision base and short precision exponent	200
IHIFSA		For every object module (except those for precompiled procedures)	5030
IHIGPR		For either GET or PUT	2420
IHIIAR		For INARRAY or INTARRAY	120
IHIIBA		For INBARRAY	70
IHIIBO		For INBOOLEAN	530
IHIIDE		For either INREAL or ININTEGER	1560
IHIIOR		For every object module	2910
IHIISY		For INSYMBOL	270
IHILAT	IHCLATAN	For a long precision arctangent operation (ARCTAN)	320
IHILEX	IHCLEXP	For a long precision exponential operation (EXP)	450
IHILLO	IHCLLOG	For a long precision logarithmic operation (LN)	310
IHILOR		For a long precision OUTREAL operation	730
IHILSC	IHCLSCN	For a long precision sine or cosine operation (SIN or COS)	370
IHILSQ	IHCLSQRT	For a long precision square root operation (SQRT)	140
IHIOAR		For OUTARRAY	120

IHIOBA		For OUTBARRAY	70
IHIOBO		For OUTBOOLEAN	400
IHIOIN		For OUTINTEGER	410
IHIOST		For OUTSTRING	300
IHIOSY		For OUTSYMBOL	290
IHIOTA		For OUTTARRAY	120
IHIPTT		For a long precision INREAL or OUTREAL operation	270
IHISAT	IHCSATAN	For a short precision arctangent operation (ARCTAN)	200
IHISEX	IHCSEXP	For a short precision exponential operation (EXP)	280
IHISLO	IHCSLOG	Fȯr a short precision logarithmic operation (LN)	210
IHISOR		For a short precision OUTREAL operation	810
IHISSC	IHCSSCN	For a short precision sine or cosine operation (SIN or COS)	260
IHISSQ	IHCSSQRT	For a short precision square root operation (SQRT)	170
IHISYS		For SYSACT	1520

Figure 18. Table of ALGOL library modules. All are contained in SYS1.ALGLIB except IHIERR which is in SYS1.LINKLIB. For mathematical routines, the corresponding name in the FORTRAN IV library is also given.

APPENDIX B: IBM-SUPPLIED CATALOGED PROCEDURES

The three cataloged procedures for ALGOL that were introduced in Section 2 are contained in the procedure library, SYS1.PROCLIB, of the operating system. They consist of the job control statements listed below.

These procedures have been designed for an optimum job, and can be over-ridden by the user if he requires different or additional system support to that provided (see Section 2). In particular it should be noted that in these procedures the object or load module produced is stored on a temporary data set and will therefore be deleted at the end of the job.

Compilation, ALGOFC

```
//ALGOL EXEC PGM=ALGOL
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD UNIT=SYSCP
//SYSLIN DD DSNAME=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,DISP=(MOD,PASS),     X
//             SPACE=(4ØØ,(4Ø,1Ø)
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT2 DD UNIT=SYSSQ,SEP=(SYSUT1,SYSLIN,SYSPUNCH),                     X
//             SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT3 DD UNIT=SYSDA,SPACE=(2ØØØ,(2Ø,5)),                              X
//             SEP=(SYSUT1,SYSUT2,SYSLIN,SYSPUNCH)
//SYSABEND DD SYSOUT=A
```

Compilation and Linkage Editing, ALGOFCL

```
//ALGOL EXEC PGM=ALGOL
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD UNIT=SYSCP
//SYSLIN DD DSNAME=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,DISP=(NEW,PASS),     X
//             SPACE=(4ØØ,(4Ø,1Ø))
//SYSLMOD DD DSNAME=&GOSET,UNIT=SYSDA,DISP=(MOD,PASS),                   X
//             SPACE=(1Ø24,(5Ø,2Ø,1))
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT2 DD UNIT=SYSSQ,SEP=(SYSUT1,SYSLIN,SYSPUNCH),                     X
//             SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT3 DD UNIT=SYSDA,SPACE=(2ØØØ,(2Ø,5))                               X
//             SEP=(SYSUT1,SYSUT2,SYSLIN,SYSPUNCH)
//SYSABEND DD SYSOUT=A
//LKED EXEC PGM=IEWL,PARM=(XREF,LIST,LET),COND=(5,LT,ALGOL)
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSNAME=*.ALGOL.SYSLIN,DISP=(OLD,DELETE)
//SYSLIB DD DSNAME=SYS1.ALGLIB,DISP=OLD
//SYSLMOD DD DSNAME=&GOSET(GO),UNIT=SYSDA,DISP=(MOD,PASS),               X
//             SPACE=(1Ø24,(5Ø,2Ø,1))
//SYSUT1 DD UNIT=SYSDA,SEP=(SYSLIN,SYSLIB,SYSLMOD),                      X
//             SPACE=(1Ø24,(5Ø,2Ø))
//SYSABEND DD SYSOUT=A
```

```
//ALGOL EXEC PGM=ALGOL
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD UNIT=SYSCP
//SYSLIN DD DSNAME=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,DISP=(NEW,PASS),    X
//             SPACE=(4ØØ,(4Ø,1Ø))
//SYSLMOD DD DSNAME=&GOSET,UNIT=SYSDA,DISP=(MOD,PASS),                 X
//             SPACE=(1Ø24,(5Ø,2Ø,1))
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT2 DD UNIT=SYSSQ,SEP=(SYSUT1,SYSLIN,SYSPUNCH),                    X
//             SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT3 DD UNIT=SYSDA,SPACE=(2ØØØ,(2Ø,5)),                            X
//             SEP=(SYSUT1,SYSUT2,SYSLIN,SYSPUNCH)
//SYSABEND DD SYSOUT=A
//LKED EXEC PGM=IEWL,PARM=(XREF,LIST,LET),COND=(5,LT,ALGOL)
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSNAME=*.ALGOL.SYSLIN,DISP=(OLD,DELETE)
//SYSLIB DD DSNAME=SYS1.ALGLIB,DISP=OLD
//SYSLMOD DD DSNAME=&GOSET(GO),UNIT=SYSDA,DISP=(OLD,PASS),             X
//             SPACE=(1Ø24,(5Ø,2Ø,1))
//SYSUT1 DD UNIT=SYSDA,SEP=(SYSLIN,SYSLIB,SYSLMOD),                    X
//             SPACE=(1Ø24,(5Ø,2Ø))
//SYSABEND DD SYSOUT=A
//GO EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,ALGOL),(5,LT,LKED))
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(TRK,(5,2))
//SYSABEND DD SYSOUT=A
```

APPENDIX C: CARD CODES

The card deck of the source program is punched line for line from the text written on the coding sheets. The card code used can be either a 53 character set in Extended Binary Coded Decimal Interchange Code (EBCDIC), or a 46 character set in Binary Coded Decimal (BCD). This latter character set has been established as standard for ALGOL by the International Standards Organization (ISO) and Deutsche Industrie Normen (DIN). Figure 19 shows these two codes.

Characters	Card Codes	
	EBCDIC	ISO/DIN
A to Z	12-1 to 0-9	12-1 to 0-9
0 to 9	0 to 9	0 to 9
+	12-8-6	12
-	11	11
*	11-8-4	11-8-4
/	0-1	0-1
=	8-6	8-3
,	0-8-3	0-8-3
.	12-8-3	12-8-3
'	8-5	8-4
(	12-8-5	0-8-4
)	11-8-5	12-8-4
blank	no punch	no punch
<	12-8-4	
>	0-8-6	
|	12-8-7	
&	12	
¬	11-8-7	
:	8-2	
;	11-8-6	

Figure 19. Source program card codes.

The object module is in a form acceptable as input to the linkage editor, that is, its records are card images having the format of ESD, RLD, TXT and END cards (see Figure 20). It is stored either on a data set (ddname SYSLIN) in the linkage editor library, or on an output data set (ddname SYSPUNCH), or on both. The parameters LOAD and DECK, used to specify these storage options are described in Appendix E.

The object module consists of:

1. An initial ESD card defining the control section. For a precompiled procedure, the procedure name (up to 6 characters) is assigned to the control section and entered into this record.
2. The Constant Pool containing all constants and strings in the module.
3. The generated instructions.
4. The Label Address Table (see Section 3) for addressing branch instructions in the module.
5. The Program Block Table containing an entry for every program block. This table indicates the active generation of data storage areas (see Section 3) and length of each data storage area.
6. The Data Set Table containing information on the current status of all data sets used. This table is not produced for precompiled procedures.
7. Program start information.
8. An END card.

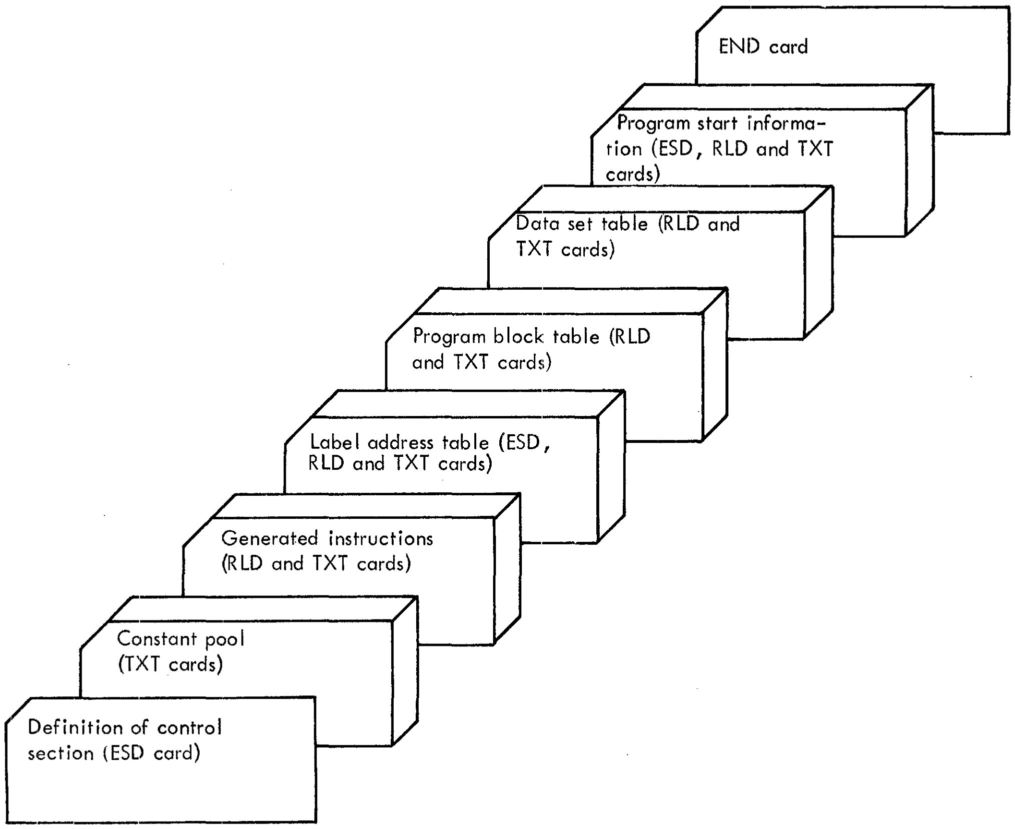

END card
Program start information (ESD, RLD and TXT cards)
Data set table (RLD and TXT cards)
Program block table (RLD and TXT cards)
Label address table (ESD, RLD and TXT cards)
Generated instructions (RLD and TXT cards)
Constant pool (TXT cards)
Definition of control section (ESD card)

Figure 20. The object module card deck. The ESD (External Symbol Dictionary) cards contain the external symbols that are defined or referred to in the module. The RLD (Relocation Dictionary) cards contain addresses used in the module. The TXT (Text) cards contain the constants and instructions used in the module. The END card indicates the end of the module.

APPENDIX E: USING JOB CONTROL LANGUAGE

This appendix describes the method of writing job control statements, and explains the options most frequently used by the ALGOL programmer. A full description of Job Control Language is given in IBM System/360 Operating System: Job Control Language.

CODING FORMAT

Control statements are identified by the initial characters // or /* and are written in columns 1 to 72 of standard 80 column punched cards. Each field is separated by one or more blanks. Column 72 must be left blank unless the statement is to be continued on another card.

If the length of a statement exceeds 71 characters, it must be continued on another card. This is done by interrupting the statement at the end of a positional or keyword parameter, following this parameter with a comma, and placing any non-blank character in column 72. The continuation card commences with the initial characters // and the statement restarts on column 16. Command statements may not be continued on another card.

Comments must be separated from the last parameter by one or more blanks. If the comment is to be continued on another card it may be interrupted at any convenient point and a non-blank character is put in column 72. The continuation card commences with the initial characters // and the comment restarts on any column from 16 to 71 inclusive.

The four possible formats for control statements are shown in Figure 21. The null and delimiter statements are blank except for the first two columns.

NAME contains the symbolic identification of the control statements. It is always placed immediately after the initial characters //. A name must contain between one and eight alphameric characters, the first of which must be alphabetic. If name is omitted, then at least one blank must separate the initial characters // and the operation field.

OPERATION identifies the type of control statement being specified.

OPERAND contains the statement parameters, separated by commas.

Format	Applicable Control Statements
//NAME OPERATION OPERAND	JOB,EXEC, DD
//OPERATION OPERAND	EXEC,DD, Command
//	Null
/*	Delimiter

Figure 21. Control statements formats.

CONVENTIONS

The conventions used in this manual for describing control statements are as follows:

Upper case letters and punctuation marks (except those listed below) represent information to be coded exactly as shown.

Lower case letters are general terms requiring substitution of specific information by the programmer.

These punctuation marks have a special meaning:

- - (hyphen) links lower case words to form a single term for substitution
- _ (underscore) indicates the option that will be assumed if none is specified
- { } (braces) mean only one of the options contained must be selected
- [ ] (brackets) mean information contained may be omitted
- ... (ellipsis) means that preceding item can be repeated successively a number of times.

CONTROL STATEMENT CODING

In the following description, certain terms are used to indicate external names which are to be specified by the programmer. These terms and their meanings are:

Term	Meaning
jobname	name of job
progname	name of program

stepname	name of job step
ddname	name of DD statement (the standard ddnames which may be specified are described in Section 2)
procname	name of cataloged procedure
procstep	name of job step within a cataloged procedure
dsname	name of data set

It is often convenient to use two or more qualification levels to specify a data set name. The highest level reference is stated first. Thus in Figure 22, data set D.M.H is found by searching the index of each volume in turn, starting with the system residence volume (the primary volume in the operating system), to find the location of data set D. This, when searched, will contain the location of data set D.M, which in turn will contain the location of data set D.M.H.

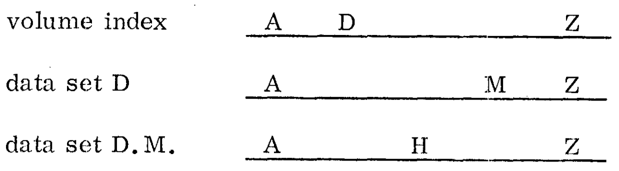


Figure 22. Data set cataloging using qualified names.

A maximum of 44 characters can be used for a qualified name. Thus, as a simple name can consist of between one and eight characters, and each name must be separated by the character period (.), a maximum of 22 qualification levels is possible.

Data set names can also be qualified by a suffix, that is, "dsname (element)", to indicate the relative generation number. For example, WEATHER (0) is the current generation of the data set named WEATHER. The preceding generation would be WEATHER (-1). A new generation during creation is known as WEATHER (+1), at the end of the job it becomes WEATHER (0). A suffix is also used to indicate the name of a member of a partitioned data set, or the area of an indexed sequential data set.

There are four types of job control parameters for inclusion in the operand fields: positional parameters, keyword parameters, positional subparameters and keyword subparameters.

Positional parameters must be stated first, and where more than one can be included they must be listed in the order given in the following descriptions. A comma must be substituted in place of any positional parameter omitted, if it is to be followed by another positional parameter, for example,

```
//name   operation   pos1,,pos3......
```

Keyword parameters can be listed in any order. They contain a keyword followed by an equal sign (=) and some specific information. All keyword parameters are optional since a default option will exist for any which must be specified.

One or more subparameters can be substituted for a positional parameter and also for the information to the right of the equal sign in the keyword parameter.

Positional subparameters have the same configuration and restrictions as positional parameters.

Keyword subparameters have the same configuration and restrictions as keyword parameters.

When more than one subparameters are used, they must be separated by commas and the list enclosed in parentheses, for example,

```
// name   operation   pos1,pos2,key1=value,
//                              key2=(sub1,sub2)
```

Since some special characters, such as the comma, parenthesis, blank and equal sign, have a special significance when used in control statements, no special characters can usually be used in job control information provided by the user. There are, however, some exceptions to this rule. The special characters @, $ and # can be represented normally. All other special characters, except the apostrophe, can be represented normally in the programmer's-name in the JOB statement, the accounting-information in the JOB and EXEC statements, and the PARM parameter options in the EXEC statement, provided that the information is enclosed in apostrophes (replacing the parentheses for a list of more than one subparameter). An apostrophe within this information is represented by two consecutive apostrophes.

JOB Statement

The name field of the JOB statement must contain the external name for the job (jobname).

The operation field must contain the characters JOB

The parameters available for the operand field are listed in Figure 23, where:

accounting-information
: identifies the installation account number to which the computer time for this job is to be charged. If the installation has an appropriate accounting routine, the account number can be followed by other subparameters, which are fixed by the user for his own installation. If the account number is omitted then its absence must be indicated with a comma.

programmer's-name
: identifies the person responsible for the job. It must not exceed 20 characters.

TYPRUN=SETUP
: indicates that the operator must mount a volume before the job can be done.

TYPRUN=NONSETUP
: indicates that a volume does not have to be mounted before the job can be done.

PRTY=job-priority
: indicates the relative priority of the job. A number from 0 to 14 is specified, with 14 being the highest priority. This parameter can be used only with priority scheduling.

Positional parameters	[accounting-information] [programmer's-name]
Keyword parameters (all optional)	TYPRUN= $\left\{\begin{array}{l}\underline{\text{SETUP}}\\ \text{NONSETUP}\end{array}\right\}$ PRTY=job-priority COND=((code,operator),...) MSGLEVEL= $\left\{\begin{array}{l}\underline{0}\\ 1\end{array}\right\}$ MSGCLASS=classname

Figure 23. JOB statement parameters

COND=((code,operator),...)
: allows conditions for the termination of the job to be specified. Up to eight (code,operator) specifications may be included in a COND parameter. Any number between 0 and 4095 is substituted for "code" and one of the following six relationships is substituted for "operator".

Operator	Meaning
GT	greater than
GE	greater than or equal to
EQ	equal to
NE	not equal to
LE	less than or equal to
LT	less than

At the completion of each job step, unless a system error occurs, the operating system will generate a return code between 0 and 4095 (see Section 1) to indicate if the program was executed successfully or not. If any of the code numbers stated in the COND parameter is related to the return code in the way specified by the associated operator then the job is terminated. For example, if

COND=((50,LT), (40,GT))

then, the job will be terminated if either 50 is less than the return code, or 40 greater than the return code.

MSGLEVEL=0
: indicates that the job scheduler is to write out control statement information only when an error occurs. The information required is a diagnostic message and the control statement in which the error occurred.

MSGLEVEL=1
: indicates that, whether an error occurs or not, the job scheduler is to write out all control statements, plus a diagnostic message if an error does occur.

MSGCLASS=classname
: allows job scheduler messages to be written in a system output class other than the one normally used by the installation. The user can fix up to 36 different classes (A to Z and 0 to 9), depending on device type, priority, destination, etc., for these messages. This parameter is not necessary if the normal class (A)

is required. For sequential scheduling only class A may be used.

EXEC Statement

The name field contains the external name of the job step (stepname). It may be omitted if no reference is to be made to the EXEC statement in another statement.

The operation field must contain the characters EXEC

The parameters available for the operand field are listed in Figure 24, where:

PGM=progname
: indicates that the job step executes the program named "progname". The program must reside on a partitioned data set.

PGM=*. stepname. ddname
: indicates that the job step executes the program named by the DSNAME parameter of a DD statement named "ddname" that was included in a previous job step named "stepname" in the same job. If "stepname" refers to a job step invoking a cataloged procedure then a job step within the procedure can be specified by putting its name after "stepname"; that is, "stepname. procstep". The program must reside on a partitioned data set.

PROC=procname
: indicates that the job step executes the cataloged procedure named "procname".

procname
: has the same effect as PROC=procname

TIME=(minutes, seconds)
: limits the computing time for the job step. If "seconds" only is specified then a comma must be substituted for "minutes". If "minutes" only is specified then the parentheses can be deleted. This parameter can be only used with supervisor configurations incorporating the timing facility.

COND=((code, operator, stepname)....)
: allows conditions to be specified for bypassing a job step whose execution depends on the return code issued by a preceding job step. "Code" and "operator" are governed by the same stipulations that applied for the JOB statement. "Stepname" indicates the previous job step which issued the return code to be used for comparison. If "stepname" is not specified then the return code issued by all previous job steps are compared. If "stepname" refers to a job step invoking a cataloged procedure then a job step within the procedure can be specified by putting its name after "stepname"; that is "stepname. procstep".

Positional parameters	PGM=progname
	PGM=*. stepname. ddname
	PROC=procname
	procname

Keyword parameters (all optional)	TIME / TIME. procstep = (minutes, seconds) COND / COND. procstep = ((code, operator, stepname), . . .) PARM / PARM. procstep = subparameter-list ACCT / ACCT. procstep = accounting-information

Figure 24. EXEC statement parameters.

PARM=subparameter list
: indicates any special conditions which apply to the job step. All the subparameters in the "subparameter-list" are optional. They can be specified in any order, and a comma does not have to be substituted for any omitted. A maximum of 40 characters may be used. For the rule to be observed when an equal sign is included in the subparameter-list (that is, with SIZE, TRBEG and TREND), see "Control Statement Coding".

For the ALGOL compiler job step, the "subparameter-list" is given below. For each of the alternatives, the compiler assumes that the option underscored applies, unless the other is specified either at this stage or during system generation. The default options PROGRAM and TEST cannot be changed at system generation. If a large number of options need to be specified for a particular job then the 40 character limitation may be exceeded. To avoid this, abbreviated forms, given at the end of the description of each option, may be used.

PROGRAM or PROCEDURE: which specifies that the source program is either an ALGOL program in the sense of the ALGOL syntax (PROGRAM), or is an ALGOL procedure to be compiled separately and used with other programs or procedures (PROCEDURE). Abbreviated forms PG or PC.

SHORT or LONG: which specifies that the internal representation of real values is in full words (SHORT); or double words (LONG). Abbreviated forms SP or LP.

NODECK or DECK: which specifies that an object module, stored on the data set specified in the SYSPUNCH DD statement, either is not to be generated (NODECK); or is to be generated (DECK). Abbreviated forms ND or D.

LOAD or NOLOAD: which specifies that the compiler is to either generate an object module for use as input to the linkage editor, using the data set specified in the SYSLIN DD statement (LOAD); or not generate this object module (NOLOAD). Abbreviated forms L or NL.

SOURCE or NOSOURCE: which specifies that the source program and identifier table listings are either to be printed (SOURCE); or not to be printed (NOSOURCE). Abbreviated forms S or NS.

EBCDIC or ISO: which specifies that the card code used to write and keypunch the source program is either a 53 character set in EBCDIC (EBCDIC); or the 46 character set in BCD which has been established as standard for ALGOL by ISO and DIN (ISO). Abbreviated forms EB or I.

TEST or NOTEST: which specifies that the generated object module is to include information which is normally used only for testing (TEST); or is not to include this information (NOTEST). The information consists of instructions to produce the semicolon count, and instructions checking the values of subscript expressions against array bounds. Abbreviated forms T or NT.

SIZE=45056 or SIZE=number: which specifies the main storage size, in bytes, that is available to the compiler. "Number" must not be less than 45056 and must not exceed 999999.

For the linkage editing job step the "subparameter-list" consists of two types of options, those which specify the output listings required, and those specifying attributes for the load module.

The options to control output listings are:

LIST which specifies that all job control statements processed by the linkage editor are to be listed on the diagnostic output data set.

MAP or XREF which specifies that either a map of the load module is to be produced (MAP); or a cross-reference table of the load module is to be produced (XREF) comprising a load module map and a list of all address constants that refer to other control sections.

The options specifying load module attributes which can be used with ALGOL programs are:

REUS which produces a load module that is serially reusable, that is, it can be used by more than one task, but only one task at a time.

DC which produces a load module that is downward compatible, that is, if the load module is produced by an F level linkage editor then it can be reprocessed by an E level linkage editor.

LET or XCAL which specifies that either the load module is to be marked as executable even when a severity 2 error is detected (LET); or the load module is to be marked as executable even though invalid external references between the segments have been made (XCAL). A severity 2 error could make execution impossible and would normally lead to the load module being marked as not executable. It includes the situation over-ridden by XCAL.

NCAL which specifies that the linkage editor automatic library call mechanism is not to call library members to resolve external references within the object module. The load module is marked as executable even though unresolved external references have been recognized.

All the linkage editor subparameters are optional.

For the execution job step of an ALGOL program the "subparameter-list" is:

TRACE which specifies that the semicolon count produced during the compilation process is to be printed as a list. This gives information on the dynamic flow of the program and is known as a program trace.

TRBEG=number which specifies that a limited program trace is to be produced beginning at the semicolon specified by "number" and ending at the physical end of the program.

TREND=number which specifies that a limited program trace is to be produced beginning at the physical beginning of the program and ending at the semicolon specified by "number".

The last two options may be specified together to define the beginning and end of the trace. When either is specified, TRACE may be omitted, but in that case precompiled procedures would not be included. If TRACE is specified with TRBEG or TREND, then only a limited program trace is produced, but it will include precompiled procedures executed in that part of the program.

DUMP which specifies that a partial main storage dump is to be produced if an error occurs. The dump contains the contents of the data storage areas and arrays.

All of the execution time subparameters are optional.

ACCT=accounting-information
: allows accounting information associated with the job step to be passed to the installation's accounting routines, using subparameters which are fixed by the user for his own installation.

keyword. procstep
: is used with the last four parameters when a cataloged procedure is being executed. It indicates that the parameter applies to the job step named "procstep" within the procedure, and may be repeated for each keyword and with different, or the same, information to the right of the equal sign, for each job step in the procedure.

DD Statement

The name field contains an identifying name (ddname) for the DD statement.

The operation field must contain the characters DD

The parameters available for the operand field are listed in Figure 25, where:

* indicates, when used as a positional parameter, that the required data follows immediately after this DD statement. The asterisk must be the only non-blank character in the operand field. For sequential scheduling it can be used only once in each job step, and the data must be followed by a delimiter statement.

DUMMY
: indicates that the user's problem program is to be executed without any I/O operations on the data set. This can be used for debugging, and also for bypassing data set references in a regularly-used program, for example, the first run of an updating program when there is no old master to be processed.

DSNAME=dsname (element)
: specifies the name of a newly defined data set, or refers to one that has been defined previously. "Element" is used only if it is necessary to specify the generation number of the data set, the name of a member of a partitioned data set, or the area of an indexed sequential data set (using the options PRIME, OVFLOW or INDEX).

DSNAME=&name (element)
: specifies that the data set is temporary and will be deleted before the end of the job. The name allocated by the operating system is "name. jobname". "Element" has the same meaning as when used with DSNAME=dsname.

DSNAME=*. stepname. ddname
: indicates that the data set is the one specified in a preceding DD statement named "ddname" occurring in the job step named "stepname". If the data set was specified in the current job step then "stepname" must be omitted. If "stepname" refers to a job step invoking a cataloged procedure then a job step within the procedure can be specified by putting its name after "stepname"; that is "*. stepname. procstep. ddname".

Note. If the DSNAME parameter is omitted then the operating system will assign a unique name to any data set created by the job step.

Positional parameters (all optional)	$\left\{\begin{array}{l}\text{*}\\ \text{DUMMY}\end{array}\right\}$
Keyword parameters (all optional, though DSNAME can be omitted only when the asterisk positional parameter is used).	DSNAME= $\left\{\begin{array}{l}\text{dsname(element)}\\ \text{\&name(element)}\\ \text{*. stepname. ddname}\end{array}\right\}$ DCB= $\left[\left\{\begin{array}{l}\text{*. stepname. ddname}\\ \text{dsname}\end{array}\right\}\right]$ [subparameter-list] $\left\{\begin{array}{l}\text{AFF=ddname}\\ \text{SEP=subparameter-list}\end{array}\right\}$ UNIT=subparameter-list $\left\{\begin{array}{l}\text{SPACE=subparameter-list}\\ \text{SPLIT=subparameter-list}\\ \text{SUBALLOC=subparameter-list}\end{array}\right\}$ VOLUME=subparameter-list LABEL=subparameter-list $\left\{\begin{array}{l}\text{DISP=subparameter-list}\\ \text{SYSOUT=subparameter-list}\end{array}\right\}$

Figure 25. DD statement parameters.

DCB= $\left\{\begin{array}{l}\text{*. stepname. ddname}\\ \text{dsname}\end{array}\right\}$ [subparameter-list]

indicates that the data control block for the data set specified in the DD statement named "ddname" in the job step named "stepname", or alternatively the cataloged data set named "dsname", is to be repeated for the current DD statement. "Stepname" must be omitted if it refers to the current job step, or may be qualified in the same way as the DSNAME parameter if it refers to a job step in a cataloged procedure. If additional information is substituted for "subparameter-list" then this over-rides the corresponding subparameters in the repeated information. Alternatively "subparameter-list" can be used alone to specify data control block information.

The "subparameter-list" for the data sets used when processing and executing an ALGOL program contains the following keyword subparameters:

BLKSIZE=number, is used to specify blocksize. "Number" is blocksize in bytes, and for fixed length records must be a multiple of record length.

RECFM=F [B] [A], is used to specify record format. F = fixed length, B = blocked, A = control character incorporated to control printed output format.

LRECL=value, is used to specify record length. "Value" is actual or maximum length in bytes.

All other valid DCB options are fixed.

AFF=ddname
indicates that the data set has affinity with the data set specified by the DD statement named "ddname" and is to use the same channel.

SEP=list-of-ddnames
indicates that the data set is to use a separate channel to the ones used by the data sets specified by the DD statements named in the "list-of-ddnames".

UNIT=subparameter-list
specifies the class and quantity of I/O devices to be allocated for use by a data set. The "subparameter-list" has three forms, any one of which may be used in an individual statement. The three forms are:

1	Positional subparameters	classname $\left\{\begin{array}{c}\underline{1}\\ \text{number}\\ \text{P}\end{array}\right\}$ [DEFER]
	Keyword subparameter	[SEP=list-of-ddnames]
2	Positional subparameters	POOL, ddname $\left\{\begin{array}{c}\underline{1}\\ 0\end{array}\right\}$
3	Keyword subparameter	AFF=ddname

"classname" indicates the device class. These names are divided into two categories.

- Those automatically incorporated in the operating system when it is generated. These are of two types - specific unit names, such as 2400 (for a magnetic tape drive) and 1403 (for a printer); and general classnames, that is,

  SYSCP for any card punch
  SYSSQ for any magnetic tape or direct access device
  SYSDA for any direct access device.

- Additional names fixed by the user for his installation when the operating system is generated.

"number" indicates the number of devices to be allocated. If the data set is cataloged but the number of devices used is unknown, then "P" substituted for "number" will ensure that the correct number is assigned.

DEFER indicates that the volume need not be mounted on the I/O device until the data set is called in the program. This subparameter must not be used with an indexed sequential data set or a new output data set on a direct access device.

SEP=list-of-ddnames indicates for direct access devices that, if possible, the data set is not to use the same access arm as the data sets specified by the DD statements, given in the "list-of-ddnames".

POOL, ddname, indicates that the data set is to use the pool of tape units previously established by the DD statement named "ddname" in the same job step. A pool could be established to conserve I/O devices if a number of output data sets, that might exceed one tape reel each, are being produced by the job step. The pool would consist of one tape unit for each data set, plus one or more additional units. When a data set reached the end of its tape reel, output would be automatically continued on one of the additional units, and the first tape reel would be rewound and then replaced by the operator with a new reel so that the unit would be available for other data sets. The pool would be established by using the first form of the UNIT "subparameter-list" in a DD statement. Only the AFF or SEP parameters may be used with the UNIT parameter in this statement.

1 or 0 indicates that an extra tape unit is either to be added to the pool, or not to be added to the pool.

AFF=ddname indicates that the data set is to use the same I/O devices as the data set specified in the DD statement named "ddname" in the same job step.

PACE=subparameter-list
indicates the space required when a direct access device is specified in the UNIT parameter. The "subparameter-list" contains only positional subparameters. The list is:

```
{TRK                   }
{CYL                   } primary-quantity
{average-record-length }

[secondary-quantity][directory-or-index-quantity]

        {MXIG  }
[RLSE][ {ALX   } ][ROUND]
        {CONTIG}
```

The first subparameter specifies the units in which the space requirements are expressed, that is, tracks, cylinders or records (with length given in bytes).

The next subparameter specifies the space required. It has three parts (of which the second and third are optional) and is enclosed in parentheses if more than one part is specified. If the second part is omitted, then it must be substituted by a comma if the third part is included. The initial space to be allocated is given by "primary-quantity". Each time this initial space is filled, additional space is to be provided as specified by "secondary-quantity". The number of 256 byte records to be allocated for the directory of a new partitioned data set, or the number of cylinders, taken from the initial space reserved, to be allocated for the index of an indexed sequential data set, is given by "directory-or-index-quantity".

RLSE indicates that any unused space assigned to the data set is to be released.

MXIG requests that the largest single block of storage available is to be allocated to the data set.

ALX requests that extra blocks of storage (in track units) are to be allocated to the data set. As many available blocks that are equal to or greater than "primary-quantity", up to a maximum of five, will be allocated.

CONTIG specifies that the space specified by "primary-quantity" is to be in a single block.

ROUND requests that when records are used to express the space required on the direct access device, the space is to begin and end on cylinder boundaries.

DISP=subparameter-list
indicates the status of the data set and specifies its disposition at the end of the job step. The "subparameter-list" consists of the following positional subparameters:

```
           {DELETE  }
{NEW}      {KEEP    }
{OLD}  [   {PASS    }  ]
{MOD}      {CATLG   }
           {UNCATLG }
```

NEW specifies that the data set is to be generated in this job step, and would be deleted at the end of the job step unless KEEP, PASS or CATLG is specified.

OLD specifies that the data set already exists, and would be kept at the end of the job step unless PASS or DELETE is specified.

MOD specifies that the data set already exists and is to be modified in this job step. If the data set cannot be found by the operating system then this parameter is equivalent to NEW.

DELETE specifies that the space used by the data set (including that in the data set catalog, etc.) is to be released at the end of the job step.

KEEP specifies that the data set is to be kept at the end of the job step.

PASS specifies that the data set is to be referred to in a later step of this job, at which

time its final disposition, or a further pass, will be specified.

CATLG specifies that the data set is to be cataloged at the end of the job step. Thus KEEP is implied. The catalog structure must already exist.

UNCATLG specifies that the data set is to be deleted from the catalog at the end of the job step. KEEP is implied.

SYSOUT=subparameter-list
specifies the printing or punching operation to be used for the data set. The "subparameter-list" is:

classname [progname][number]

"classname specifies the system output class to be used. Up to 36 different classes (A to Z, 0 to 9) may be fixed by the user for his installation, according to device type, priority, destination, etc. The standard classname is A.

"progname" can be used to specify the name of a user-written output routine.

"number" can be used to specify an installation form number to be assigned to the output.

For sequential scheduling, the "subparameter-list" consists of only the standard classname A.

VOLUME=subparameter-list
indicates the volume or volumes assigned to the data set. If the data set is cataloged this parameter is not necessary. The "subparameter-list" is:

Positional subparameters	[RETAIN][number][value]
Keyword subparameters	$\left\{\begin{array}{l}\text{SER=list-of-serial-numbers}\\ \text{REF=}\left\{\begin{array}{l}\text{dsname}\\ \text{*. ddname}\\ \text{*. stepname. ddname}\\ \text{*. stepname. procstep. ddname}\end{array}\right\}\end{array}\right\}$

RETAIN specifies that, if possible, the volume is to remain mounted until referred to in a later DD statement, or until the end of the job, whichever is first. "number" is any number between 2 and 9999, and is used if an input or output operation, on a cataloged data set residing on more than one volume, does not start on the first volume of the data set. The number specifies the position of the volume on which input or output does start (for example, 3 indicates the third volume of the data set).

"value" specifies the number of volumes required by an output data set. It is not required if SER or REF is used.

SER=list-of-serial-numbers, specifies the serial numbers allocated by the user to the volumes required by the data set. These serial numbers can consist of between one and six alphameric characters.

$$\text{REF=}\left\{\begin{array}{l}\text{dsname}\\ \text{*. ddname}\\ \text{*. stepname. ddname}\\ \text{*. stepname. procstep. ddname}\end{array}\right\}$$

specifies that this data set is to use the same volume or volumes as the data set specified by one of the alternative sub-subparameter forms. If the latter data set resides on more than one tape volume, then only the last volume (as specified in the SER subparameter) can be used.

LABEL=subparameter-list
indicates the type of label or labels associated with the data set. If the data set is cataloged this parameter is not necessary. The "subparameter-list" is:

Positional subparameters	[number] $\left\{\begin{array}{l}\text{NL}\\ \underline{\text{SL}}\\ \text{NSL}\\ \text{SUL}\end{array}\right\}$
Keyword subparameters	$\left\{\begin{array}{l}\text{EXPDT=yyddd}\\ \text{RETPD=dddd}\end{array}\right\}$

"number" is any number between 2 and 9999, and specifies the position of the data set on the volume (for example, 3 would indicate the third data set on the volume).

NL, SL, NSL, and SUL specify the type of label or labels to be used, that is, no labels, standard labels, non-standard labels, and standard

and user labels, respectively. The routines to produce non-standard labels must be written and incorporated into the operating system by the user.

EXPDT=yyddd specifies that the data set cannot be deleted or opened, without operator intervention, until the date given by yy (year) and ddd (day).

RETPD=dddd specifies that the data set is to be retained for the number of days given by dddd.

Command Statement

The options available for the operation and operand fields of the command statement are described in IBM System/360 Operating System: Operator's Guide.

DATA SET CONTATENATION

Unless it has been created in the same job, a load module specified for execution in an EXEC control statement must be contained in the SYS1. LINKLIB library of the operating system. If the load module is not a permanent member of this library then it is temporarily combined by using a DD statement with the name JOBLIB.

If the load module is a member of another library then this whole library is combined with the SYS1. LINKLIB library. This temporary combining is termed concatenation and lasts only for the duration of the job. A statement of this kind would have the form:

```
//JOBLIB DD DSNAME=dsname,DISP=OLD
```

where "dsname" is the name of the data set or library containing the load module to be executed.

Only one JOBLIB DD statement can be used in each job and it must immediately follow the JOB statement. If more than one load module contained in a library being concatenated is required in the same job then the parameter DISP=(OLD,PASS) placed immediately after the DSNAME parameter, will extend the effect of the concatenation through each step of the job.

If the job requires load modules from a number of data sets which are not created in the job or not permanent members of the SYS1. LINKLIB library then one data set is concatenated to this library, as described above, and the others are concatenated to this first data set by listing their DD statements immediately after the JOBLIB DD statement and leaving the name fields blank. This has the effect of concatenating all the data sets to the SYS1. LINKLIB library.

JOB CONTROL LANGUAGE EXAMPLES

Three different types of jobs are described here to illustrate the use of job control language. Some of the subparameters used, such as I/O device classnames and volume serial numbers, may change for different installations.

Example 1: Executing a Single Load Module

Statement of problem: A set of 80 matrices are contained in data set SCIENCE.MATH.MATRICES. Each matrix is an array containing real variables. The size of the matrices vary from 2x2 to 25x25; the average size is 10x10. The matrices are to be inverted using a program MATINV contained in a partitioned data set MATPROGS. Each inverted matrix is to be written as a single record on the data set SCIENCE.MATH.INVMATRS. The first variable in each record is to denote the size of the matrix. Each matrix is to be printed.

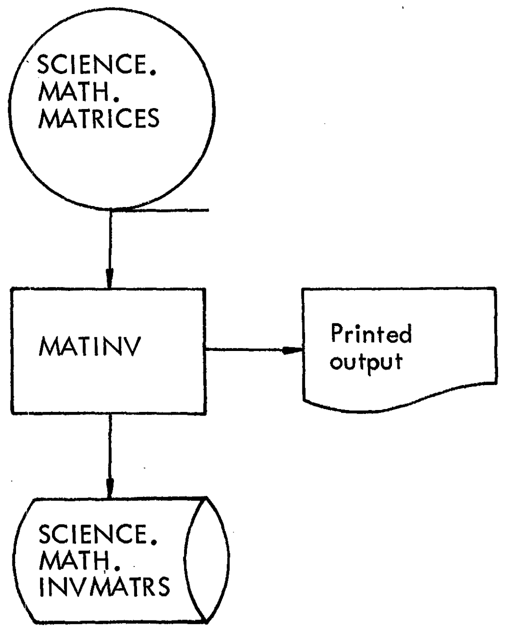
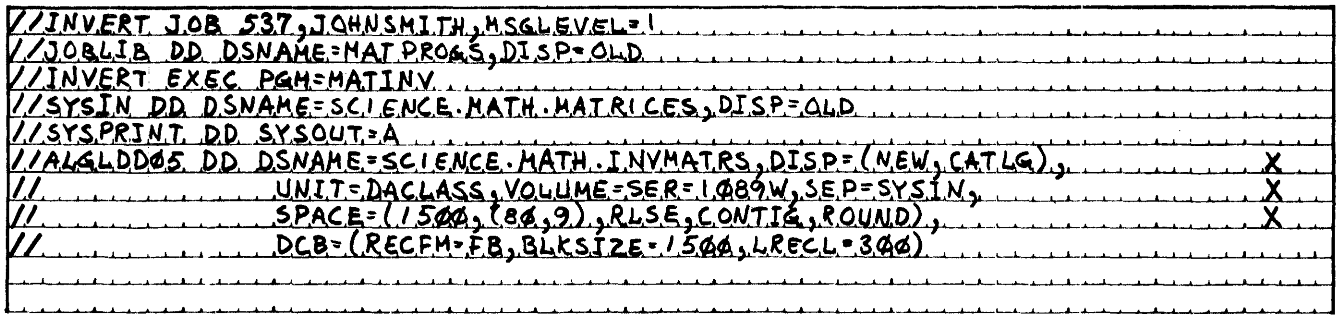


Figure 26. I/O flow for Example 1.



```
//INVERT JOB 537,JOHNSMITH,MSGLEVEL=1
//JOBLIB DD DSNAME=MATPROGS,DISP=OLD
//INVERT EXEC PGM=MATINV
//SYSIN DD DSNAME=SCIENCE.MATH.MATRICES,DISP=OLD
//SYSPRINT DD SYSOUT=A
//ALGLDD05 DD DSNAME=SCIENCE.MATH.INVMATRS,DISP=(NEW,CATLG),          X
//             UNIT=DACLASS,VOLUME=SER=1089W,SEP=SYSIN,                X
//             SPACE=(1500,(80,9),RLSE,CONTIG,ROUND),                  X
//             DCB=(RECFM=FB,BLKSIZE=1500,LRECL=300)
```

Figure 27. Job control statements for Example 1.

Explanation of coding: The job control statements used in Figure 27 specify that:

1. The job is

   - to be charged to the installation's account number 537
   - the responsibility of John Smith
   - to have all control statements (plus control statement diagnostic messages if an error occurs) printed on the normal system output device.

2. The partitioned data set MATPROGS is concatenated with the operating system library, SYS1. LINKLIB.

3. The program to be executed is MATINV.

4. The input data set is SCIENCE.MATH.MATRICES

5. The printed output is to use the standard output format class for the installation.

6. The output data set is

   - to be called SCIENCE.MATH.INVMATRS.
   - to be cataloged
   - to use the device class DACLASS
   - to use volume 1089W
   - to use a separate channel to the input data set
   - to have space reserved for 80 records, each 1500 bytes long. This space is to be incremented in 9-record units each time more is required and any unused space is to be released. The space is contiguous and aligned on cylinder boundaries.
   - to have fixed length blocked records, 300 bytes long, and a maximum block size of 1500 bytes.

<u>Example 2: Compiling, Linkage Editing and Executing Three Source Programs</u>

Statement of problem: Raw data from a rocket test firing is contained in a data set RAWDATA. The forecasted results for this firing are contained in a data set PROJDATA. A program PROGRD is to be used to produce refined data from these two data sets.

The refined data is to be stored in a temporary data set and used by a program ANALYZ, containing a series of equations, to develop values from which graphs and reports can be generated. Parameters needed by ANALYZ are contained on a cataloged data set PARAMS.

The values are to be stored on a temporary data set and used by a program REPORT to print graphs and reports. The programs PROGRD, ANALYZ and REPORT are written in ALGOL. They are still in source program form, and therefore must be compiled and linkage edited before execution.

Explanation of coding: The job control statements used in Figure 29 specify that:

1. The job is

   - the responsibility of John Smith
   - to have all control statements (plus control statement diagnostic messages if an error

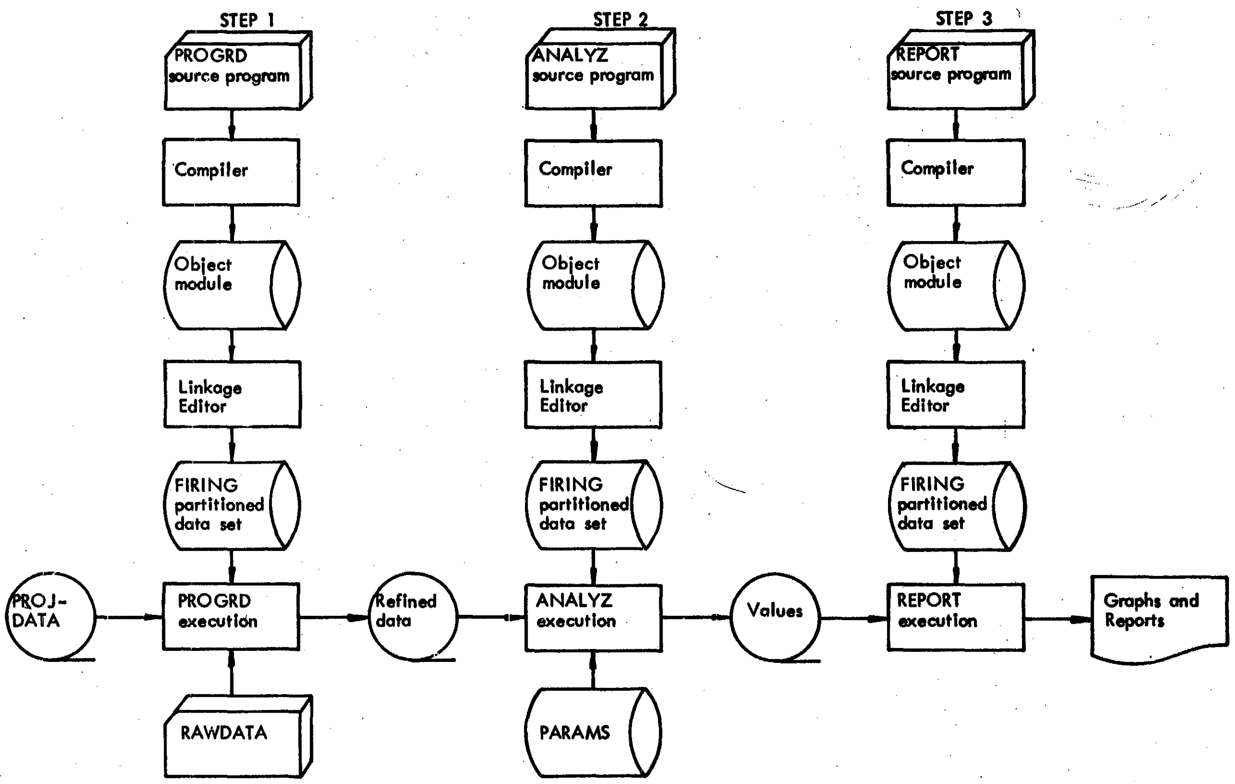


Figure 28. Basic I/O flow for Example 2. The data sets for information listings, ALGOL library routines intermediate work and the execution time error routine are not shown.

```
//TESTFIRE JOB ,JOHNSMITH,MSGLEVEL=1
//STEP1 EXEC ALGOFCLG
//SYSIN DD *
 SOURCE PROGRAM(PROGRD)
/*
//GO.ALGLDD11 DD DSNAME=PROJDATA,DISP=OLD
//GO.ALGLDD12 DD DSNAME=&REFDATA,DISP=(NEW,PASS),UNIT=TAPECLS,          X
//             VOLUME=(RETAIN,SER=21Ø7),                                X
//             DCB=(RECFM=F,BLKSIZE=4ØØ,LRECL=8Ø)
//GO.SYSIN DD *
 INPUT DATA(RAWDATA)
/*
//STEP2 EXEC ALGOFCLG
//SYSIN DD *
 SOURCE PROGRAM(ANALYZ)
/*
//LKED.SYSLMOD DD DSNAME=&GOSET(ANALYZ)
//GO.ALGLDDØ6 DD DSNAME=*.STEP1.ALGLDD12,DISP=OLD
//GO.ALGLDDØ7 DD DSNAME=PARAMS,DISP=OLD
//GO.ALGLDDØ3 DD DSNAME=&VALUES,DISP=(NEW,PASS),UNIT=TAPECLS,           X
//             DCB=(RECFM=F,BLKSIZE=2Ø4,LRECL=68),VOLUME=SER=21Ø8
//STEP3 EXEC ALGOFCLG
//SYSIN DD *
 SOURCE PROGRAM(REPORT)
/*
//LKED.SYSLMOD DD DSNAME=&GOSET(REPORT)
//GO.ALGLDD14 DD DSNAME=*.STEP2.ALGLDDØ3,DISP=OLD
```

Figure 29. Job control statements for Example 2.

occurs) printed on the normal system output device for information listings

2. The first job step invokes the ALGOFCLG cataloged procedure (see Appendix B) to process and execute the ALGOL source program (PROGRD) entered in the input stream

3. The other input data sets are RAWDATA and PROJDATA. RAWDATA is also entered in the input stream

4. The temporary output data set is

    - to be called REFDATA. TESTFIRE and to be passed for use in a later job step
    - to use the device class TAPECLS
    - to be written on volume 2107, which is to remain mounted for use later
    - to have fixed length records, 80 bytes long, and a maximum block size of 400 bytes

5. The second job step invokes the ALGOFCLG cataloged procedure to process and execute the ALGOL source program (ANALYZ) entered in the input stream

6. The SYSLMOD DD statement in the LKED step of the cataloged procedure is overridden to specify that the load module produced by the linkage editor is

    - to be a new member, PROGRD, of the temporary partitioned data set FIRING

7. The other input data sets are REFDATA. TESTFIRE and PARAMS. Both will be kept at the end of the job step

8. The temporary output data set is

    - to be called VALUES. TESTFIRE and is to be passed for use in a later job step
    - to use the device class TAPECLS
    - to be written on volume 2108
    - to have fixed length records, 68 bytes long, and a maximum block size of 204 bytes

9. The third job step invokes the ALGOFCLG cataloged procedure to process and execute the ALGOL source program (REPORT) entered in the input stream. The output data will be listed on the printer specified in the cataloged procedure

10. The SYSLMOD DD statement in the LKED step of the cataloged procedure is over-ridden to specify that the load module produced by the linkage editor is

    - to be a new member, REPORT, of the temporary partitioned data set FIRING

11. The other input data set is VALUES. TESTFIRE which will be kept at the end of the job step

<u>Example 3: Executing Two Load Modules</u>

Statement of problem: Data on current weather conditions is to be read from cards and used by the program FILECR to create a new generation of a data set WEATHER, and also to print a report.

Then the new generation and the three immediately preceding generations of the WEATHER data set are to be used by the program FORCST to produce a printed weather forecast. The pro-

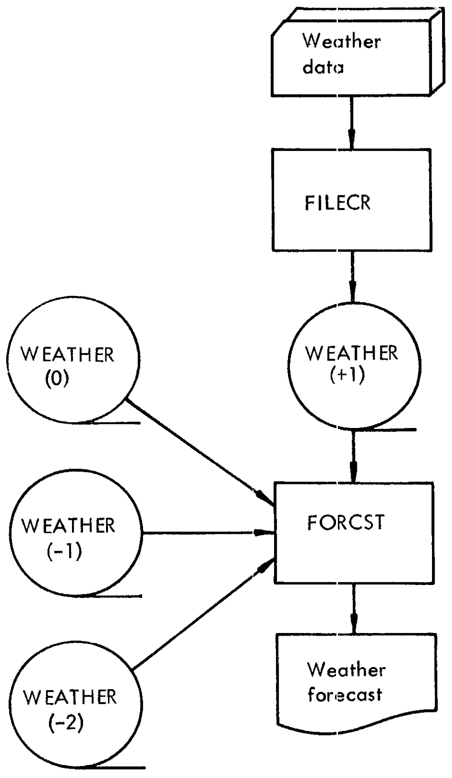
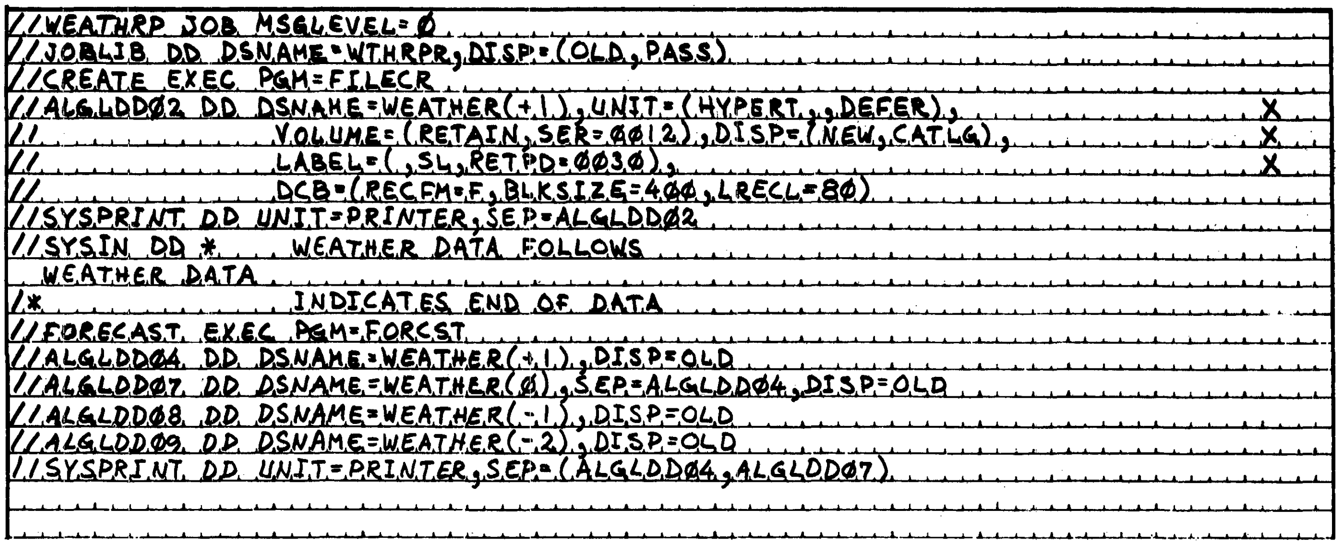


Figure 30. I/O flow for Example 3.



```
//WEATHRP JOB MSGLEVEL=Ø
//JOBLIB DD DSNAME=WTHRPR,DISP=(OLD,PASS)
//CREATE EXEC PGM=FILECR
//ALGLDDØ2 DD DSNAME=WEATHER(+1),UNIT=(HYPERT,,DEFER),                  X
//             VOLUME=(RETAIN,SER=ØØ12),DISP=(NEW,CATLG),               X
//             LABEL=(,SL,RETPD=ØØ3Ø),                                  X
//             DCB=(RECFM=F,BLKSIZE=4ØØ,LRECL=8Ø)
//SYSPRINT DD UNIT=PRINTER,SEP=ALGLDDØ2
//SYSIN DD *    WEATHER DATA FOLLOWS
 WEATHER DATA
/*              INDICATES END OF DATA
//FORECAST EXEC PGM=FORCST
//ALGLDDØ4 DD DSNAME=WEATHER(+1),DISP=OLD
//ALGLDDØ7 DD DSNAME=WEATHER(Ø),SEP=ALGLDDØ4,DISP=OLD
//ALGLDDØ8 DD DSNAME=WEATHER(-1),DISP=OLD
//ALGLDDØ9 DD DSNAME=WEATHER(-2),DISP=OLD
//SYSPRINT DD UNIT=PRINTER,SEP=(ALGLDDØ4,ALGLDDØ7)
```

Figure 31. Job control statements for Example 3.

grams FILECR and FORCST are contained in a partitioned data set WTHRPR.

Explanation of coding: The job control statements used in Figure 31 specify that:

1. The job is to have control statement messages plus the relevant control statement printed on the normal system output device only if an error occurs
2. The partitioned data set WTHRPR is concatenated to the operating system library, SYS1. LINKLIB
3. The first job step executes the program FILECR
4. The output data set is
   - a new generation of the data set WEATHER
   - to use the device class HYPERT
   - to be written on volume 0012 which need not be mounted until the data set is opened, and is then to remain mounted for later use
   - to be cataloged and have standard labels
   - to be retained for 30 days
   - to have fixed length records, 80 bytes long, and a maximum block size of 400 bytes
5. The printed output is
   - to use the device class PRINTER
   - to use a separate channel to the output data set
6. The input data is included in the input stream
7. The second job step executes the program FORCST
8. The input data sets are the last four generations of WEATHER, all of which are to be kept at the end of the job step
9. The output data set is
   - to use the device class PRINTER
   - to use a separate channel to the last two generations of WEATHER

APPENDIX F: DIAGNOSTIC MESSAGES

Each of the three operations-compilation, linkage editing and execution - may produce diagnostic messages.

COMPILER MESSAGES

The diagnostic messages that may be produced by the ALGOL compiler are given below. Each diagnostic message occupies one or more printed lines and contains:

- The message key, consisting of the letters IEX, a three digit decimal number identifying the message, and the letter I to indicate an informative message requiring no action from the operator.
- The severity code W, S or T (see below)
- The semicolon number (see Section 3). This number is sometimes omitted if the error cannot be directly related to a point in the program. The semicolon number is indicated in the list below by the sequence NNNNN
- The message text describing the error and, in the case of some W or S type errors, the modification performed on the program by the compiler. In the message text listed below the words in parentheses, together with the parentheses themselves, will be replaced in the actual message that is printed, by specific information taken from the program. The word "operator" usually refers to all delimiters defined in IBM System/360 Operating System: ALGOL Language, but an internal compiler operator may sometimes be listed. The word "operand" refers to an identifier or an expression.

The three severity codes for errors and their corresponding compiler action are as follows:

W (Warning): The program is modified internally and the compilation is continued. The modification may not make the program correct but it allows object module generation to continue. A diagnostic message is produced.

S (Serious): An attempt is made to modify the program internally, including skipping or changing parts of it. Generation of the object module is stopped, but syntax checking continues. A diagnostic message is produced.

T (Terminating): A diagnostic message is produced and the compilation is terminated.

IEX001I W NNNNN INVALID CHARACTER DELETED.

Explanation: A character not recognized by the compiler has been deleted from the program.

IEX002I W NNNNN ILLEGAL PERIOD. PERIOD DELETED.

Explanation: The character period has been used wrongly and deleted from the program. It can be used only as a decimal point, or as part of a colon or semicolon.

IEX003I W NNNNN INVALID COLON AFTER (six characters). COLON DELETED.

Explanation: The character colon has been used wrongly and has been deleted from the program. It can be used only after a label, between subscript bounds, within a parameter delimiter or as part of an assign symbol.

IEX004I T NNNNN LETTER STRING TOO LONG.

Explanation: A letter string used to supply explanatory information exceeds capacity limitations (see Section 4).

IEX005I S NNNNN IDENTIFIER BEGINS WITH INVALID CHARACTER. IDENTIFIER DELETED.

Explanation: An identifier has been deleted because it does not begin with an alphabetic character.

IEX006I T NNNNN LABEL CONTAINS TOO MANY CHARACTERS.

Explanation: A label identifier has been used whose length exceeds capacity limitations (see Section 4).

IEX007I W NNNNN LABEL BEGINNING WITH (up to six characters) CONTAINS INVALID CHARACTER. COLON DELETED.

Explanation: A label has been deleted because it contains a character of other than alphameric type.

IEX008I W NNNNN LABEL BEGINS WITH INVALID CHARACTER. COLON DELETED.

Explanation: A label has been deleted because it does not begin with an alphabetic character.

IEX010I S NNNNN SPECIFICATION PART OF PROCEDURE (identifier) INCOMPLETE.

Explanation: Not all of the formal parameters used in a procedure have been specified.

IEX011I S NNNNN PROGRAM STARTS WITH ILLEGAL DELIMITER.

Explanation: A program has been written not starting with 'BEGIN', 'PROCEDURE', 'REAL', 'INTEGER' or 'BOOLEAN'.

IEX012I W NNNNN TWO APOSTROPHES AFTER (six characters). FIRST APOSTROPHE DELETED.

Explanation: In this context, two apostrophes cannot be used together so one has been deleted.

IEX013I W NNNNN APOSTROPHE ASSUMED AFTER DELIMITER BEGINNING WITH (up to six characters).

Explanation: All delimiters involving words must begin and end with apostrophes. One has been left out of the program and has been inserted by the compiler.

IEX014I S NNNNN DELIMITER BEGINNING WITH (up to six characters) INVALID. FIRST APOSTROPHE DELETED.

Explanation: An invalid sequence of characters has been used after an apostrophe which apparently started a delimiter. The apostrophe is therefore deleted to remove the delimiter status from the characters but still include them in the program.

IEX015I W NNNNN MISSING SEMICOLON AFTER 'CODE'. SEMICOLON INSERTED.

Explanation: Self-explanatory.

IEX016I S NNNNN IDENTIFIER BEGINNING WITH (up to six characters) CONTAINS INVALID CHARACTER. IDENTIFIER DELETED.

Explanation: A character other than an alphameric type has been used in an identifier and so the identifier has been deleted.

IEX017I S NNNNN MORE THAN 65535 SEMICOLONS. SEMICOLON COUNTER RESET TO ZERO.

Explanation: Number of semicolons used exceeds capacity limitations. Duplicate numbers are allocated.

IEX018I W NNNNN DELIMITER 'COMMENT' IN ILLEGAL POSITION.

Explanation: 'COMMENT' has not been placed after a 'BEGIN' or a semicolon. Compilation continues normally.

IEX020I T NNNNN BLOCKS, COMPOUND STATEMENTS, FOR STATEMENTS, AND PROCEDURE DECLARATIONS NESTED TO TOO MANY LEVELS.

Explanation: Structure of program causes it to exceed capacity limitations (see Section 4).

IEX021I S NNNNN DECLARATOR (declarator) IN ILLEGAL POSITION.

Explanation: A declarator must come between either 'BEGIN' and the first statement of a block, or 'PROCEDURE' and the procedure body.

IEX022I T NNNNN MORE THAN 255 PROGRAM BLOCKS.

Explanation: Number of program blocks used exceeds capacity limitations.

IEX023I S NNNNN STRING POOL OVERFLOW.

Explanation: Total length of strings used exceeds capacity limitations (see Section 4).

IEX024I S NNNNN DELIMITER 'CODE' IN ILLEGAL POSITION. 'CODE' DELETED.

Explanation: 'CODE' has not been placed immediately after a procedure heading so it has been deleted.

IEX025I S NNNNN SPECIFIER 'STRING' OR 'LABEL' IN ILLEGAL POSITION. SPECIFICATION DELETED.

Explanation: 'STRING' and 'LABEL' have been used outside a procedure heading, so they have been deleted.

IEX026I S NNNNN PARAMETER (identifier) MULTIPLY SPECIFIED. FIRST SPECIFICATION USED.

Explanation: Self-explanatory.

IEX027I W NNNNN PARAMETER (identifier) MISSING FROM FORMAL PARAMETER LIST. SPECIFICATION IGNORED.

Explanation: A parameter has been specified in a procedure heading which does not exist in the formal parameter list, so it has been ignored.

IEX028I S NNNNN DELIMITER 'VALUE' IN ILLEGAL POSITION. VALUE PART DELETED.

Explanation: 'VALUE' has been placed outside a procedure heading so the value part has been deleted.

IEX029I W NNNNN SPECIFICATION PART PRECEDES VALUE PART.

Explanation: The specification part in a procedure heading has been incorrectly placed before the value part.

IEX030I W NNNNN PARAMETER (identifier) REPEATED IN VALUE PART.

Explanation: A parameter has been included in the value part of a procedure heading more than once.

IEX031I W NNNNN LEFT PARENTHESIS NOT FOLLOWED BY / AFTER ARRAY IDENTIFIER (identifier). SUBSCRIPT BRACKET ASSUMED.

Explanation: The subscript bounds after an array identifier have been preceded by a left parenthesis instead of a subscript bracket.

IEX032I S NNNNN MISSING RIGHT PARENTHESIS IN BOUND PAIR LIST OF ARRAY (identifier). DECLARATION DELETED.

Explanation: A right parenthesis has been omitted in the list of subscript bounds for an array identifier, so the declaration is deleted.

IEX033I T NNNNN MORE THAN 16 DIMENSIONS OR COMPONENTS IN DECLARATION OF (identifier).

Explanation: The number of dimensions or components used with an array or switch identifier exceeds the maximum allowed.

IEX034I S NNNNN ARRAY SEGMENT (identifier) NOT FOLLOWED BY SEMICOLON OR COMMA. CHARACTERS TO NEXT SEMICOLON DELETED.

Explanation: An array segment must be followed by a semicolon if it is the only or last segment of an array declaration; or a comma if it is followed by another segment.

IEX035I W NNNNN ILLEGAL PERIOD IN ARRAY OR SWITCH LIST. PERIOD DELETED.

Explanation: A period has been used wrongly in an array or switch list and deleted from the program. A period can be used only as a decimal point, or as part of a colon or semicolon.

IEX036I T NNNNN MORE THAN 15 PARAMETERS IN DECLARATION OF (identifier).

Explanation: The number of formal parameters specified for a procedure exceeds the maximum allowed.

IEX037I S NNNNN SEMICOLON MISSING AFTER FORMAL PARAMETER LIST OF (identifier). CHARACTERS TO NEXT SEMICOLON DELETED.

Explanation: The formal parameter list of a procedure must be followed by a semicolon.

IEX038I T NNNNN TOO MANY IDENTIFIERS DECLARED IN A BLOCK.

Explanation: Number of identifiers declared in a block exceeds capacity limitations (see Section 4).

IEX039I S NNNNN NNN MISSING 'END BRACKETS. OPEN BLOCKS, COMPOUND STATEMENTS, FOR STATEMENTS, AND PROCEDURE DECLARATIONS CLOSED.

Explanation: Syntax of ALGOL requires that a program contains the same number of 'BEGIN's and 'END's. The number of 'END's specified by NNN have been omitted in this case so any open blocks and statements are closed.

IEX041I T NNNNN MORE THAN 255 FOR STATEMENTS.

Explanation: Number of for statements used in a program exceeds capacity limitations.

IEX042I W NNNNN 'BEGIN' PRECEDES PRECOMPILED PROCEDURE. 'BEGIN' DELETED.

Explanation: A precompiled procedure has been specified so a 'BEGIN' is not required.

IEX043I S NNNNN EQUAL NUMBER OF 'BEGIN' AND 'END' BRACKETS FOUND. REMAINING PART OF PROGRAM IGNORED.

Explanation: The compiler assumes it has reached the end of the program when the number of 'END' brackets equals the number of 'BEGIN' brackets.

IEX044I T NNNNN NO SOURCE PROGRAM FOUND.

Explanation: For example, there has been an incorrect card code specification.

IEX045I S IDENTIFIER (identifier) MULTIPLY DECLARED. LAST DECLARATION USED.

Explanation: An identifier has been declared more than once in a program block heading. The last declaration is taken to be the one required.

IEX047I S ILLEGAL CALL BY VALUE OF IDENTIFIER (identifier).

Explanation: A procedure, switch or string has been wrongly called by value.

IEX080I S NNNNN OPERAND BEGINNING WITH (up to six characters) IS SYNTACTICALLY INCORRECT.

Explanation: Invalid characters have been used in the operand. If the six characters are all periods, this may indicate the internal representation of a string or logical value.

IEX081I S NNNNN IDENTIFIER (identifier) NOT DECLARED.

Explanation: An identifier has been used which is not declared in a block or procedure heading.

IEX082I S NNNNN REAL CONSTANT BEGINNING WITH (up to twelve characters) OUT OF RANGE.

Explanation: A real constant has been assigned a value which is outside capacity limitations.

IEX083I W NNNNN INTEGER BEGINNING WITH (up to twelve characters) OUT OF RANGE. INTEGER CONSTANT CONVERTED TO REAL.

Explanation: An integer constant has been assigned a value which is outside storage capacity limitations, so it has been converted to a real constant.

IEX084I W NNNNN PRECISION OF REAL CONSTANT BEGINNING WITH (up to twelve characters) EXCEEDS INTERNALLY HANDLED PRECISION. CONSTANT TRUNCATED.

Explanation: A real constant has exceeded capacity limitations regarding precision and has been truncated.

IEX085I S NNNNN ILLEGAL USE OF LABEL (label).

Explanation: A label defined in a for statement has been used in a goto statement outside the for statement, or the label occurs in a syntactically illegal position.

IEX086I S NNNNN TOO MANY CONSTANTS

Explanation: Number of constants used exceeds capacity limitations (see Section 4).

IEX087I W NNNNN FULL OPTIMIZATION NOT POSSIBLE DUE TO INTERNAL OVERFLOW.

Explanation: Main storage capacity available prevents for statement optimization by the compiler after the overflow occurs.

IEX088I W NNNNN IDENTIFIER (identifier) IN BOUND EXPRESSION DECLARED IN SAME PROGRAM BLOCK AS ARRAY. DECLARATION IN SURROUNDING BLOCK SEARCHED FOR.

Explanation: A bound expression can depend only on variables and procedures which are non-local to the block for which the array declaration is valid, because local variables do not have values before entering the statements of the block.

IEX089I W NNNNN 'GOTO' (identifier) INVALID OUTSIDE FOR STATEMENT CONTAINING THIS LABEL.

Explanation: A switch may have been misused, since a label has been found in a switch declaration outside a for statement containing a definition of the same label.

IEX160I S NNNNN SEQUENCE (operator) (operator) NOT ALLOWED.

Explanation: In this context, this sequence is not allowed.

IEX161I S NNNNN SEQUENCE (operator) OPERAND (operator) NOT ALLOWED.

Explanation: In this context, this sequence is not allowed.

IEX162I S NNNNN OPERAND MISSING BETWEEN (operator) AND (operator).

Explanation: In this context, there must be an operand between the two operators.

IEX163I S NNNNN OPERAND FOLLOWING (operator) MUST BE OF ARITHMETICAL TYPE.

Explanation: An arithmetical operand must follow an arithmetical operator.

IEX164I S NNNNN NO OPERAND ALLOWED BETWEEN (operator) AND (operator).

Explanation: In this context, no operand is allowed between the two operators.

IEX165I S NNNNN EXPRESSIONS BEFORE AND AFTER 'ELSE' NOT COMPATIBLE.

Explanation: For example, if an arithmetical expression is specified before 'ELSE', then an arithmetical expression must be specified after.

IEX166I S NNNNN DECLARATOR IN ILLEGAL POSITION.

Explanation: A declaration has occurred outside the block heading, or, for instance, a label precedes the declaration.

IEX168I S NNNNN OPERAND PRECEDING (operator) CANNOT POSSESS VALUE.

Explanation: Only quantities that can possess a value can be used in expression. For example, not standard I/O or non-type procedure identifier.

IEX169I S NNNNN LABEL FOLLOWING (operator) ILLEGAL.

Explanation: In this context, a label is not allowed due, for example, to a semicolon being missing.

IEX172I S NNNNN DIFFERENT TYPES IN LEFT PART LIST.

Explanation: The identifiers in a left part list must be of similar type.

IEX173I T NNNNN COMPILATION UNSUCCESSFUL DUE TO COMPILER OR MACHINE ERROR.

Explanation: Self-explanatory.

IEX174I S NNNNN PARAMETERS NOT ALLOWED FOR TYPE PROCEDURE CALLED BY VALUE.

Explanation: A type procedure called by value must have an empty parameter part.

IEX175I S NNNNN OPERAND FOLLOWING (operator) MUST BE LABEL OR SWITCH.

Explanation: For example, 'GOTO' must be followed by a designational expression.

IEX176I S NNNNN OPERAND MISSING BEFORE (operator).

Explanation: In this context, the operator must be preceded by an operand.

IEX177I S NNNNN OPERAND NOT ALLOWED BEFORE (operator).

Explanation: In this context, no operand is allowed before the operator.

IEX178I S NNNNN ILLEGAL OPERAND IN EXPRESSION BEFORE OR AFTER 'ELSE'.

Explanation: For example, only arithmetical operands may be used in an arithmetical expression.

IEX179I S NNNNN NUMBER OF SUBSCRIPT EXPRESSIONS DIFFERS FROM DIMENSION IN ARRAY DECLARATION FOR VARIABLE.

Explanation: A subscript list must contain the same number of subscript expressions as the dimension in the corresponding array declaration.

IEX180I S NNNNN INVALID SWITCH DESIGNATOR.

Explanation: More than one subscript expression in switch designator.

IEX181I S NNNNN SWITCH DESIGNATOR IN ILLEGAL POSITION.

Explanation: A switch designator must follow only 'THEN', 'ELSE', 'GOTO',:= or,.

IEX182I S NNNNN OPERAND FOLLOWING (operator) MUST BE BOOLEAN.

Explanation: A non-Boolean operand has been specified where a Boolean one was required.

IEX183I S NNNNN OPERAND PRECEDING (operator) MUST BE A PROCEDURE IDENTIFIER.

Explanation: A non-procedure identifier has been specified where a procedure one was required.

IEX184I S NNNNN OPERAND PRECEDING (operator) MUST BE AN ARRAY OR SWITCH IDENTIFIER.

Explanation: A non-array or non-switch identifier has been specified where an array or switch one was required.

IEX185I S NNNNN REAL OPERAND PRECEDING (operator) NOT ALLOWED FOR INTEGER DIVISION.

Explanation: A real operand has been specified for an integer division.

IEX186I T NNNNN SYNTACTICAL STRUCTURE TOO COMPLICATED. INTERNAL OVERFLOW.

Explanation: The syntactical structure of the program has caused an internal overflow in the compiler. A larger main storage size is required.

IEX187I S NNNNN INCORRECT NUMBER OF ACTUAL PARAMETERS.

Explanation: The number of actual parameters does not correspond to the number of formal parameters in a procedure.

IEX188I S NNNNN INVALID ACTUAL PARAMETER FOR STANDARD PROCEDURE. DSN= (number).

Explanation: An actual parameter has been specified incorrectly in a standard procedure. Either semicolon number or data set number is given. In the case where the data set number is given instead of the semicolon number, the error is due to SYSACT8 having been specified for the data set when SYSACT4, SYSACT13 or an input operation has been specified also. Such a combination is invalid.

IEX189I S NNNNN DATA SET NUMBER OR FUNCTION OF SYSACT OUT OF ALLOWED RANGE.

Explanation: Data set numbers are 0-15. SYSACT functions are 1-15.

IEX190I S NNNNN ASSIGNMENT NOT POSSIBLE.

Explanation: Only variable allowed in for clause. Only variable or type procedure identifier allowed in left part list.

IEX191I S NNNNN NO OPERAND ALLOWED BETWEEN) AND (operator).

Explanation: When a right parenthesis is used it must be followed by an apostrophe, a semicolon, an arithmetical operator, a comma, or another right parenthesis.

IEX192I S NNNNN INVALID RIGHT PART IN ASSIGNMENT STATEMENT.

Explanation: The right part must be either an arithmetic or a Boolean expression.

IEX193I S NNNNN INCOMPATIBLE TYPES IN ASSIGNMENT STATEMENT.

Explanation: Value assigned to right part does not correspond to type of left part list in assignment statement.

IEX194I S NNNNN (operator) NOT ALLOWED.

Explanation: In this context, the operator is not allowed.

IEX195I S NNNNN SEQUENCE OPERAND (operator) NOT ALLOWED.

Explanation: In this context, this sequence is not allowed.

IEX196I S NNNNN ARRAY IDENTIFIER PRECEDING (operator) NOT ALLOWED.

Explanation: In this context, an array identifier is not allowed.

IEX200I W NNNNN OPTION PARAMETER (parameter) INVALID. PARAMETER IGNORED.

Explanation: An invalid option has been specified in the PARM parameter and ignored by the compiler.

IEX201I T NNNNN DD CARD FOR (ddname) INCORRECT OR MISSING.

Explanation: One of the SYSIN, SYSPRINT, or SYSUT1, 2, 3 data sets used by the compiler has been specified incorrectly or not specified at all. This message is typed on the console typewriter when it concerns SYSPRINT.

IEX202I W NNNNN DD CARD FOR SYSLIN INCORRECT OR MISSING. OPTION NOLOAD ASSUMED.

Explanation: The SYSLIN data set has been specified incorrectly or not at all when the LOAD option is specified, so an object module is not generated.

IEX203I W NNNNN DD CARD FOR SYSPUNCH INCORRECT OR MISSING. OPTION NODECK ASSUMED.

Explanation: The SYSPUNCH data set has been specified incorrectly or not at all when the DECK option is specified, so an object deck is not punched.

IEX204I T NNNNN BLOCKSIZE SPECIFIED FOR SYSIN INCORRECT.

Explanation: The blocksize specified for SYSIN does not correspond to the actual blocksize.

IEX205I W NNNNN BLOCKSIZE SPECIFIED FOR (ddname) INCORRECT. UNBLOCKED OUTPUT ASSUMED.

Explanation: One of the output data sets has had an incorrect blocksize specified so unblocked output is generated (see Figure 6).

IEX206I W NNNNN TOO MANY OPTION PARAMETER ERRORS. SUBSEQUENT PARAMETERS IGNORED.

Explanation: Too many incorrect parameters have been specified in the PARM parameter so the rest are ignored.

IEX207I W NNNNN POSSIBLE ERROR IN DD NAMES PARAMETER.

Explanation: An incorrect ddname may have been specified in the DD statement.

IEX208I W NNNNN SIZE PARAMETER INVALID. SIZE 45056 ASSUMED.

Explanation: The main storage size specified as being available to the compiler is less than the minimum required, so the minimum value is assumed.

IEX209I T NNNNN COMPILATION UNSUCCESSFUL DUE TO PROGRAM INTERRUPT. PSW (hexadecimal digits).

Explanation: A program interrupt has occurred causing termination of the job step. The program status word when the error occurred is given.

IEX210I T NNNNN UNRECOVERABLE I/O ERROR ON DATA SET (ddname).

Explanation: An I/O error has occurred on the data set specified causing termination of the job. This message is typed on the console typewriter when it concerns SYSPRINT. This is most likely to be a random error, so the user is recommended to rerun the program.

IEX211I T NNNNN PROGRAM INTERRUPT IN ERROR MESSAGE EDITING ROUTINE. PSW (hexadecimal digits).

Explanation: A program interrupt has occurred in the error message editing routine, ending the job.

IEX212I T NNNNN TOO MANY ERRORS.

Explanation: The total length of the error message patterns produced exceeds capacity limitations.

IEX213I T NNNNN INTERNAL OVERFLOW OF IDENTIFIER TABLE.

Explanation: The number of identifiers declared exceeds capacity limitations.

IEX214I S NNNNN DATA STORAGE AREA EXCEEDED. PROGRAM BLOCK NO. (number).

Explanation: The data storage area required by the program block specified exceeds 4096 bytes.

IEX215I T NNNNN SOURCE PROGRAM TOO LONG.

Explanation: The capacity limitations (see

Explanation: The source program exceeds capacity limitations (see Section 4).

IEX216I S NNNNN TOO MANY LABELS. LABEL NUMBER RESET.

Explanation: The total number of labels used exceeds capacity limitations, so duplicated numbers are allocated (see Section 4).

LINKAGE EDITOR MESSAGES

Each message occupies one or more printed lines and contains:

- The message key, consisting of the letters IEW, a three digit decimal number identifying the message, and a final digit, either 1, 2, 3 or 4, indicating the severity code.
- The message text describing the error. For severity code 1 the message is preceded by 'WARNING'. For all other severity codes the message is preceded by 'ERROR'.

The severity codes have the following meaning:

1 indicates a condition that may cause an error during execution of the load module. A module map or cross-reference table is produced if it was required by the programmer. The output load module is marked as executable.

2 indicates an error that could make execution of the load module impossible. Processing continues. When possible, a module map or cross-reference table is produced if it was required. The load module is marked as not executable unless the LET option has been specified.

3 indicates an error that will make execution of the load module impossible. Processing continues. If possible a module map or cross-reference table is produced if it was required. The load module is marked as not executable.

4 indicates an error condition from which no recovery is possible. Processing terminates. The only output is diagnostic messages.

A full list of the linkage editor diagnostic messages is contained in IBM System/360 Operating System: Linkage Editor.

EXECUTION TIME MESSAGES

The list of diagnostic messages that may be produced by the load module is given below. Each message occupies one or more printed lines and contains:

- The message key, consisting of the letters IHI, a three digit decimal number identifying the message, and the letter I to indicate an informative message requiring no action from the operator.
- The characters SC = followed by the semicolon number (see Section 3). This number does not always indicate the statement in which the error occurred. For example, after a branch ('GOTO' or 'FOR'), if no semicolon has occurred before the error is detected, then the semicolon number preceding the branching instruction will be listed. For I/O errors, the semicolon number indicates the statement being executed when the error was detected, not the statement calling the I/O procedure.
- The message text describing the error. Where appropriate this begins by indicating the number of the data set (DSN) on which the error occurred, or the ddname if the data set does not have a number (that is, SYSUT1 and SYSUT2), or the program status word (PSW) held by the operating system when the error occurred. The PSW contains 16 hexadecimal digits. Message texts preceded by ** indicate that the program does not correspond with parameters specified in the job control cards.

IHI000I SC=NNNNN DATA SET NUMBER OUT OF RANGE

Explanation: A data set number must be in the range 0 to 15.

IHI001I SC=NNNNN DSN=NN. REAL NUMBER TO BE CONVERTED OUT OF INTEGER RANGE

Explanation: A real number has been included which exceeds capacity limitations when converted to integer. This message applies for input/output operations.

IHI002I SC=NNNNN DSN=NN. INCOMPATIBLE ACTIONS ON DATA SET

Explanation: The I/O operation requested is allowed only for an unblocked data set.

IHI003I SC=NNNNN DSN=NN. INPUT BEYOND LAST OUTPUT

Explanation: Before reading data which has just been written on the same data set, backward repositioning must be specified.

IHI004I SC=NNNNN TOO MANY REPOSITIONINGS IN DATA SETS. INTERNAL OVERFLOW

Explanation: Too many repositionings have caused an internal overflow of the Note Table (see Section 4).

IHI005I SC=NNNNN DSN=NN. INPUT REQUEST BEYOND END OF DATA SET

Explanation: Input has been requested to start beyond the end of the data set.

IHI006I SC=NNNNN DSN=NN. EXPONENT PART OF INPUT NUMBER CONSISTS OF MORE THAN TWO SIGNIFICANT DIGITS

Explanation: The length of the exponent part of an input number exceeds capacity limitations.

IHI007I SC=NNNNN DSN=NN. **NO CONTROL CHARACTER SPECIFIED IN RECORD FORMAT OF DATA SET. SPLITTING INTO SECTIONS IMPOSSIBLE

Explanation: A control character is required to define printing format.

IHI008I SC=NNNNN DSN=NN. SOURCE IN PROCEDURE OUTSYMBOL DOES NOT MATCH STRING

Explanation: The symbol specified by the third parameter of the OUTSYMBOL procedure does not correspond to any symbol in the string specified by the second parameter.

IHI009I SC=NNNNN DSN=NN. UNDEFINED FUNCTION NUMBER IN SYSACT PROCEDURE

Explanation: A function number has not been defined for a SYSACT procedure. The function number range is 1 to 15.

IHI010I SC=NNNNN DSN=NN. DATA SET CLOSED

Explanation: The data set is closed but a SYSACT procedure has been specified which requires it to be open.

IHI011I SC=NNNNN DSN=NN. DATA SET OPEN

Explanation: The data set is open but a SYSACT procedure has been specified which requires it to be closed.

IHI012I SC=NNNNN DSN=NN. QUANTITY IN SYSACT PROCEDURE MUST BE VARIABLE

Explanation: The third parameter of the SYSACT procedure must be a variable.

IHI013I SC=NNNNN DSN=NN. QUANTITY IN SYSACT PROCEDURE OUT OF RANGE

Explanation: The variable specified in the third parameter of the SYSACT procedure exceeds capacity limitations.

IHI014I SC=NNNNN DSN=NN. BACKWARD REPOSITIONING NOT DEFINED

Explanation: Backward repositioning is defined using SYSACT 13.

IHI015I SC=NNNNN UPPER BOUND LESS THAN LOWER BOUND IN ARRAY DECLARATION

Explanation: The upper subscript bound specified in an array declaration must not be less than the lower subscript bound.

IHI016I SC=NNNNN VALUE OF SUBSCRIPT EXPRESSION NOT WITHIN DECLARED BOUNDS

Explanation: This error is detected only when the subscripted variable address falls outside the area reserved by the compiler for the array identifier.

IHI017I SC=NNNNN ENDLESS LOOP IN FOR STATEMENT

Explanation: The expressions used in the for statement result in an endless loop.

IHI018I SC=NNNNN MAIN STORAGE REQUESTED NOT AVAILABLE

Explanation: The storage space required by an array exceeds capacity available.

IHI019I SC=NNNNN UNEQUAL NUMBER OF DIMENSIONS FOR ACTUAL AND FORMAL PARAMETER

Explanation: An array identifier being used as a parameter in a procedure has had a different number of dimensions assigned in the formal and actual positions.

IHI020I SC=NNNNN ACTUAL AND CORRESPONDING FORMAL PARAMETER OF DIFFERENT TYPE OR KIND

Explanation: An actual parameter has been assigned which does not have the type or kind declared for the corresponding formal parameter.

IHI021I SC=NNNNN UNEQUAL NUMBER OF PARAMETERS IN PROCEDURE DECLARATION AND PROCEDURE STATEMENT/FUNCTION DESIGNATOR

Explanation: Either not all, or more than, the formal parameters used in a procedure have been assigned in a procedure call.

IHI022I SC=NNNNN ASSIGNMENT TO A FORMAL PARAMETER NOT POSSIBLE

Explanation: A value cannot be assigned to an expression used in a standard input procedure, assignment statement, or for clause.

IHI023I SC=NNNNN ARGUMENT OF SQRT LESS THAN ZERO

Explanation: The ALGOL library SQRT routine cannot handle arguments with a value less than zero.

IHI024I SC=NNNNN ARGUMENT OF EXP GREATER THAN 174,673

Explanation: The argument of EXP exceeds capacity limitations.

IHI025I SC=NNNNN ARGUMENT OF LN NOT GREATER THAN ZERO

Explanation: A number not greater than zero cannot have a natural logarithm.

IHI026I SC=NNNNN ABS VALUE OF ARGUMENT OF SIN OR COS NOT LESS THAN PI*2**18

Explanation: The argument exceeds capacity limitations for a short precision real value.

IHI027I SC=NNNNN ABS VALUE OF ARGUMENT OF SIN OR COS NOT LESS THAN PI*2**50

Explanation: The argument exceeds capacity limitations for a long precision real value.

IHI028I SC=NNNNN PSW=XXXXXXXX XXXXXXXX. FIXED POINT OVERFLOW INTERRUPT

Explanation: An interrupt has occurred due to an overflow of a fixed point number.

IHI029I SC=NNNNN PSW=XXXXXXXX XXXXXXXX. FLOATING POINT EXPONENT OVERFLOW INTERRUPT

Explanation: An interrupt has occurred due to an overflow of a floating point exponent.

IHI030I SC=NNNNN PSW=XXXXXXXX XXXXXXXX. DIVISION BY ZERO. FIXED POINT

Explanation: An attempt has been made to divide a fixed point number by zero.

IHI031I SC=NNNNN PSW=XXXXXXXX XXXXXXXX. DIVISION BY ZERO. FLOATING POINT

Explanation: An attempt has been made to divide a floating point number by zero.

IHI032I SC=NNNNN DSN=NN. UNRECOVERABLE I/O ERROR

Explanation: An error has occurred on an input/output device. This message is printed on the console typewriter when the error occurs on SYSPRINT.

IHI033I SC=NNNNN PSW=XXXXXXXX XXXXXXXX. PROGRAM INTERRUPT

Explanation: A program interrupt has occurred.

IHI034I SC=NNNNN VALUE OF SWITCH DESIGNATOR NOT DEFINED IN DECLARATION OF SWITCH

Explanation: The designational expressions in the switch list of a switch declaration must define the values of all the corresponding switch designators.

IHI035I SC=NNNNN BASE NOT GREATER THAN ZERO

Explanation: Exponentiation is not defined in this case, because the base is zero or negative.

IHI036I SC=NNNNN TOO MANY NESTED BLOCKS AND CALLS OF PROCEDURES, SWITCHES, AND PARAMETERS. INTERNAL OVERFLOW

Explanation: Structure of program causes it to exceed the internal capacity limitations.

IHI037I SC=NNNNN DSN=NN. **BLOCKSIZE NOT A MULTIPLE OF LOGICAL RECORD LENGTH

Explanation: Blocksize must be an exact multiple of logical record length.

IHI038I SC=NNNNN DSN=NN TOO LONG RECORD

Explanation: Record is longer than specified.

IHI039I SC=NNNNN GET/PUT IDENTIFICATION OUT OF RANGE

Explanation: The identification number specified for a GET/PUT operation is out of range.

IHI040I SC=NNNNN REAL NUMBER TO BE CONVERTED OUT OF INTEGER RANGE

Explanation: A real number has been included which exceeds capacity limitations when converted to integer. This message applies to internal operations.

IHI041I SC=NNNNN DSN=NN. DD CARD INCORRECT OR MISSING

Explanation: One of the data sets used by the load module has been specified incorrectly or not at all. This message is printed on the console typewriter when the error occurs on SYSPRINT.

IHI042I SC=NNNNN INVALID OPTION PARAMETER

Explanation: An invalid option parameter has been specified in the PARM parameter.

IHI043I SC=NNNNN ILLEGAL CALL OF GET/PUT OR LIST PROCEDURE

Explanation: Recursive calls of GET/PUT or list procedures are not allowed.

INDEX

IBM **Technical Newsletter**

File Number	S360-26
Re: Form No.	C33-4000-0
This Newsletter No.	N33-8000
Date	October 31, 1967
Previous Newsletter Nos.	None

IBM System/360 Operating System
ALGOL Programmer's Guide

This technical newsletter amends the publication IBM System/360 Operating System: ALGOL Programmer's Guide, Form C33-4000-0. The attached pages replace pages in the publication. Corrections and additions to the text are noted by vertical bars to the left of the affected text. Revised figures are marked by the symbol ● to the left of the caption.

Pages to be Inserted	Pages to be Removed
1-2	1-2
5-8	5-8
11-18	11-18
21-22	21-22
25-26	25-26
33-34	33-34
37-50	37-50
53-54	53-54
59-60	59-60

In addition the following changes should be made.

Page	Amendment
28	In the left column, under "Termination Instructions", change "Reset" to "Restore".
29	Change "Reset" to "Restore" in all five positions that it appears.

Summary of Amendments

This newsletter corrects minor errors and omissions throughout the manual, primarily on the subject of Job Control Language.

It also includes estimates for specifying space on a direct access device for the SYSUT1, SYSUT2 and SYSUT3 data sets.

Note: Please file this cover letter at the back of the publication. Cover letters provide a quick reference to changes, and a means of checking receipt of all amendments.

File No. S360-26
Form C33-4000-0

OS

IBM Systems Reference Library

IBM System/360 Operating System

ALGOL Programmer's Guide

Program Number 360S-AL-531....Compiler
360S-LM-532....Library Routines

This publication describes how to compile, linkage edit and execute a program written in the System/360 Operating System Algorithmic Language (ALGOL). It includes an introduction to the operating system and a description of the information listings that can be produced, the job control language, and the subroutine library.

PREFACE

This publication is intended for use by Application Programmers, Systems Programmers and IBM Systems Engineers. A knowledge of ALGOL is assumed, and the reader is expected to be familiar with the prerequisite publication:

IBM System/360 Operating System: ALGOL Language, Form C28-6615.

In Section 2, the description "IBM-Supplied Cataloged Procedures" provides sufficient information to process and execute an ALGOL program that can use the IBM-supplied cataloged procedures without modification.

The rest of Section 2, together with information in Section 1 and the Appendices, will be required for programs that cannot use the IBM-supplied cataloged procedures without modification.

The description of information listings in Section 3 and the list of diagnostic messages given in Appendix F will be helpful in interpreting system output, especially for debugging.

An extensive index has been provided to assist the reader in using the manual for reference purposes.

This publication contains most of the information required by the Applications Programmer. The following publications are referred to within the text for information beyond the scope of this publication.

IBM System/360 Operating System:

Assembler Language, Form C28-6514

Linkage Editor, Form C28-6538

Job Control Language, Form C28-6539

Operator's Guide, Form C28-6540

Utilities, Form C28-6586

FORTRAN IV Library Subprograms, Form C28-6596

Message Completion Codes, and Storage Dumps, Form C28-6631

Supervisor and Data Management Services, Form C28-6646

Supervisor and Data Management Macro-Instructions, Form C28-6647

First Edition (April 1967)
Significant changes or additions to the specifications contained in this publication will be reported in subsequent revisions or Technical Newsletters.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Department 813, 112 East Post Road, White Plains, N.Y. 10601, or IBM Nordic Laboratory, Technical Communications, Vesslevägen 3, Lidingö, Sweden.

The primary constituent of a System/360 data processing operation is a job. This, basically, is the work that the user requires the computer to do. To carry out a job, a computer needs two types of information -- a program and data.

- A program (known in this context as a source program) is a sequence of instructions which specify the actions to be performed by the machine. These instructions are written in a symbolic language and are translated into machine language by a processing program contained in the operating system before they are performed.

- Data is the information to be processed by the program. The source program is regarded as data while it is being processed by operating system programs to make it suitable for execution.

From this brief introduction it can be seen that a job is affected by three separate factors -- the source program, the operating system and the machine configuration.

SOURCE PROGRAM

For jobs discussed in this publication, the source program will be written primarily in System/360 Operating System ALGOL (Algorithmic Language). This is defined in IBM System/360 Operating System: ALGOL Language. In addition the programmer must observe the restrictions, caused by internal capacity limitations, listed in Section 4.

An ALGOL source program may be written in freeform on any 80 column coding sheet. The program text is contained in columns 1 to 72. Columns 73 to 80 can be used by the programmer for program identification. To avoid confusion with job control statements (see "Operating System"), the character sequences // and /* must not be used in columns 1 and 2. It is possible to do this since these sequences are syntactically incorrect outside strings, and when they occur within strings, they may be shifted into non-critical columns by inserting a blank space before the opening string quotes '('. Two character sets are available for punching the source program into a card deck (see Appendix C).

For operations that require more precise control over the machine than can be provided by ALGOL, subprograms written in Assembler language can be included in the source program (see Section 4). Assembler language subprograms can also be used as a link to other languages, such as PL/I, COBOL and FORTRAN. The Assembler language is defined in IBM System/360 Operating System: Assembler Language.

OPERATING SYSTEM

The System/360 Operating System is a set of IBM-supplied, control and processing programs (supplemented if necessary by user-written programs) that assist the programmer to use the computer efficiently. The operating system selected for a particular installation is generated during the initial setting-up of the computer, by a process known as system generation.

Job Control

Operating system instructions (known as job control statements) must be added to the source program to control its handling within the operating system and to specify the data management facilities required.

These statements do not need to be specified until the program is ready to be executed. This means that the program can be prepared independent of installation considerations.

Six types of statements are available, which, in conjunction with associated parameters, can supply all information required by the operating system for job control. To save programming effort, commonly used sequences of control statements can be stored by the system for subsequent recall by identifying names. These sequences are known as cataloged procedures.

JOB is the first statement of each job. It indicates that a new job is beginning and, consequently, that the previous job has ended. A job can be divided into a number of job steps, which can be inter-related to improve processing efficiency. For example, the execution of one job step can be made dependent on the result of a previous one. This is an important feature of the operating system and users are recommended to exploit it as fully as possible.

EXEC (Execution) is the first statement in each job step. It specifies the program or cataloged procedure to be executed, and must be included even if the job consists of only one job step.

DD (Data Definition) is the statement used to describe a data set and to specify associated data control block information. It also specifies input/output (I/O) device assignment. One or more DD statements are usually required for each job step.

In addition the command statement is used to place operator commands into the input stream, the null statement indicates the end of the last job in the input stream, and the delimiter statement separates data from subsequent control statements when sequential scheduling is used. The command statement, when used, must immediately precede a JOB, EXEC or null statement.

The job control statements required for an ALGOL source program are described in Section 2.

Control Program

The control program is the primary program within the operating system and must be included with all installations. It is divided into a number of functions. Those affecting the applications programmer are described in the following text.

Job Scheduling

A job scheduler is included as part of the control program to control the flow of jobs and allocate the I/O devices required. Two forms of job scheduling are available.

With sequential scheduling the jobs are carried out in the order they are presented in the input stream to the computer.

With priority scheduling a summary of the input job stream is stored on a direct access device and jobs are carried out in order of priority (as specified in the JOB control statement). Any hold-up in the execution of a program, due, for example, to a delay in mounting a volume, will cause the job scheduler to select the next job available, in order of priority, and the revert back to the higher priority job when it is ready.

Supervisor

The supervisor is a set of subroutines, included in the control program, for transferring control of the central processing unit of the computer from one program to another and co-ordinating I/O operations. Initialization and termination of all programs described in this publication are achieved using the standard method given in IBM System/360 Operating System: Supervisor and Data Management Services.

Data Management

This sub-section is a summary of data management facilities. Full details are given in IBM System/360 Operating System: Supervisor and Data Management Services.

Data Sets: Data is usually stored on I/O devices and is only brought into main storage for processing. It is organized into data sets. These are collections of records that are logically related (for example, a set of test readings).

System/360 Operating System allows a data set to be identified and accessed by symbolic name only, without any reference to its location on the storage device. To do this the operating system builds a catalog of data set location against name. This catalog resides on one or more direct access volumes. A volume is one complete physical unit of storage such as a tape reel or a disk pack. It may contain a number of data sets, or alternatively one data set may stretch over a number of volumes. Data sets are created using DD statements.

Data Control Blocks: The operating system must be provided with information describing the characteristics of a data set before the data set can be processed. This information is assembled in the data control block associated with each data set. Data control blocks are automatically created for each data set that is to be processed by the program, and are completed from two sources:

1. Any information provided in the program is included first.

2. Information provided by the DD statement is then included, but this cannot over-ride any information stated in the program.

In the case of an existing data set, further information is taken from the data set label. Again, this cannot over-ride previously inserted information. Any DCB information provided by the programmer is checked by an appropriate routine to ensure its validity and to assign default values.

Data Set Labels: Data set labels, if requested by the programmer in the DD statement, are created by the operating system to store information relevant to the data set such as name and retention period. Tapes must have been previously initialized. The labels can supplement information in the data control block and serve as identifiers during accessing. They are positioned at the beginning and end of the data set.

Records and Blocks: Records are the smallest items of data which can be read or written separately. Their length can be specified as fixed, variable or undefined. The unit of length is known as a byte, which is normally equivalent to one character. For mechanical reasons it is necessary to have a fixed length gap between each record. This means that the smaller the average length of the records so the smaller the amount of information that can be stored in a given area of storage. To conserve space a number of records can be grouped together to form a block, which is treated as a single record for I/O operations. The complete block is read into main storage and then unblocked for the required record to be processed. Record format and blocksize are defined in the data control block. For fixed length records blocksize must be a multiple of record length. This multiplication factor is known as the blocking factor.

A control character can be specified for inclusion in each record of a data set. This selects carriage control when the data set is printed, or stacker when the data set is punched.

Data Set Organization: According to how they are going to be used, records can be organized within the data set in a number of ways, as described below. Only sequential organization can be used with ALGOL.

Sequential organization is a feature of I/O devices such as magnetic tapes. To access a particular record the data set must be read sequentially until the record is found. This is satisfactory for many applications where a large proportion of the records will be required on each run but could be time-consuming where data is being accessed randomly.

To avoid reading each record in turn the indexed sequential method is often employed, in which the location of the required record is found from an index at the beginning of its data set. On a disk pack the specification of a record location is broken down into two levels - cylinder and track. Each level has its own index. With large data sets up to three levels of master index can also be used. Overflow areas are provided for the primary storage area so that insertions can be made.

Alternatively, a data set can be partitioned into blocks of identical format called members. A directory is built up at the beginning of the data set so that each member can be accessed independently by specifying its name as a suffix to the data set name. This form of data set is described as a library.

Direct organization allows records to be stored and retrieved using an absolute or relative address (cylinder, head, track). For example, an algorithm could be used to determine the address from data in the record.

Access Language: When using assembler language, two access languages are available to store and retrieve records. The queued access language provides a full range of buffering and blocking facilities to improve processing efficiency. It can only be used with sequential and indexed sequential data sets.

The basic access language gives the programmer more direct control over the I/O device but does not provide buffering and blocking facilities. These must be constructed by the user (see IBM System/360 Operating System: Supervisor and Data Management Services.

Access Methods: The data set organization and access language used are combined to fully describe the method of handling a data set, for example, Queued Sequential Access Method, Basic Partitioned Access Method, etc. The access method is specified in the data control block.

Input/Output Devices: Data can be stored on a number of input/output devices depending, among other things, on the method of data set organization required. The devices most commonly used in scientific and engineering installations are:

Card readers and punches Printers (output only) Paper tape devices Magnetic tape devices	All data handled by these devices is sequentially organized.
Disk storage devices Data cell storage devices Drum storage devices	These are known as direct access devices and can be used for sequential, indexed sequential or partitioned organization.

A console typewriter is used for direct two-way communication between the operator and the operating system.

Areas of main storage known as buffers are used to provide overlapping of reading, writing and processing operations. The transfer of data between main storage and I/O devices is controlled through units known as channels.

Processing Programs

In addition to the control program, a number of processing programs may be included in the operating system depending on the requirements of the installation. To carry out a job that contains a source program written in ALGOL the following processing programs are required:

1. ALGOL compiler

2. Linkage editor

The ALGOL compiler processes the source program to translate it into machine language. The translated source program (known as the object module) is then processed by a linkage editor to combine any routines required from the ALGOL library (see Appendix A). The result of these two operations (known as the load module) is then loaded into main storage and control is passed to the load module so that it can be executed. The basic flowchart for handling an ALGOL source program is shown in Figure 1.

ALGOL Compiler

This processing program is available for the F level of main storage size, and requires a minimum of 44K bytes. If extra storage capacity is provided it is used to increase compiler capacity (see Figure 6).

Initialization and Termination: The standard method is used for initialization and termination of the compiler (see "Supervisor"). At the end of the compilation one of the following return codes is generated:

0 meaning normal conclusion. Object module has been generated unless both the NODECK and NOLOAD options (see Appendix E) are specified in the invoking statement. No diagnostic messages have been listed.

4 meaning object module has been generated unless both the NODECK and NOLOAD options are specified. Only warning diagnostic messages (severity code W) have been listed.

12 meaning process has been completed but a complete object module could not be generated due to a serious error. Diagnostic messages (severity codes S and possibly W) have been listed.

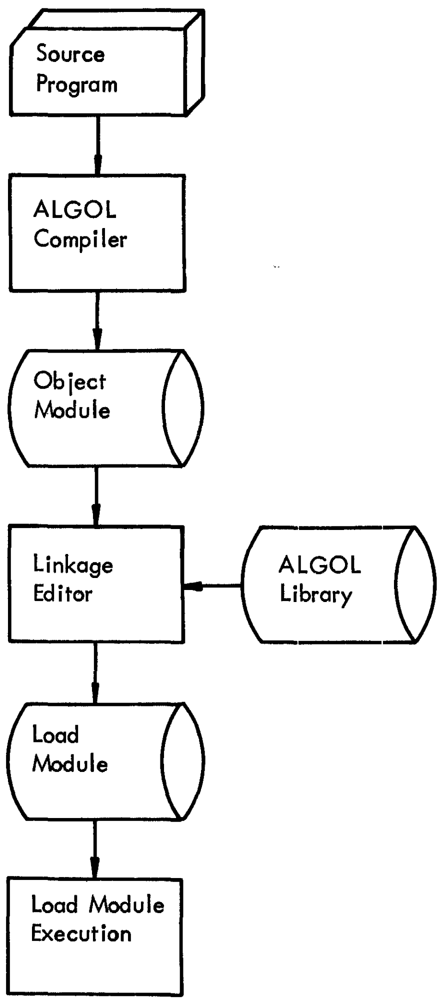


Figure 1. Basic flowchart for handling an ALGOL program.

16 meaning process has been terminated abnormally due to a terminating error. A complete object module could therefore not be generated. Diagnostic messages (severity codes T and possibly W and S) have been listed. The severity codes are described in Appendix F.

Output: A successful compilation of an ALGOL source program produces the following output:

- An object module (described in Appendix D) which can be:
  - Included in a data set for use as input to the linkage editor (optional).
  - Included in another data set to give some other form of output, such as a card deck (optional).

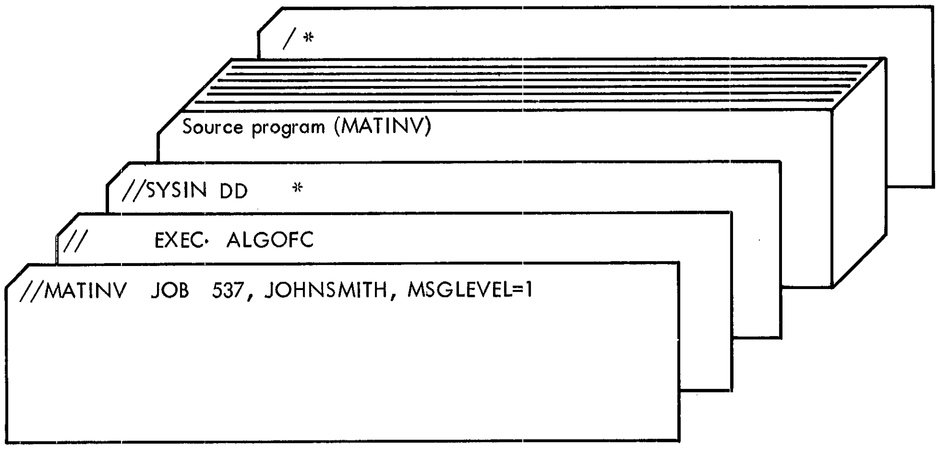


Figure 2. Sample deck for using ALGOFC cataloged procedure with a single source program. This job compiles the MATINV source program used in Example 1 of Appendix E.

If more than one source program is to be processed in the same job, then all job control statements except the JOB statement must be repeated for each source program.

If it is required to keep a load module for use in a later job (as in the case when the load module is a precompiled procedure), then the SYSLMOD DD statement in the cataloged procedure must be over-ridden to specify a permanent data set. This has to be done for each load module that is kept. The over-riding statement is placed at the end of the job step to which it applies, and has the form:

```
//LKED.SYSLMOD  DD  DSNAME=dsname(member),
                    DISP=(MOD,KEEP)
```

where "dsname" is the name of a partitioned data set and "member" is the member name assigned to the load module on the partitioned data set.

A sample deck of job control statements to compile and linkage edit two source programs is shown in Figure 3.

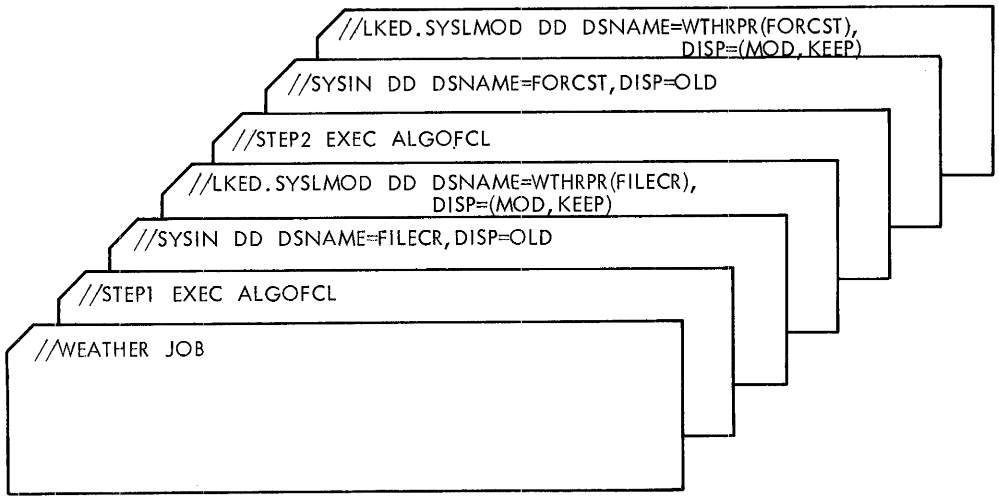

//LKED.SYSLMOD DD DSNAME=WTHRPR(FORCST),
DISP=(MOD, KEEP)
//SYSIN DD DSNAME=FORCST,DISP=OLD
//STEP2 EXEC ALGOFCL
//LKED.SYSLMOD DD DSNAME=WTHRPR(FILECR),
DISP=(MOD, KEEP)
//SYSIN DD DSNAME=FILECR,DISP=OLD
//STEP1 EXEC ALGOFCL
//WEATHER JOB

Figure 3. Sample deck for using ALGOFCL cataloged procedure with two source programs. These two job steps compile and linkage edit the two source programs used in Example 3 of Appendix E. Both source programs have been previously stored on intermediate I/O devices.

Compilation, Linkage Editing and Execution

The cataloged procedure used to compile an ALGOL source program, linkage edit the resulting object module, and execute the load module produced by the linkage editor is ALGOFCLG.

The statements used in this cataloged procedure are shown in Appendix B. The following statements can be used to invoke the ALGOFCLG cataloged procedure:

```
//jobname   JOB
//JOBLIB    DD DSNAME=dsname1, DISP=OLD
//          EXEC  ALGOFCLG
//SYSIN     DD   {* or parameters defining an
                    input data set containing
                    the source program }
//GO.ALGLDD02  DD  DSNAME=dsname2
                                 .
                                 .
                                 .
//GO.ALGLDD15  DD  DSNAME=dsname15
```

where "jobname" is the name assigned to the job. "dsname1" is the name of a data set that contains a precompiled procedure (see Section 4) which is called by the load module being executed. The DD statement containing dsname1 need not be used if no precompiled procedure is used.

For a description of the correct use of the JOBLIB DD statement when more than one precompiled procedure is used in a job, or when a precompiled procedure resides on more than one data set, see "Data Set Concatenation" in Appendix E.

"dsname2"... "dsname15" are the names of input data sets required by the load module at execution time and output data sets to be created at execution time. In addition, a data set for printed output (ddname SYSPRINT) is supplied by the cataloged procedure, and a data set for input only can be specified by using the following statement after the invoking sequence just given.

```
//GO.SYSIN  DD  {* or parameters defining an
                  input data set }
```

If DD* is used then the data must follow immediately afterwards in the input stream. For sequential scheduling, the data must be followed by a delimiter statement (/*).

If more than one source program is to be processed and executed in the same job, then all job control statements except the JOB statement and the JOBLIB DD statement must be repeated for each source program.

A sample deck of job control statements required to compile, linkage edit and execute three source programs is shown in Figure 29.

Over-riding Cataloged Procedures

The programmer can change any of the statements in a cataloged procedure, except the name of the program in an EXEC statement.

These over-riding conditions are temporary, and will be in effect only until the next job step is started. The following text describes methods of temporarily modifying existing parameters and adding new parameters to the EXEC and DD statements used in the cataloged procedures. The full list of parameters available to the ALGOL programmer for these statements, and detailed explanations of the parameters, is given in Appendix E. The EXEC and DD statements used in the IBM-supplied cataloged procedures are shown in Appendix B.

Over-riding EXEC Statements

In the EXEC statement, the programmer can change or add any of the keyword parameters by using the following format:

keyword. procstep=option

where:

"keyword" is the parameter to be changed in, or added to, the specified procedure job step: either COND, PARM, ACCT, TIME or REGION. TIME and REGION are valid only for priority scheduling.

"procstep" is the procedure job step in which the change or addition is to occur: either ALGOL, LKED or GO.

"option" is the new option required.

For example, if the EXEC statement used to invoke the ALGOFCLG cataloged procedure was written as:

```
// EXEC  ALGOFCLG,PARM.ALGOL=DECK,
//                 PARM.LKED=XREF,
//                 COND.GO=(3,LT,ALGOL)
```

then the following changes would be made to the ALGOFCLG cataloged procedure:

1. In the PARM parameter of the job step ALGOL, the option DECK would be used instead of the default option NODECK (assuming that the standard default NODECK was not changed at system generation). Over-riding this option will not affect the other default options assumed for this parameter.

2. In the job step LKED, the option XREF is specified for the PARM parameter. Since the options specified in the cataloged procedure were XREF, LIST and LET, this statement has the effect of deleting the options LIST and LET since they were not default options.

3. In the job step GO, the COND parameter code is changed from 5, as it appears in the cataloged procedure, to 3. In this example, the code 3 causes the job step GO to be bypassed if a warning message is generated during the job step ALGOL. Note that although the other options (LT and ALGOL) are not to be altered, the entire parameter being modified must be respecified.

If "procstep" is not specified when overriding a multi-step cataloged procedure, the operating system makes the following assumptions:

- COND, ACCT and REGION parameters apply to all procedure job steps.

- A PARM parameter applies to the first procedure job step and any options already specified in the PARM parameters for the remaining procedure job steps are cancelled.

- A TIME parameter specifies the computing time for the entire job and any options already specified in the TIME parameters for individual procedure job steps are cancelled.

Over-riding DD Statements

An additional DD statement is used in the invoking sequence for each DD statement in the cataloged procedure that is to be over-ridden. The following format is used:

```
//procstep.ddname  DD  parameter-list
```

where:

"procstep" is the procedure job step containing the DD statement to be over-ridden: either ALGOL, LKED or GO. If "procstep" is omitted then the first procedure job step is assumed.

"ddname" is the name of the DD statement to be over-ridden.

"parameter-list" is the list of parameters that are being added or changed. In both cases the whole parameter must be specified. Unchanged parameters in the original statement need not be specified. For example, the statement:

```
//ALGOL.SYSLIN  DD  SPACE=(400,(80,10))
```

will change the SPACE parameter of the SYSLIN DD statement in the ALGOL job step so that space will be allocated for 80 physical records instead of 40.

DD statements that are used to over-ride other DD statements in the cataloged procedures must be placed immediately after the EXEC statement invoking the cataloged procedure, and must be in the same order as their corresponding DD statements in the cataloged procedures.

Adding DD Statements

Complete, new DD statements that are to be added to the cataloged procedure use the same format as over-riding DD statements. The "ddname" specified must not exist in the job step specified by "procstep". These new DD statements must follow immediately after the over-riding DD statements which apply to the same procedure job step.

USER-WRITTEN PROCEDURES

The information required by the programmer to write his own job control procedures is given in the following text, and in Appendix E. Cataloging user-written procedures, or permanently modifying the IBM-supplied cataloged procedures, is accomplished using the IEBUPDAT utility program, described in IBM System/360 Operating System: Utilities. The statements required in user-written procedures are:

- An EXEC statement to invoke the program.

- DD statements to define the data sets used by the program.

Compilation

Invoking Statement

The ALGOL compiler consists of ten load modules contained in the link library, SYS1.LINKLIB, of the operating system. The compiler is activated

by invoking its first load module, named ALGOL, which then internally invokes the other load modules of the compiler.

The usual method of invoking the compiler is by means of an EXEC statement of the form:

```
//stepname EXEC PGM=ALGOL
```

where "stepname" is the name assigned to the job step (optional).

Other EXEC statement parameters may be included if required (see Appendix E).

(A method of dynamically invoking the compiler within a job step, by means of the CALL, LINK, XCTL or ATTACH macro-instructions, is described in Section 4.)

Data Sets Used

The data sets used in the compilation process are illustrated in Figure 4, and described in Figure 5. These data sets must be specified by the programmer with suitable DD statements.

Blocksize DCB information may be specified by the user for SYSIN, SYSLIN, SYSPRINT and SYSPUNCH. The maximum blocking factor depends on the main storage size available (see Figure 6). Record length is fixed at 80 bytes for SYSIN, SYSLIN and SYSPUNCH, and 91 bytes for SYSPRINT.

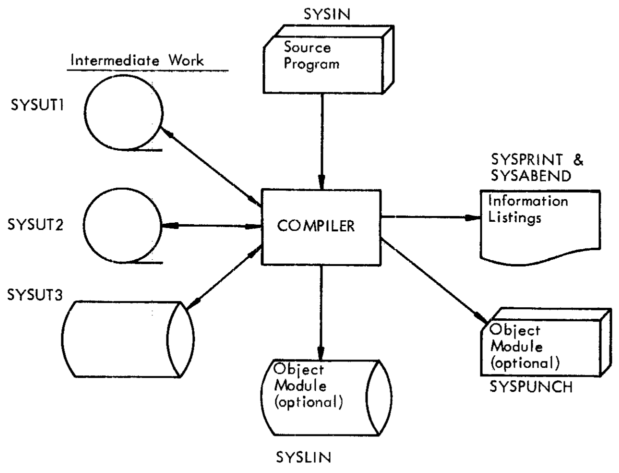


Figure 4. Flowchart showing data sets used by the compiler.

The space required for the compiler data sets depends on the size and structure of the source program, however it can be assumed that only in rare cases will the object module exceed four times the source program and usually much less will be required.

Purpose	Standard ddname	Devices required
For ALGOL source program	SYSIN	Card reader*
For object module to be used by linkage editor	SYSLIN	Direct access or magnetic tape
For compilation listings	SYSPRINT	Printer*
For object module (copied from SYSLIN)	SYSPUNCH	Card punch*
For the control program dump	SYSABEND	Printer*
For intermediate compiler working	SYSUT1	Direct access or magnetic tape
For intermediate compiler working	SYSUT2	Direct access or magnetic tape
For intermediate compiler working	SYSUT3	Direct access

* Some form of intermediate storage, such as magnetic tape, may be used to reduce I/O delay for the central processing unit.

Figure 5. Data sets used by the ALGOL compiler.

The primary quantity specified in the SPACE parameter of the DD statements for SYSUT1, SYSUT2 and SYSUT3 must be large enough to contain the entire data set. The use of a secondary quantity for any of these data sets will increase the need for main storage by 40%. The following estimates can be used to allocate space on a 2311 direct access device:

SYSUT1	-	1 track per 100 source cards
SYSUT2	-	1 track per 100 source cards
SYSUT3	-	1 track per 200 source cards.

SYSABEND is used for control program listings (see Section 3).

Processing of all data sets by the compiler is independent of the I/O device used except for the intermediate work data sets. These require magnetic tape or direct access devices.

Linkage Editing

Invoking Statement

The linkage editor is usually invoked with an EXEC statement of the form:

```
//stepname EXEC PGM=IEWL
```

where "stepname" is the name assigned to the job step (optional).

Other EXEC statement parameters may be included if required (see Appendix E). IEWL specifies the highest-level linkage editor in the installation's operating system.

(A method of dynamically invoking the linkage editor within a job step, by means of the CALL, LINK, XCTL or ATTACH instructions, is described in Section 4.)

Main storage size in bytes at which changes occur	Maximum blocking factor			
	SYSIN	SYSPRINT	SYSLIN	SYSPUNCH
45056 (44K)	5	5	5	1
51200 (50K)	5	5	5	5
59392 (58K)	5	5	5	5
67584 (66K)	5	5	5	5
77824 (76K)	5	5	5	5
90112 (88K)	20	20	40	20
104448 (102K)	20	20	40	20
120832 (118K)	20	20	40	20
139264 (136K)	20	20	40	20
159744 (156K)	20	20	40	20
184320 (180K)	40	40	40	40
212992 (208K)	40	40	40	40

Figure 6. Effect on compiler data sets if more than 44K bytes of main storage is available. The capacity of internal tables in the compiler is increased at each of the main storage sizes listed in this table, allowing, for example, a larger number of identifiers to be included in the source program. Therefore to get optimum performance, the user is recommended to use this list when specifying main storage size available to the compiler.

Data Sets Used

The data sets used by the linkage editor (see Figures 7 and 8) must be defined by the programmer with suitable DD statements.

Blocksize DCB information may be specified by the user for SYSLIN and SYSPRINT if the F level linkage editor is being used. Maximum blocking factor is 5 when 44K bytes of main storage size is available, and 40 when 88K bytes is available. Record length is fixed at 80 bytes for SYSLIN and 121 bytes for SYSPRINT.

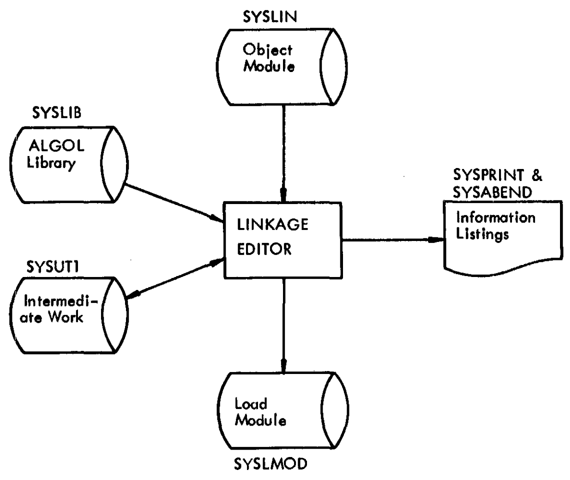


Figure 7. Flowchart showing data sets used by the linkage editor.

SYSABEND is used for control program listings (see Section 3).

Load Module Execution

Invoking Statement

The usual method of invoking the load module generated by the linkage editor is with an EXEC statement of the form:

//stepname EXEC PGM=member-name

Purpose	Standard ddname	Devices used
For object module input	SYSLIN	Direct access or magnetic tape
For load module output, stored as a member of a partitioned data set	SYSLMOD	Direct access
For ALGOL library, SYS1.ALGLIB. A partitioned data set containing routines in load module form	SYSLIB	Direct access
For linkage editing listings	SYSPRINT	Printer*
For intermediate linkage editor working	SYSUT1	Direct access or magnetic tape
For the control program dump	SYSABEND	Printer*

* Some form of intermediate storage, such as magnetic tape, may be used to reduce output delay for the central processing unit.

Figure 8. Data sets used by the linkage editor.

where "stepname" is the name assigned to the job step (optional).

"member-name" indicates the name of the partitioned data set member which contains the load module. This name is specified by the programmer in the SYSLMOD DD statement for the linkage editor. Other EXEC statement parameters may be included if required (see Appendix E).

(A method of dynamically invoking the load module within a job step, by means of the CALL, LINK, XCTL or ATTACH macro-instructions is described in Section 4.)

Data Sets Used

Up to 16 data sets for use at execution time may be specified by the programmer in the ALGOL source program by using the appropriate data set number. The numbers used and the corresponding names of their DD statements are listed below.

Data set number used in ALGOL source program	Corresponding ddname
0	SYSIN
1	SYSPRINT
2	ALGLDD02
3	ALGLDD03
4	ALGLDD04
5	ALGLDD05
6	ALGLDD06
7	ALGLDD07
8	ALGLDD08
9	ALGLDD09
10	ALGLDD10
11	ALGLDD11
12	ALGLDD12
13	ALGLDD13
14	ALGLDD14
15	ALGLDD15

Any reference to a data set number by an I/O procedure within an ALGOL source program is translated into a reference to a data control block using the corresponding ddname. It is the responsibility of the programmer to supply the DD statements which correspond to the data set numbers used in the ALGOL source program.

The execution time data sets are illustrated in Figure 9 and described in Figure 10. For ALGLDD02 to ALGLDD15, case 1 in the column showing device used, applies if the source program contains any of the following:

- A backward repositioning specification by the procedures SYSACT4 or SYSACT13 for this data set.
- Both input and output procedure statements for this data set.
- Procedure statements which prevent the compiler from recognizing whether either of these applies; for example, if the data set number or SYSACT function number is not an integer constant or if a precompiled procedure is used.

If the source program has already been compiled and linkage edited in a previous job, then the data set on which it has been stored (in load module form) must be concatenated to SYS1. LINKLIB. Data sets containing precompiled procedures called by the source program (see Section 4) must also be concatenated to SYS1. LINKLIB.

If the programmer specifies a TRACE, TRBEG or TREND option in the EXEC statement of the execution job step, the semicolon count (see Section 3) is stored intermediately on a data set with the ddname SYSUT1. The programmer must supply a corresponding DD statement if he uses this option. The semicolon count is converted to external form and transferred to the SYSPRINT data set as soon as the execution ends either by reaching the logical end of the source program or due to an error.

The space required for the semicolon count is:

For the main heading	6 bytes
For each semicolon	2 bytes
For each call of a precompiled procedure	12 bytes
For each physical record on SYSUT1	4 - 6 bytes

System/360 ALGOL permits data to be temporarily stored on and retrieved from external devices without conversion, using the ALGOL I/O procedures PUT and GET. If the programmer uses this facility in his source program, then he must supply a DD statement with the ddname SYSUT2. The device specified by this statement for storing such intermediate data should be a direct access device to guarantee reasonable performance, though programming is performed independently between magnetic tape and direct access devices. All data passed by a single PUT is

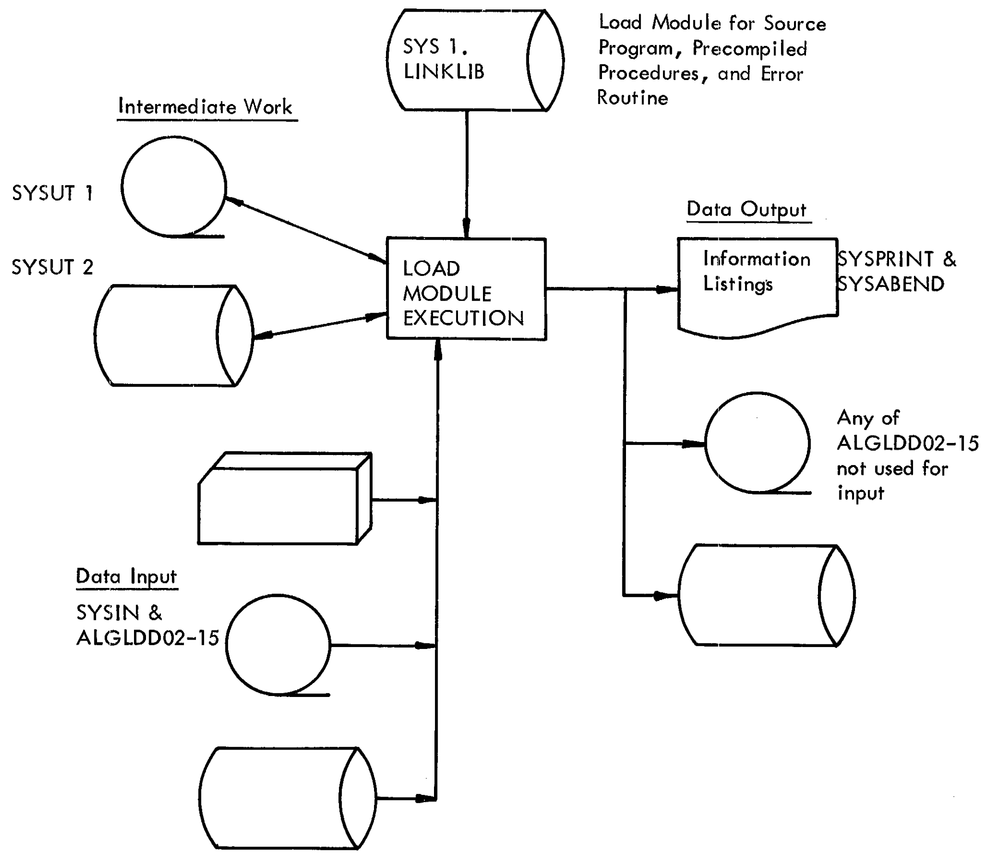


Figure 9. Flowchart showing data sets used at load module execution. The data input and output requirements are variable.

stored as one record. This record will be as long as the data passed, plus 8 bytes. The maximum record length accepted is 2048 bytes.

The DCB information which may be specified by the user for execution time data sets is blocksize, record format and record length (see page 44 for details), except for the trace and PUT/GET data sets (ddnames SYSUT1 and SYSUT2) for which only blocksize may be specified (up to a maximum of 2048 bytes).

For information not provided, default values will be inserted by a routine in the ALGOL library. In particular, blocksize is assumed as 2048 bytes for SYSUT1 and SYSUT2 if none is specified.

SYSABEND is used for control program listings (see Section 3).

	Standard ddname	Device Used
For data input to load module	SYSIN	Any input device
For execution time listings and data output	SYSPRINT	Printer*
For data input or output	ALGLDD02 : ALGLDD15	1. Direct access or magnetic tape 2. Any
For intermediate storage of semicolon counter when TRACE is specified	SYSUT1	Direct access or magnetic tape
For temporary storage when PUT is specified	SYSUT2	Direct access or magnetic tape
For the control program dump	SYSABEND	Printer*

* Some form of intermediate storage, such as magnetic tape, may be used to reduce I/O delay for the central processing unit.

Figure 10. Data sets used at execution time.

SECTION 3: INFORMATION LISTINGS

To assist the programmer to find the cause of any faults in the processing or execution of his program, various forms of information listings are produced for the compilation, linkage editing and execution operations. Some of these listings are optional. Examples are illustrated in Figures 11 to 16.

CONTROL PROGRAM LISTINGS

All three operations may produce listings generated by the control program. These are described in IBM System/360 Operating System: Messages, Completion Codes, and Storage Dumps. The ABEND macro-instruction for specifying the main storage dump is described in IBM System/360 Operating System: Control Program Services.

COMPILATION LISTINGS

A successful compilation of an ALGOL source program produces the following information listings:

- Job control statement information according to which MSGLEVEL option was specified in the JOB statement.
- The source program supplemented by a count of the semicolons occurring in the program (optional).
- A table giving details of all identifiers used in the program (optional).
- Any warning diagnostic messages.
- Information on main storage requirements at execution time.

If a serious diagnostic message is produced (meaning that object module generation has ended) then the source program and identifier table listings will be printed in full if they have been requested, but the information on main storage requirements will not be printed. If a terminating diagnostic message is produced then the source program and identifier table listings can be printed only as far as they have been produced.

Source Program

If the SOURCE option has been specified, the source program is transferred by the compiler to an output data set in order to be listed by a printer. This source program is supplemented by a semicolon count, which is referred to in the diagnostic messages to help localize errors.

The compiler generates this semicolon count when scanning the source program, by counting all semicolons occurring in the source program outside strings, except those following the delimiter 'COMMENT'. The value of this semicolon count at the beginning of each record of the source program is printed at the left of that record. It is assigned by the compiler in order to have a clear, problem-oriented reference. Any reference to a particular semicolon number refers to the segment of source program following the specified semicolon, for example, the semicolon number 5 refers to the program segment between the fifth and sixth semicolons.

Identifier Table

If the SOURCE option has been specified, a list of all identifiers declared or specified within the source program is transferred by the compiler to the output data set for printing after the source program listing. This identifier table gives information about the characteristics and internal representation of all identifiers. The identifiers are grouped together within the identifier table according to their scopes.

All blocks and procedure declarations within the source program are numbered according to the order of occurrence of their opening delimiters 'BEGIN' or 'PROCEDURE'. Therefore, if the body of a procedure declaration is a block, then usually this block has the same number as the procedure declaration itself. These numbers are called program block numbers (even if they belong to a procedure declaration and not to a block).

Each line in the table contains entries for up to three identifiers and the line begins with the number of the program block in which the identifiers were declared or specified, the value of the semicolon count at the commencement of the program block, and the number of the immediately surrounding program block. Each identifier entry contains:

- A cross-reference table of the load module, or alternatively, a module map (both optional).

If a diagnostic message of severity code 2 or 3 is produced then the other information listings might not be produced. If a diagnostic message of severity code 4 is produced then the other information listings will not be produced.

Diagnostic Messages

A description of the diagnostic messages that may be produced by the linkage editor is contained in Appendix F.

Module Map

If MAP is specified in the invoking statement for the linkage editor, then a module map is transferred to the output data set to be listed by a printer. The module map shows all control sections (the smallest separately relocatable units of a program) in the load module and all entry names (to routines in the ALGOL library) in each control section. The control sections are arranged in ascending order according to their origins (which are temporary addresses assigned by the linkage editor prior to loading for execution). The entry names are listed below the control section in which they are defined. The origins and lengths (in bytes) of the control sections, and the location of the entry names are listed in hexadecimal form. Unnamed control sections are identified by $ in the list.

At the end of the module map is the entry address of the instructions with which processing of the module begins. It is followed by the total length of the module, in bytes. Both values are in hexadecimal form.

Cross-Reference Table

If XREF is specified in the invoking statement for the linkage editor, the cross-reference table is transferred to the output data set to be listed by a printer.

The cross-reference table consists of a module map and a list of cross-references for each control section. In the list of cross-references, each address constant that refers to a symbol defined in another control section is listed with its assigned location (in hexadecimal form), the symbol referred to, and the name of the control section in which the symbol is defined.

If a symbol is unresolved after processing by the linkage editor, it is identified by $UNRESOLVED in the list. However, if an unresolved symbol is marked by the never call function, it is identified by $NEVER-CALL.

The entry address and total length are listed after the list of cross-references.

EXECUTION TIME LISTINGS

A successful execution of the load module produces the following information listings:

- Job control statement information according to which MSGLEVEL option was specified in the JOB statement.
- The ALGOL program trace, which is a list of the semicolon numbers assigned by the compiler (optional).

If an error is detected during execution of the load module, additional information listings are printed before the trace: these are;

- A diagnostic message
- The contents of the data storage areas (optional)

Diagnostic Messages

Any error detected at execution time causes abnormal termination. A diagnostic message is produced which is transferred to an output data set to be listed by a printer. The diagnostic messages which may be produced during load module execution are listed in Appendix F.

Data Storage Areas

If DUMP is specified in the invoking statement for the execution operation, the data storage areas (DSA) in main storage are transferred to the output data set to be listed by a printer. They are listed in the reverse order to which they were created.

A DSA is created for each call of a program block (see "Compilation Listings") and exists in main storage as long as the call is effective. The DSA contains:

1. All execution time values of variables declared or specified in the program block except for arrays. The array values are stored separately but are included in the listing because they are referenced by the SMF which is contained within the DSA.

2. Intermediate results (known as the object time stack).

The information listed for each DSA consists of:

- Name of load module
- Program block number
- Description of program block; either BLOCK, PROCEDURE or TYPE PROCEDURE
- The values in the DSA, in batches according to their category, that is, formal parameters, declared identifiers and object time stack, arrays called by value, and declared arrays.

The values are those which exist at the time the error was detected (in hexadecimal form). The displacement in the DSA of the first value in each line is printed at the beginning of each line. This is a six digit hexadecimal number.

For formal parameters, each entry has 16 digits, and in the case of parameters called by name the entry contains an address constant pointing indirectly to the value.

For declared identifiers and the object time stack, the identifier entries are listed first and they can be located using the identifier table if it was listed by the compiler. The object time stack contains various intermediate results and addresses which are not directly related to the identifiers in the source program.

For arrays the length depends on the SMF. The displacement of the SMF in the DSA is given for each array.

In the listings, real values have a length of 8 hexadecimal digits when SHORT is specified and 16 digits when LONG is specified. They are in standard floating point representation. Integer values have a length of 8 hexadecimal digits and are in standard fixed point representation. Boolean values have a length of 2 hexadecimal digits which appear as 00 for 'FALSE' and 01 for 'TRUE'.

An editing routine inserts blanks between each set of 8 digits to improve readability.

ALGOL Program Trace

A program trace, listing the semicolon numbers assigned by the compiler (see "Compilation Listings") in the order the corresponding semicolons were encountered during execution, is transferred to an output data set to be listed by a printer if TRACE, TRBEG or TREND is specified in the invoking statement for the execution. The completeness of the trace depends on the option or options specified (see Appendix E). Only the semicolons actually passed through at execution time are included in the trace.

If a precompiled procedure is used in the program and TRACE is specified, then the semicolon numbers for the procedure are included in the correct position within the program. The appropriate load module name (first four characters only) is inserted at the beginning of the listings and each time a change occurs in the first four characters of the module name.

CAPACITY LIMITATIONS

In addition to those given in IBM System/360 Operating System: ALGOL Language, the following restrictions must be observed when writing an ALGOL source program:

Number of blocks and procedure declarations (NPB)	≤255
Number of for statements	≤255
Number of identifiers declared or specified in one block or procedure. F is at most twice the number of for statements occurring in that block	≤179-F for type procedures ≤180-F otherwise
Length of letter string serving as parameter delimiter	≤1024 letters when main storage size available is less than 50K, ≤2000 letters otherwise
Length of label identifer	≤1024 characters when main storage size available is less than 50K, ≤2000 characters otherwise
Length of source program	≤255K
Number of semicolons in the whole program	≤65535
Number of nested blocks, compound statements, for statements and procedure declarations	≤999
Number of labels declared or additionally generated by the compiler	≤1024

The compiler generates the following additional labels:

For each switch declaration	2
For each procedure declaration	2
For each procedure activation (including function designators)	1
For each 'THEN' and each 'ELSE'	1
For each for statement	at most L + 3 where L is the number of for list elements

Length of constant pool	≤(256 - NPB) x 4096 bytes

The requirements of components within the pool are

Integer constant	4 bytes
Real constant (SHORT)	4 bytes
Real constant (LONG)	8 bytes
String (in bytes)	2 + number of symbols of open string between the outermost string quotes

The constant pool is divided into blocks of 4096 bytes each. The first block contains the integer constants 0 to 15 (64 bytes). All strings together are restricted to fill not more than the rest of this block (4096 - 64 - 2S bytes, where S = number of strings).

No constant occurring more than once in the source program is stored twice in the same block; however, it may possibly be stored more than once in different blocks. Up to seven bytes may be left unused.

Length of data storage area for each block or procedure declaration	≤4096 bytes
Number of blank spaces serving as delimiters on I/O data sets	≤255
Number of records in a data set	≤32760

Number of records per section	≤255
Number of entries in the Note Table	≤127

(The Note Table stores information to retrieve records which may be required again later. An entry for a record is made each time the ALGOL I/O procedures PUT and SYSACT13 are executed, and each time an input operation, with backward repositioning, follows an output operation on the same data set.)

Identification number (N) used by PUT or GET	0≤N≤65535

INVOKING A PROGRAM WITHIN A JOB STEP

Any one of the four macro-instructions, CALL, LINK, XCTL or ATTACH, may be used to dynamically invoke the compiler, linkage editor and load module within a job step. This is an alternative to the more usual method of invoking a program by starting a job step with an EXEC statement. Full details of the four macro-instructions are given in <u>IBM System/360 Operating System: Control Program Services.</u>

To invoke a program with the CALL macro-instruction, the program must first be loaded into main storage, using the LOAD macro-instruction. This returns, in general register 15, the entry address which is used by the CALL macro-instruction. The instructions used could be:

```
LOAD     EP=member-name

LR       15,0

CALL     (15), (option-address), VL
```

To invoke a program with one of the LINK, XCTL or ATTACH macro-instructions would need instructions such as:

```
LINK     EP=member-name,

         PARAM=(option-address), VL=1

XCTL     EP=member-name

ATTACH   EP=member-name,

         PARAM=(option-address), VL=1
```

"member-name" specifies the name of the member of a partitioned data set which contains the program required.

For the compiler, member-name=ALGOL

For the linkage editor, member-name=IEWL

For the load module, member-name is specified by the programmer in the SYSLMOD DD statement for the linkage editor.

"option-address" specifies the address of a list containing the options required by the user. An address must be given even if no options are specified. The list must begin on a half-word boundary. The first two bytes contain a number giving the number of bytes in the remainder of the list. (If no options are specified this number must be zero). The list itself contains any of the options available to the PARM parameter in an EXEC statement (see Appendix E).

When using CALL, LINK or ATTACH to invoke the compiler, other ddnames may be used in place of the standard ddnames given in Section 2 for the data sets (except for SYSABEND), and an alternative page number (instead of the normal 001) may be specified for the start of output listings.

If alternative ddnames are used, then in the statement invoking the compiler, "option-address" must be followed by "ddname-address" giving the address of a list containing the alternative ddnames. If alternative page numbers are used, then "page-address" giving the address of a location containing the alternative page number must be placed after "ddname-address"; though if alternative ddnames are not required "ddname-address" may be replaced by a comma.

The ddname list must begin on a half-word boundary. The first two bytes contain a number giving the number of bytes in the remainder of the list. The list itself contains up to ten 8-byte fields, separated by commas, for specifying alternative ddnames for the data sets. As only seven data sets are used by the compiler, three of the fields are left blank. The alternative ddnames must be listed in the following order:

Purpose of data set	Standard ddname
Output of object module for linkage editor	SYSLIN

IHIOBA		For OUTBARRAY	70
IHIOBO		For OUTBOOLEAN	400
IHIOIN		For OUTINTEGER	410
IHIOST		For OUTSTRING	300
IHIOSY		For OUTSYMBOL	290
IHIOTA		For OUTTARRAY	120
IHIPTT		For a long precision INREAL or OUTREAL operation	270
IHISAT	IHCSATAN	For a short precision arctangent operation (ARCTAN)	200
IHISEX	IHCSEXP	For a short precision exponential operation (EXP)	280
IHISLO	IHCSLOG	For a short precision logarithmic operation (LN)	210
IHISOR		For a short precision OUTREAL operation	810
IHISSC	IHCSSCN	For a short precision sine or cosine operation (SIN or COS)	260
IHISSQ	IHCSSQRT	For a short precision square root operation (SQRT)	170
IHISYS		For SYSACT	1520

Figure 18. Table of ALGOL library modules. All are contained in SYS1.ALGLIB except IHIERR which is in SYS1.LINKLIB. For mathematical routines, the corresponding name in the FORTRAN IV library is also given.

APPENDIX B: IBM-SUPPLIED CATALOGED PROCEDURES

The three cataloged procedures for ALGOL that were introduced in Section 2 are contained in the procedure library, SYS1.PROCLIB, of the operating system. They consist of the job control statements listed below.

These procedures have been designed for an optimum job, and can be over-ridden by the user if he requires different or additional system support to that provided (see Section 2). In particular it should be noted that in these procedures the object or load module produced is stored on a temporary data set and will therefore be deleted at the end of the job.

Compilation, ALGOFC

```
//ALGOL EXEC PGM=ALGOL
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD UNIT=SYSCP
//SYSLIN DD DSNAME=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,DISP=(MOD,PASS),    X
//               SPACE=(4ØØ,(4Ø,1Ø))
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT2 DD UNIT=SYSSQ,SEP=(SYSUT1,SYSLIN,SYSPUNCH),                    X
//               SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT3 DD UNIT=SYSDA,SPACE=(2ØØØ,(2Ø,5)),                             X
//               SEP=(SYSUT1,SYSUT2,SYSLIN,SYSPUNCH)
//SYSABEND DD SYSOUT=A
```

Compilation and Linkage Editing, ALGOFCL

```
//ALGOL EXEC PGM=ALGOL
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD UNIT=SYSCP
//SYSLIN DD DSNAME=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,DISP=(NEW,PASS),    X
//               SPACE=(4ØØ,(4Ø,1Ø))
//SYSLMOD DD DSNAME=&GOSET,UNIT=SYSDA,DISP=(MOD,PASS),                  X
//               SPACE=(1Ø24,(5Ø,2Ø,1))
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT2 DD UNIT=SYSSQ,SEP=(SYSUT1,SYSLIN,SYSPUNCH),                    X
//               SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT3 DD UNIT=SYSDA,SPACE=(2ØØØ,(2Ø,5)),                             X
//               SEP=(SYSUT1,SYSUT2,SYSLIN,SYSPUNCH)
//SYSABEND DD SYSOUT=A
//LKED EXEC PGM=IEWL,PARM=(XREF,LIST,LET),COND=(5,LT,ALGOL)
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSNAME=*.ALGOL.SYSLIN,DISP=(OLD,DELETE)
//SYSLIB DD DSNAME=SYS1.ALGLIB,DISP=OLD
//SYSLMOD DD DSNAME=&GOSET(GO),UNIT=SYSDA,DISP=(MOD,PASS),              X
//               SPACE=(1Ø24,(5Ø,2Ø,1))
//SYSUT1 DD UNIT=SYSDA,SEP=(SYSLIN,SYSLIB,SYSLMOD),                     X
//               SPACE=(1Ø24,(5Ø,2Ø))
//SYSABEND DD SYSOUT=A
```

The object module is in a form acceptable as input to the linkage editor, that is, its records are card images having the format of ESD, RLD, TXT and END cards (see Figure 20). It is stored either on a data set (ddname SYSLIN) in the linkage editor library, or on an output data set (ddname SYSPUNCH), or on both. The parameters LOAD and DECK, used to specify these storage options are described in Appendix E.

The object module consists of:

1. An initial ESD card defining the control section. For a precompiled procedure, the procedure name (up to 6 characters) is assigned to the control section and entered into this record.

2. The Constant Pool containing all constants and strings in the module.

3. The generated instructions.

4. The Label Address Table (see Section 3) for addressing branch instructions in the module.

5. The Program Block Table containing an entry for every program block. This table indicates the active generation of data storage areas (see Section 3) and length of each data storage area.

6. The Data Set Table containing information on the current status of all data sets used. This table is not produced for precompiled procedures.

7. Program start information.

8. An END card.

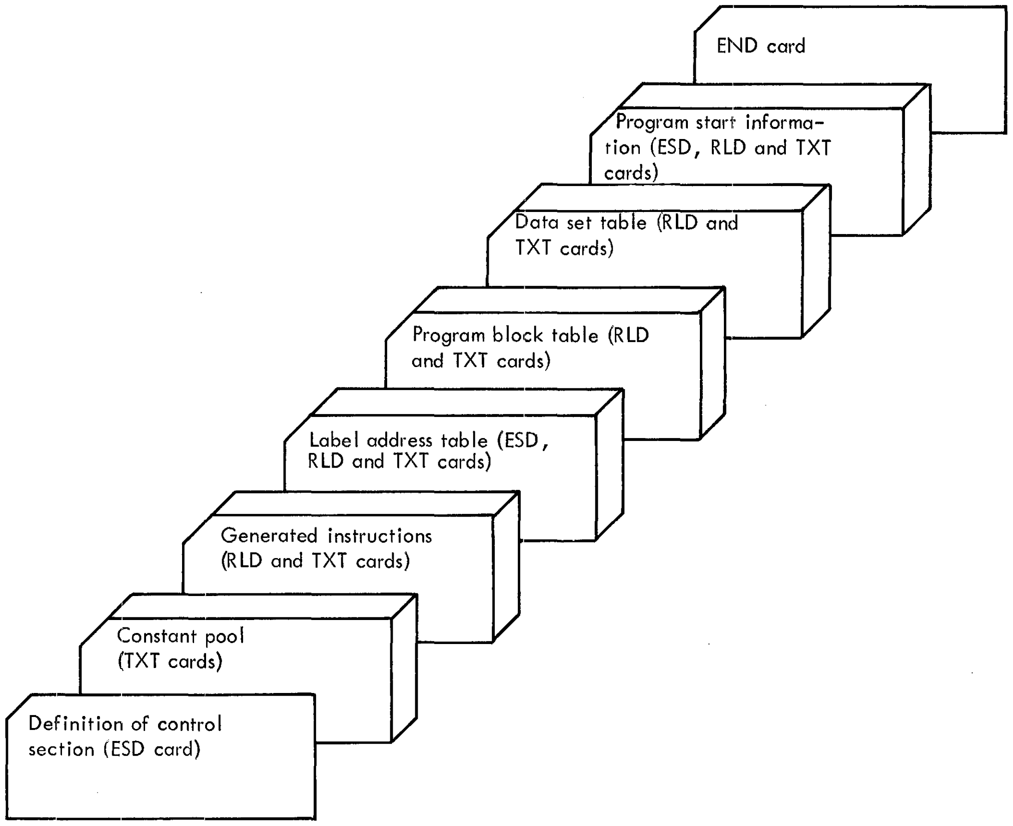

END card
Program start information (ESD, RLD and TXT cards)
Data set table (RLD and TXT cards)
Program block table (RLD and TXT cards)
Label address table (ESD, RLD and TXT cards)
Generated instructions (RLD and TXT cards)
Constant pool (TXT cards)
Definition of control section (ESD card)

Figure 20. The object module card deck. The ESD (External Symbol Dictionary) cards contain the external symbols that are defined or referred to in the module. The RLD (Relocation Dictionary) cards contain addresses used in the module. The TXT (Text) cards contain the constants and instructions used in the module. The END card indicates the end of the module.

APPENDIX E: USING JOB CONTROL LANGUAGE

This appendix describes the method of writing job control statements, and explains the options most frequently used by the ALGOL programmer. A full description of Job Control Language is given in IBM System/360 Operating System: Job Control Language.

Three types of operating system are available:

1. Primary Control Program (PCP), using a sequential scheduler,
2. Multiprogramming with a Fixed number of Tasks (MFT), using a sequential scheduler,
3. Multiprogramming with a Variable number of Tasks (MVT), using a priority scheduler.

CODING FORMAT

Control statements are identified by the initial characters // or /* and are written in columns 1 to 72 of standard 80 column punched cards. Each field is separated by one or more blanks. Column 72 must be left blank unless the statement is to be continued on another card.

If the length of a statement exceeds 71 characters, it must be continued on another card. This is done by interrupting the statement at the end of a positional or keyword parameter, following this parameter with a comma, and placing any non-blank character in column 72. The continuation card commences with the initial characters // and the statement restarts on column 16. Command statements may not be continued on another card.

Comments must be separated from the last parameter by one or more blanks. If the comment is to be continued on another card it may be interrupted at any convenient point and a non-blank character is put in column 72. The continuation card commences with the initial characters // and the comment restarts on any column from 16 to 71 inclusive.

Format	Applicable Control Statements
//NAME OPERATION OPERAND	JOB,EXEC, DD
// OPERATION OPERAND	EXEC,DD, Command
//	Null
/*	Delimiter

• Figure 21. Control statements formats.

The four possible formats for control statements are shown in Figure 21. The null and delimiter statements are blank except for the first two columns.

NAME contains the symbolic identification of the control statements. It is always placed immediately after the initial characters //. A name must contain between one and eight alphameric characters, the first of which must be alphabetic. If name is omitted, then at least one blank must separate the initial characters // and the operation field.

OPERATION identifies the type of control statement being specified.

OPERAND contains the statement parameters, separated by commas.

CONVENTIONS

The conventions used in this manual for describing control statements are as follows:

Upper case letters and punctuation marks (except those listed below) represent information to be coded exactly as shown.

Lower case letters are general terms requiring substitution of specific information by the programmer.

These punctuation marks have a special meaning:

- - (hyphen) links lower case words to form a single term for substitution
- _ (underscore) indicates the option that will be assumed if none is specified
- { } (braces) mean only one of the options contained must be selected
- [ ] (brackets) mean information contained may be omitted
- ... (ellipsis) means that preceding item can be repeated successively a number of times.

CONTROL STATEMENT CODING

In the following description, certain terms are used to indicate external names which are to be specified by the programmer. These terms and their meanings are:

Term	Meaning
jobname	name of job
progname	name of program

stepname	name of job step
ddname	name of DD statement (the standard ddnames which may be specified are described in Section 2)
procname	name of cataloged procedure
procstep	name of job step within a cataloged procedure
dsname	name of data set

It is often convenient to use two or more qualification levels to specify a data set name. The highest level reference is stated first. Thus in Figure 22, data set D.M.H is found by searching the index of each volume in turn, starting with the system residence volume (the primary volume in the operating system), to find the location of data set D. This, when searched, will contain the location of data set D.M, which in turn will contain the location of data set D.M.H.

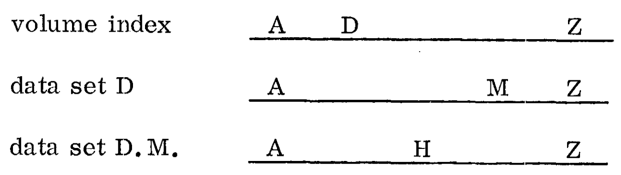


Figure 22. Data set cataloging using qualified names.

A maximum of 44 characters can be used for a qualified name. Thus, as a simple name can consist of between one and eight characters, and each name must be separated by the character period (.), a maximum of 22 qualification levels is possible.

Data set names can also be qualified by a suffix, that is, "dsname (element)", to indicate the relative generation number. For example, WEATHER (0) is the current generation of the data set named WEATHER. The preceding generation would be WEATHER (-1). A new generation during creation is known as WEATHER (+1), at the end of the job it becomes WEATHER (0). A suffix is also used to indicate the name of a member of a partitioned data set, or the area of an indexed sequential data set.

There are four types of job control parameters for inclusion in the operand fields: positional parameters, keyword parameters, positional subparameters and keyword subparameters.

Positional parameters must be stated first, and where more than one can be included they must be listed in the order given in the following descriptions. A comma must be substituted in place of any positional parameter omitted, if it is to be followed by another positional parameter, for example,

```
//name   operation   pos1,,pos3......
```

Keyword parameters can be listed in any order. They contain a keyword followed by an equal sign (=) and some specific information. All keyword parameters are optional since a default option will exist for any which must be specified.

One or more subparameters can be substituted for a positional parameter and also for the information to the right of the equal sign in the keyword parameter.

Positional subparameters have the same configuration and restrictions as positional parameters.

Keyword subparameters have the same configuration and restrictions as keyword parameters.

When more than one subparameters are used, they must be separated by commas and the list enclosed in parentheses, for example,

```
// name   operation   pos1,pos2,key1=value,
//                              key2=(sub1,sub2)
```

Since some special characters, such as the comma, parenthesis, blank and equal sign, have a special significance when used in control statements, no special characters can usually be used in job control information provided by the user. There are, however, some exceptions to this rule. The special characters @, $ and # can be represented normally. All other special characters, except the apostrophe, can be represented normally in the programmer's-name in the JOB statement, the accounting-information in the JOB and EXEC statements, and the PARM parameter options in the EXEC statement, provided that the information is enclosed in apostrophes (replacing the parentheses for a list of more than one subparameter). An apostrophe within this information is represented by two consecutive apostrophes.

JOB Statement

The name field of the JOB statement must contain the external name for the job (jobname).

The operation field must contain the characters JOB

The parameters available for the operand field are listed in Figure 23, where:

accounting-information
identifies the installation account number to which the computer time for this job is to be charged. If the installation has an appropriate accounting routine, the account number can be followed by other subparameters, which are fixed by the user for his own installation. If the account number is omitted then its absence must be indicated with a comma.

programmer's-name
identifies the person responsible for the job. It must not exceed 20 characters.

TYPRUN=HOLD
indicates that the job is not to be processed until a RELEASE command is issued by the operator. For priority scheduling only.

PRTY=job-priority
indicates the relative priority of the job. A number from 0 to 13 is specified, with 13 being the highest priority. This parameter can be used only with MFT or MVT systems.

Positional parameters	[accounting-information] [programmer's-name]

Keyword parameters (all optional)	CLASS=jobclass
	TYPRUN=HOLD
	PRTY=job-priority
	COND=((code, operator),...)
	MSGLEVEL= $\left\{\begin{matrix}\underline{0}\\1\end{matrix}\right\}$
	MSGCLASS=classname
	REGION=nnnnnK
	ROLL=($\left\{\begin{matrix}\underline{\text{YES}}\\\text{NO}\end{matrix}\right\}$, $\left\{\begin{matrix}\text{YES}\\\underline{\text{NO}}\end{matrix}\right\}$)

•Figure 23. JOB statement parameters.

COND=((code,operator),...)
allows conditions for the termination of the job to be specified. Up to eight (code,operator) specifications may be included in a COND parameter. Any number between 0 and 4095 is substituted for "code" and one of the following six relationships is substituted for "operator".

Operator	Meaning
GT	greater than
GE	greater than or equal to
EQ	equal to
NE	not equal to
LE	less than or equal to
LT	less than

At the completion of each job step, unless a system error occurs, the operating system will generate a return code between 0 and 4095 (see Section 1) to indicate if the program was executed successfully or not. If any of the code numbers stated in the COND parameter is related to the return code in the way specified by the associated operator then the job is terminated. For example, if

```
COND=((50,LT), (40,GT))
```

then, the job will be terminated if either 50 is less than the return code, or 40 greater than the return code.

MSGLEVEL=0
indicates that the job scheduler is to write out control statement information only when an error occurs. The information required is a diagnostic message and the control statement in which the error occurred.

MSGLEVEL=1
indicates that, whether an error occurs or not, the job scheduler is to write out all control statements, plus a diagnostic message if an error does occur.

MSGCLASS=classname
allows job scheduler messages to be written in a system output class other than the one normally used by the installation. The user can fix up to 36 different classes (A to Z and 0 to 9), depending on device type, priority, destination, etc., for these messages. This parameter is not necessary if the normal class (A)

is required. For PCP systems only class A may be used.

REGION=nnnnnK
indicates the main storage size that is to be allocated to the job (including system components) instead of the default value established in the input reader procedure. nnnnn is replaced by a value between 0 and 16384; thus 32 would represent 32 x 1024 = 32768 bytes. This parameter can be used only with priority scheduling.

CLASS=jobclass
indicates the relative class of a job in systems with MFT. "jobclass" is replaced by an alphabetic character, A through O.

$\text{ROLL}=(\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\},\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\})$
indicates the rollout/rollin attributes associated with a job in MVT systems. The first subparameter specifies if the job steps in this job can be rolled out to provide main storage space for job steps in other jobs. The second parameter specifies if the job steps in other jobs may be rolled out to provide main storage space for job steps in this job. The ROLL parameter can be specified in EXEC statements to control rollout/rollin for individual job steps.

EXEC Statement

The name field contains the external name of the job step (stepname). It may be omitted if no reference is to be made to the EXEC statement in another statement.

The operation field must contain the characters EXEC

The parameters available for the operand field are listed in Figure 24, where:

PGM=progname
indicates that the job step executes the program named "progname". The program must reside on a partitioned data set.

PGM=*. stepname. ddname
indicates that the job step executes the program named by the DSNAME parameter of a DD statement named "ddname" that was included in a previous job step named "stepname" in the same job. If "stepname" refers to a job step invoking a cataloged procedure then a job step within the procedure can be specified by putting its name after "stepname"; that is, "stepname. procstep". The program must reside on a partitioned data set.

PROC=procname
indicates that the job step executes the cataloged procedure named "procname".

procname
has the same effect as PROC=procname

TIME=(minutes, seconds)
limits the computing time for the job step. If "seconds" only is specified then a comma must be substituted for "minutes". If "minutes" only is specified then the parentheses can be deleted. This parameter can be used only with priority scheduling.

COND=((code, operator, stepname)....)
allows conditions to be specified for bypassing a job step whose execution depends on the return code issued by a preceding job step. "Code" and "operator" are governed by the same stipulations that applied for the JOB statement. "Stepname" indicates the previous job step which issued the return code to be used for comparison.

If "stepname" is not specified then the return code issued by all previous job steps are compared. If "stepname" refers to a job step invoking a cataloged procedure then a job step

Positional parameters	$\left\{\begin{matrix}\text{PGM=progname}\\ \text{PGM=*. stepname. ddname}\\ \text{PROC=procname}\\ \text{procname}\end{matrix}\right\}$
Keyword parameters (all optional)	$\left\{\begin{matrix}\text{TIME}\\ \text{TIME. procstep}\end{matrix}\right\}$=(minutes, seconds)
	$\left\{\begin{matrix}\text{COND}\\ \text{COND. procstep}\end{matrix}\right\}$=((code, operator, stepname), . . .)
	$\left\{\begin{matrix}\text{PARM}\\ \text{PARM. procstep}\end{matrix}\right\}$=subparameter-list
	$\left\{\begin{matrix}\text{ACCT}\\ \text{ACCT. procstep}\end{matrix}\right\}$=accounting-information
	$\left\{\begin{matrix}\text{REGION}\\ \text{REGION. procstep}\end{matrix}\right\}$=nnnnnK
	$\text{ROLL}=(\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\},\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\})$

• Figure 24. EXEC statement parameters.

within the procedure can be specified by putting its name after "stepname"; that is "stepname. procstep".

PARM=subparameter list

indicates any special conditions which apply to the job step. All the subparameters in the "subparameter-list" are optional. They can be specified in any order, and a comma does not have to be substituted for any omitted. A maximum of 40 characters may be used. For the rule to be observed when an equal sign is included in the subparameter-list (that is, with SIZE, TRBEG and TREND), see "Control Statement Coding".

For the ALGOL compiler job step, the "subparameter-list" is given below. For each of the alternatives, the compiler assumes that the option underscored applies, unless the other is specified either at this stage or during system generation. The default options PROGRAM and TEST cannot be changed at system generation. If a large number of options need to be specified for a particular job then the 40 character limitation may be exceeded. To avoid this, abbreviated forms, given at the end of the description of each option, may be used.

PROGRAM or PROCEDURE: which specifies that the source program is either an ALGOL program in the sense of the ALGOL syntax (PROGRAM), or is an ALGOL procedure to be compiled separately and used with other programs or procedures (PROCEDURE). Abbreviated forms PG or PC.

SHORT or LONG: which specifies that the internal representation of real values is in full words (SHORT); or double words (LONG). Abbreviated forms SP or LP.

NODECK or DECK: which specifies that an object module, stored on the data set specified in the SYSPUNCH DD statement, either is not to be generated (NODECK); or is to be generated (DECK). Abbreviated forms ND or D.

LOAD or NOLOAD: which specifies that the compiler is to either generate an object module for use as input to the linkage editor, using the data set specified in the SYSLIN DD statement (LOAD); or not generate this object module (NOLOAD). Abbreviated forms L or NL.

SOURCE or NOSOURCE: which specifies that the source program and identifier table listings are either to be printed (SOURCE); or not to be printed (NOSOURCE). Abbreviated forms S or NS.

EBCDIC or ISO: which specifies that the card code used to write and keypunch the source program is either a 53 character set in EBCDIC (EBCDIC); or the 46 character set in BCD which has been established as standard for ALGOL by ISO and DIN (ISO). Abbreviated forms EB or I.

TEST or NOTEST: which specifies that the generated object module is to include information which is normally used only for testing (TEST); or is not to include this information (NOTEST). The information consists of instructions to produce the semicolon count, and instructions checking the values of subscript expressions against array bounds. Abbreviated forms T or NT.

SIZE=45056 or SIZE=number: which specifies the main storage size, in bytes, that is available to the compiler. "Number" must not be less than 45056 and must not exceed 999999.

For the linkage editing job step the "subparameter-list" consists of two types of options, those which specify the output listings required, and those specifying attributes for the load module.

The options to control output listings are:

LIST which specifies that all job control statements processed by the linkage editor are to be listed on the diagnostic output data set.

MAP or XREF which specifies that either a map of the load module is to be produced (MAP); or a cross-reference table of the load module is to be produced (XREF) comprising a load module map and a list of all address constants that refer to other control sections.

The options specifying load module attributes which can be used with ALGOL programs are:

REUS which produces a load module that is serially reusable, that is, it can be used by more than one task, but only one task at a time.

DC which produces a load module that is downward compatible, that is, if the load module is produced by an F level linkage editor then it can be reprocessed by an E level linkage editor.

LET or XCAL which specifies that either the load module is to be marked as executable even when a severity 2 error is detected (LET); or the load module is to be marked as executable even though valid exclusive references between the segments have been made (XCAL). A severity 2 error could make execution impossible and would normally lead to the load module being marked as not executable. It includes the situation over-ridden by XCAL.

NCAL which specifies that the linkage editor automatic library call mechanism is not to call library members to resolve external references within the object module. The load module is marked as executable even though unresolved external references have been recognized.

All the linkage editor subparameters are optional.

For the execution job step of an ALGOL program the "subparameter-list" is:

TRACE which specifies that the semicolon count produced during the compilation process is to be printed as a list. This gives information on the dynamic flow of the program and is known as a program trace.

TRBEG=number which specifies that a limited program trace is to be produced beginning at the semicolon specified by "number" and ending at the physical end of the program.

TREND=number which specifies that a limited program trace is to be produced beginning at the physical beginning of the program and ending at the semicolon specified by "number".

The last two options may be specified together to define the beginning and end of the trace. When either is specified, TRACE may be omitted, but in that case precompiled procedures would not be included. If TRACE is specified with TRBEG or TREND, then only a limited program trace is produced, but it will include precompiled procedures executed in that part of the program. No program trace is possible if NOTEST has been specified for the compilation process.

DUMP which specifies that a partial main storage dump is to be produced if an error occurs. The dump contains the contents of the data storage areas and arrays.

All of the execution time subparameters are optional.

ACCT=accounting-information
allows accounting information associated with the job step to be passed to the installation's accounting routines, using subparameters which are fixed by the user for his own installation.

REGION=nnnnnK
indicates the main storage size for the job step if it has not already been specified in the JOB statement (see page 41).

keyword.procstep
is used with the last five parameters when a cataloged procedure is being executed. It indicates that the parameter applies to the job step named "procstep" within the procedure, and may be repeated for each keyword and with different, or the same, information to the right of the equal sign, for each job step in the procedure.

DD Statement

The name field contains an identifying name (ddname) for the DD statement.

The operation field must contain the characters DD

The parameters available for the operand field are listed in Figure 25, where:

* indicates, when used as a positional parameter, that the required data follows immediately after this DD statement. The asterisk must be the only non-blank character in the operand field. For sequential scheduling it can be used only once in each job step, and the data must be followed by a delimiter statement.

DUMMY
indicates that the user's problem program is to be executed without any I/O operations on the data set. This can be used for debugging, and also for bypassing data set references in a regularly-used program, for example, the first run of an updating program when there is no old master to be processed.

DSNAME=dsname (element)
specifies the name of a newly defined data set, or refers to one that has been defined previously. "Element" is used only if it is necessary to specify the generation number of the data set, the name of a member of a partitioned data set, or the area of an indexed sequential data set (using the options PRIME, OVFLOW or INDEX).

DSNAME=&name (element)
specifies that the data set is temporary and will be deleted before the end of the job. The name allocated by the operating system is "name. jobname". "Element" has the same meaning as when used with DSNAME=dsname.

DSNAME=*. stepname. ddname
indicates that the data set is the one specified in a preceding DD statement named "ddname" occurring in the job step named "stepname". If the data set was specified in the current job step then "stepname" must be omitted. If "stepname" refers to a job step invoking a cataloged procedure then a job step within the procedure can be specified by putting its name after "stepname"; that is "*. stepname. procstep. ddname".

Note. If the DSNAME parameter is omitted then the operating system will assign a unique name to any data set created by the job step.

Positional parameters (all optional)	$\left\{\begin{matrix}\text{*}\\ \text{DUMMY}\end{matrix}\right\}$

Keyword parameters (all optional, though DSNAME can be omitted only when the asterisk positional parameter is used).	DSNAME= $\left\{\begin{matrix}\text{dsname(element)}\\ \text{\&name(element)}\\ \text{*. stepname. ddname}\end{matrix}\right\}$
	DCB= $\left[\left\{\begin{matrix}\text{* . stepname. ddname}\\ \text{dsname}\end{matrix}\right\}\right]$ [subparameter-list]
	$\left\{\begin{matrix}\text{AFF=ddname}\\ \text{SEP=subparameter-list}\end{matrix}\right\}$
	UNIT=subparameter-list
	$\left\{\begin{matrix}\text{SPACE=subparameter-list}\\ \text{SPLIT=subparameter-list}\\ \text{SUBALLOC=subparameter-list}\end{matrix}\right\}$
	VOLUME=subparameter-list
	LABEL=subparameter-list
	$\left\{\begin{matrix}\text{DISP=subparameter-list}\\ \text{SYSOUT=subparameter-list}\end{matrix}\right\}$

Figure 25. DD statement parameters.

DCB= $\left\{\begin{array}{l}\text{*. stepname. ddname}\\ \text{dsname}\end{array}\right\}$ [subparameter-list]

indicates that the data control block for the data set specified in the DD statement named "ddname" in the job step named "stepname", or alternatively the cataloged data set named "dsname", is to be repeated for the current DD statement. "Stepname" must be omitted if it refers to the current job step, or may be qualified in the same way as the DSNAME parameter if it refers to a job step in a cataloged procedure. If additional information is substituted for "subparameter-list" then this over-rides the corresponding subparameters in the repeated information. Alternatively "subparameter-list" can be used alone to specify data control block information.

The "subparameter-list" for the data sets used when processing and executing an ALGOL program contains the following keyword subparameters:

BLKSIZE=number, is used to specify blocksize. "Number" is blocksize in bytes, and for fixed length records must be a multiple of record length.

RECFM=F [B] [A], is used to specify record format. F = fixed length, B = blocked, A = control character incorporated to control printed output format.

LRECL=value, is used to specify record length. "Value" is actual length in bytes.

All other valid DCB options are fixed.

AFF=ddname
indicates that the data set has affinity with the data set specified by the DD statement named "ddname" and is to use the same channel.

SEP=list-of-ddnames
indicates that the data set is to use a separate channel to the ones used by the data sets specified by the DD statements named in the "list-of-ddnames".

UNIT=subparameter-list
specifies the class and quantity of I/O devices to be allocated for use by a data set. The "subparameter-list" has two forms, either one of which may be used in an individual statement. The two forms are:

1	Positional subparameters	classname $\left\{\begin{array}{c}\underline{1}\\ \text{number}\\ \text{P}\end{array}\right\}$ [DEFER]
	Keyword subparameter	[SEP=list-of-ddnames]
2	Keyword subparameter	AFF=ddname

"classname" indicates the device class. These names are divided into two categories.

- Those automatically incorporated in the operating system when it is generated. These are of two types - specific unit names, such as 2400 (for a magnetic tape drive) and 1403 (for a printer); and general classnames, that is,

  SYSCP for any card punch
  SYSSQ for any magnetic tape or direct access device
  SYSDA for any direct access device.

- Additional names fixed by the user for his installation when the operating system is generated.

"number" indicates the number of devices to be allocated. If the data set is cataloged but the number of devices used is unknown, then "P" substituted for "number" will ensure that the correct number is assigned.

DEFER indicates that the volume need not be mounted on the I/O device until the data set is called in the program. This subparameter must not be used with an indexed sequential data set or a new output data set on a direct access device.

SEP=list-of-ddnames indicates for direct access devices that, if possible, the data set is not to use the same access arm as the data sets specified by the DD statements, given in the "list-of-ddnames".

AFF=ddname indicates that the data set is to use the same I/O devices as the data set specified in the DD statement named "ddname" in the same job step.

SPACE=subparameter-list
indicates the space required when a direct access device is specified in the UNIT parameter. The "subparameter-list" contains only positional subparameters. The list is:

$$\left\{\begin{array}{l}\text{TRK}\\ \text{CYL}\\ \text{average-record-length}\end{array}\right\}\text{primary-quantity}$$

[secondary-quantity][directory-or-index-quantity]

$$\text{[RLSE][}\left\{\begin{array}{l}\text{MXIG}\\ \text{ALX}\\ \text{CONTIG}\end{array}\right\}\text{][ROUND]}$$

The first subparameter specifies the units in which the space requirements are expressed, that is, tracks, cylinders or records (with length given in bytes).

The next subparameter specifies the space required. It has three parts (of which the second and third are optional) and is enclosed in parentheses if more than one part is specified. If the second part is omitted, then it must be substituted by a comma if the third part is included. The initial space to be allocated is given by "primary-quantity". Each time this initial space is filled, additional space is to be provided as specified by "secondary-quantity". The number of 256 byte records to be allocated for the directory of a new partitioned data set, or the number of cylinders, taken from the initial space reserved, to be allocated for the index of an indexed sequential data set, is given by "directory-or-index-quantity".

RLSE indicates that any unused space assigned to the data set is to be released.

MXIG requests that the largest single block of storage available is to be allocated to the data set.

ALX requests that extra blocks of storage (in track units) are to be allocated to the data set. As many available blocks that are equal to or greater than "primary-quantity", up to a maximum of five, will be allocated.

CONTIG specifies that the space specified by "primary-quantity" is to be in a single block.

ROUND requests that when records are used to express the space required on the direct access device, the space is to begin and end on cylinder boundaries.

DISP=subparameter-list
indicates the status of the data set and specifies its disposition at the end of the job step. The "subparameter-list" consists of the following positional subparameters:

$$\left\{\begin{array}{l}\underline{\text{NEW}}\\ \underline{\text{OLD}}\\ \text{MOD}\\ \text{SHR}\end{array}\right\}\left[\left\{\begin{array}{l}\text{DELETE}\\ \text{KEEP}\\ \text{PASS}\\ \text{CATLG}\\ \text{UNCATLG}\end{array}\right\}\right]$$

NEW specifies that the data set is to be generated in this job step, and would be deleted at the end of the job step unless KEEP, PASS or CATLG is specified.

OLD specifies that the data set already exists, and would be kept at the end of the job step unless PASS or DELETE is specified.

MOD specifies that the data set already exists and is to be modified in this job step. If the data set cannot be found by the operating system then this parameter is equivalent to NEW.

SHR specifies that, in a multiprogramming environment, an existing data set may be used simultaneously by more than one job.

DELETE specifies that the space used by the data set (including that in the data set catalog, etc.) is to be released at the end of the job step.

KEEP specifies that the data set is to be kept at the end of the job step.

PASS specifies that the data set is to be referred to in a later step of this job, at which

time its final disposition, or a further pass, will be specified.

CATLG specifies that the data set is to be cataloged at the end of the job step. Thus KEEP is implied. The catalog structure must already exist.

UNCATLG specifies that the data set is to be deleted from the catalog at the end of the job step. KEEP is implied.

SYSOUT=subparameter-list
specifies the printing or punching operation to be used for the data set. The "subparameter-list" is:

classname [progname][number]

"classname specifies the system output class to be used. Up to 36 different classes (A to Z, 0 to 9) may be fixed by the user for his installation, according to device type, priority, destination, etc. The standard classname is A.

"progname" can be used to specify the name of a user-written output routine.

"number" can be used to specify an installation form number to be assigned to the output.

For sequential scheduling, the "subparameter-list" consists of only the standard classname A.

VOLUME=subparameter-list
indicates the volume or volumes assigned to the data set. If the data set is cataloged this parameter is not necessary. The "subparameter-list" is:

Positional subparameters	[RETAIN][number][value]

Keyword subparameters	SER=list-of-serial-numbers
	REF= { dsname / *. ddname / *. stepname. ddname / *. stepname. procstep. ddname }

RETAIN specifies that, if possible, the volume is to remain mounted until referred to in a later DD statement, or until the end of the job, whichever is first. "number" is any number between 2 and 9999, and is used if an input or output operation, on a cataloged data set residing on more than one volume, does not start on the first volume of the data set. The number specifies the position of the volume on which input or output does start (for example, 3 indicates the third volume of the data set).

"value" specifies the number of volumes required by an output data set. It is not required if SER or REF is used.

SER=list-of-serial-numbers, specifies the serial numbers allocated by the user to the volumes required by the data set. These serial numbers can consist of between one and six alphameric characters.

```
     { dsname                       }
REF= { *. ddname                    }
     { *. stepname. ddname          }
     { *. stepname. procstep. ddname }
```

specifies that this data set is to use the same volume or volumes as the data set specified by one of the alternative sub-subparameter forms. If the latter data set resides on more than one tape volume, then only the last volume (as specified in the SER subparameter) can be used.

LABEL=subparameter-list
indicates the type of label or labels associated with the data set. If the data set is cataloged this parameter is not necessary. The "subparameter-list" is:

Positional subparameters	[number]	{ NL / <u>SL</u> / NSL / SUL / BLP }
Keyword subparameters	{ EXPDT=yyddd / RETPD=dddd }	

"number" is any number between 2 and 9999, and specifies the position of the data set on the volume (for example, 3 would indicate the third data set on the volume).

NL, SL, NSL, and SUL specify the type of label or labels to be used, that is, no labels, standard labels, non-standard labels, and standard

and user labels, respectively. The routines to produce non-standard labels must be written and incorporated into the operating system by the user. BLP indicates that label processing is to be bypassed.

EXPDT=yyddd specifies that the data set cannot be deleted or opened, without operator intervention, until the date given by yy (year) and ddd (day).

RETPD=dddd specifies that the data set is to be retained for the number of days given by dddd.

Command Statement

The options available for the operation and operand fields of the command statement are described in IBM System/360 Operating System: Operator's Guide.

DATA SET CONCATENATION

Unless it has been created in the same job, a load module specified for execution in an EXEC control statement must be contained in the SYS1. LINKLIB library of the operating system. If the load module is not a permanent member of this library then it is temporarily combined by using a DD statement with the name JOBLIB.

If the load module is a member of another library then this whole library is combined with the SYS1. LINKLIB library. This temporary combining is termed concatenation and lasts only for the duration of the job. A statement of this kind would have the form:

```
//JOBLIB DD DSNAME=dsname,DISP=OLD
```

where "dsname" is the name of the data set or library containing the load module to be executed.

Only one JOBLIB DD statement can be used in each job and it must immediately follow the JOB statement. If more than one load module contained in a library being concatenated is required in the same job then the parameter DISP=(OLD,PASS) placed immediately after the DSNAME parameter, will extend the effect of the concatenation through each step of the job.

If the job requires load modules from a number of data sets which are not created in the job or not permanent members of the SYS1. LINKLIB library then one data set is concatenated to this library, as described above, and the others are concatenated to this first data set by listing their DD statements immediately after the JOBLIB DD statement and leaving the name fields blank. This has the effect of concatenating all the data sets to the SYS1. LINKLIB library.

JOB CONTROL LANGUAGE EXAMPLES

Three different types of jobs are described here to illustrate the use of job control language. Some of the subparameters used, such as I/O device classnames and volume serial numbers, may change for different installations.

Example 1: Executing a Single Load Module

Statement of problem: A set of 80 matrices are contained in data set SCIENCE. MATH. MATRICES. Each matrix is an array containing real variables. The size of the matrices vary from 2x2 to 25x25; the average size is 10x10. The matrices are to be inverted using a program MATINV contained in a partitioned data set MATPROGS. Each inverted matrix is to be written as a single record on the data set SCIENCE. MATH. INVMATRS. The first variable in each record is to denote the size of the matrix. Each matrix is to be printed.

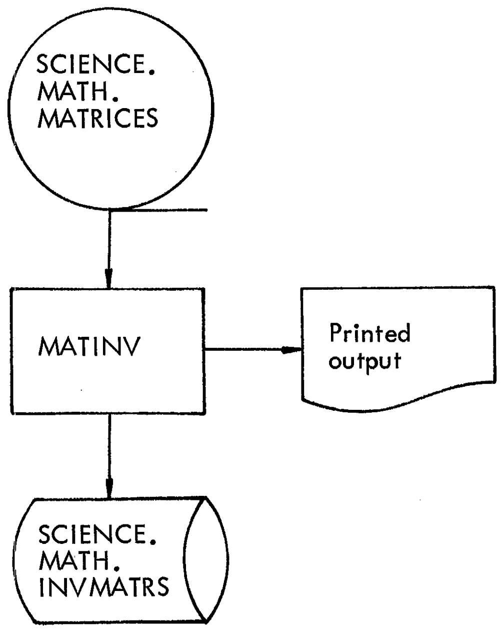
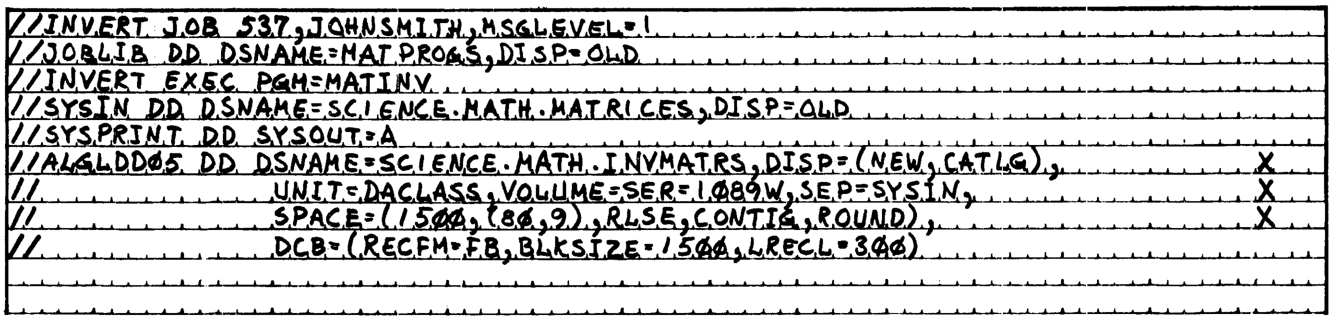


Figure 26. I/O flow for Example 1.



```
//INVERT JOB 537,JOHNSMITH,MSGLEVEL=1
//JOBLIB DD DSNAME=MATPROGS,DISP=OLD
//INVERT EXEC PGM=MATINV
//SYSIN DD DSNAME=SCIENCE.MATH.MATRICES,DISP=OLD
//SYSPRINT DD SYSOUT=A
//ALGLDD05 DD DSNAME=SCIENCE.MATH.INVMATRS,DISP=(NEW,CATLG),                X
//             UNIT=DACLASS,VOLUME=SER=1089W,SEP=SYSIN,                     X
//             SPACE=(1500,(80,9),RLSE,CONTIG,ROUND),                        X
//             DCB=(RECFM=FB,BLKSIZE=1500,LRECL=300)
```

Figure 27. Job control statements for Example 1.

Explanation of coding: The job control statements used in Figure 27 specify that:

1. The job is
   - to be charged to the installation's account number 537
   - the responsibility of John Smith
   - to have all control statements (plus control statement diagnostic messages if an error occurs) printed on the normal system output device.
2. The partitioned data set MATPROGS is concatenated with the operating system library, SYS1. LINKLIB.
3. The program to be executed is MATINV.
4. The input data set is SCIENCE. MATH. MATRICES
5. The printed output is to use the standard output format class for the installation.
6. The output data set is
   - to be called SCIENCE. MATH. INVMATRS.
   - to be cataloged
   - to use the device class DACLASS
   - to use volume 1089W
   - to use a separate channel to the input data set
   - to have space reserved for 80 records, each 1500 bytes long. This space is to be incremented in 9-record units each time more is required and any unused space is to be released. The space is contiguous and aligned on cylinder boundaries.
   - to have fixed length blocked records, 300 bytes long, and a maximum block size of 1500 bytes.

Example 2: Compiling, Linkage Editing and Executing Three Source Programs

Statement of problem: Raw data from a rocket test firing is contained in a data set RAWDATA. The forecasted results for this firing are contained in a data set PROJDATA. A program PROGRD is to be used to produce refined data from these two data sets.

The refined data is to be stored in a temporary data set and used by a program ANALYZ, containing a series of equations, to develop values from which graphs and reports can be generated. Parameters needed by ANALYZ are contained on a cataloged data set PARAMS.

The values are to be stored on a temporary data set and used by a program REPORT to print graphs and reports. The programs PROGRD, ANALYZ and REPORT are written in ALGOL. They are still in source program form, and therefore must be compiled and linkage edited before execution.

Explanation of coding: The job control statements used in Figure 29 specify that:

1. The job is
   - the responsibility of John Smith
   - to have all control statements (plus control statement diagnostic messages if an error

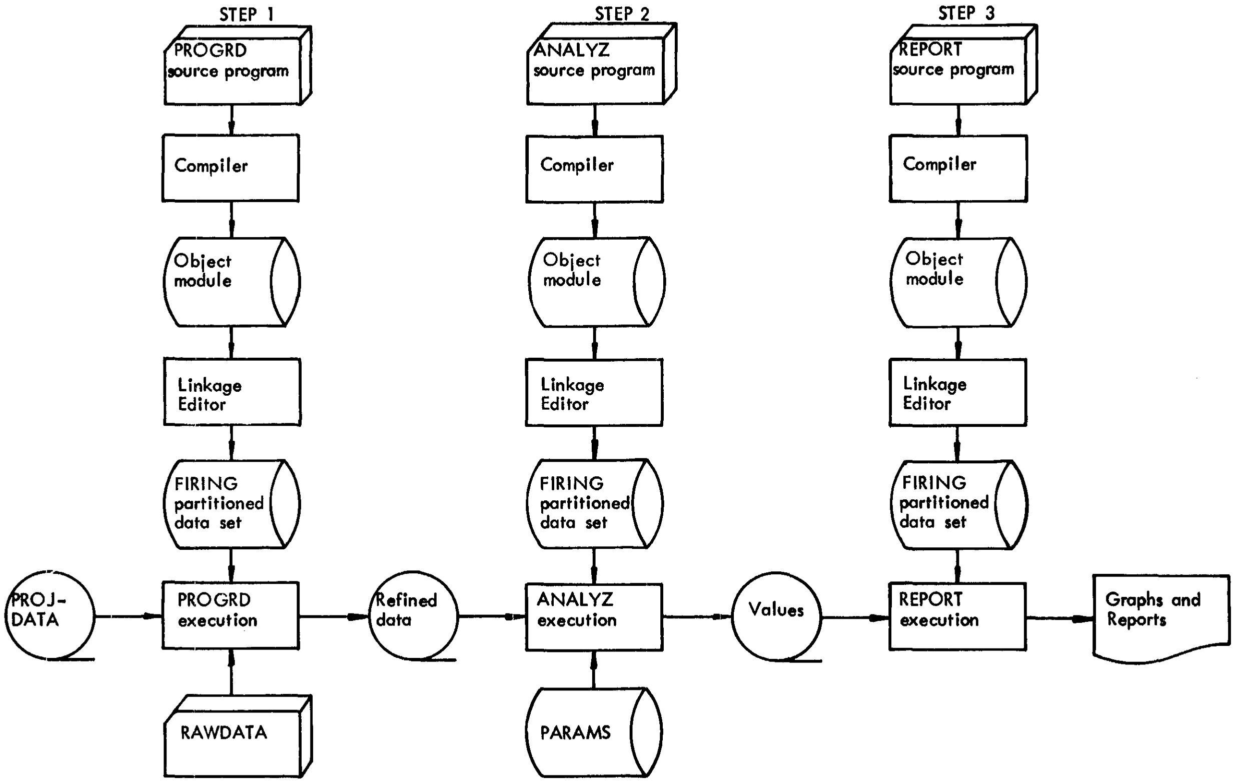
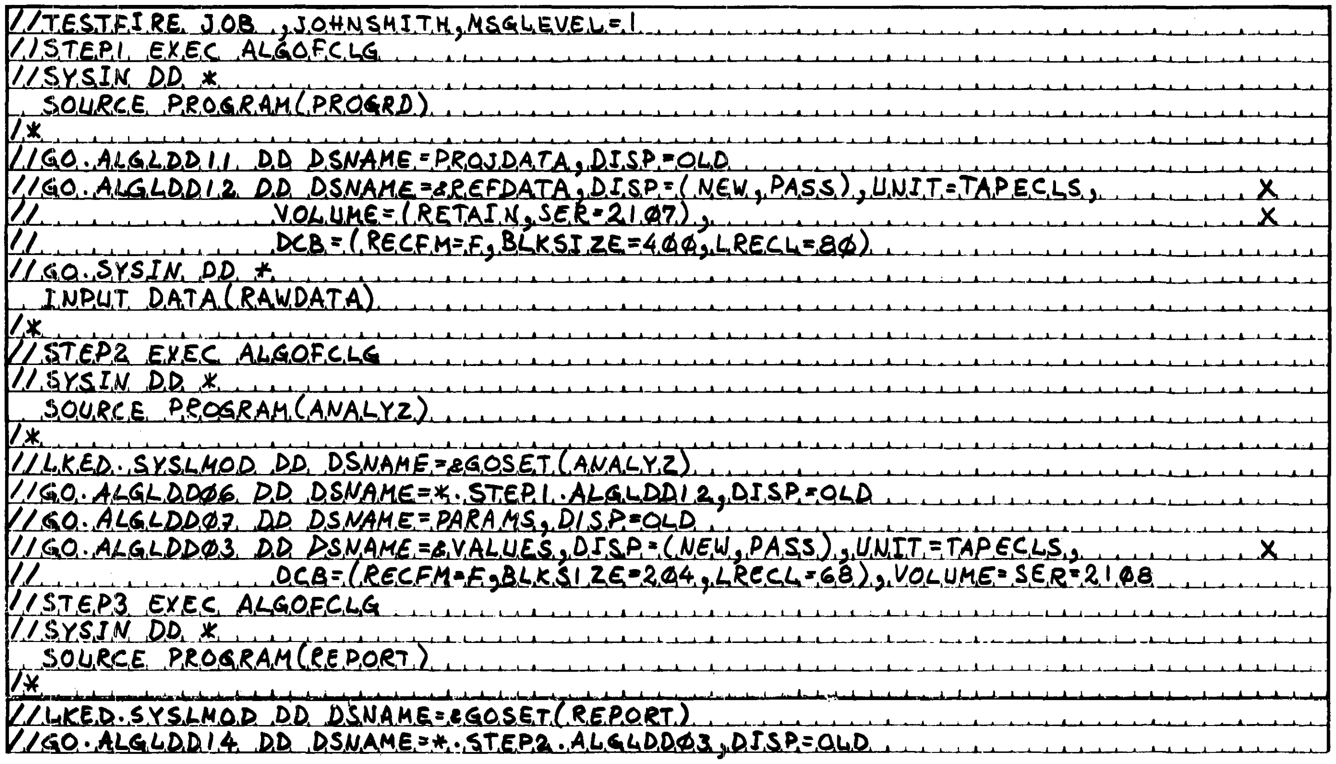


Figure 28. Basic I/O flow for Example 2. The data sets for information listings, ALGOL library routines intermediate work and the execution time error routine are not shown.



```
//TESTFIRE JOB ,JOHNSMITH,MSGLEVEL=1
//STEP1 EXEC ALGOFCLG
//SYSIN DD *
  SOURCE PROGRAM(PROGRD)
/*
//GO.ALGLDD11 DD DSNAME=PROJDATA,DISP=OLD
//GO.ALGLDD12 DD DSNAME=&REFDATA,DISP=(NEW,PASS),UNIT=TAPECLS,            X
//             VOLUME=(RETAIN,SER=2107),                                 X
//             DCB=(RECFM=F,BLKSIZE=400,LRECL=80)
//GO.SYSIN DD *
  INPUT DATA(RAWDATA)
/*
//STEP2 EXEC ALGOFCLG
//SYSIN DD *
  SOURCE PROGRAM(ANALYZ)
/*
//LKED.SYSLMOD DD DSNAME=&GOSET(ANALYZ)
//GO.ALGLDD06 DD DSNAME=*.STEP1.ALGLDD12,DISP=OLD
//GO.ALGLDD07 DD DSNAME=PARAMS,DISP=OLD
//GO.ALGLDD03 DD DSNAME=&VALUES,DISP=(NEW,PASS),UNIT=TAPECLS,             X
//             DCB=(RECFM=F,BLKSIZE=204,LRECL=68),VOLUME=SER=2108
//STEP3 EXEC ALGOFCLG
//SYSIN DD *
  SOURCE PROGRAM(REPORT)
/*
//LKED.SYSLMOD DD DSNAME=&GOSET(REPORT)
//GO.ALGLDD14 DD DSNAME=*.STEP2.ALGLDD03,DISP=OLD
```

Figure 29. Job control statements for Example 2.

occurs) printed on the normal system output device for information listings

2. The first job step invokes the ALGOFCLG cataloged procedure (see Appendix B) to process and execute the ALGOL source program (PROGRD) entered in the input stream

3. The other input data sets are RAWDATA and PROJDATA. RAWDATA is also entered in the input stream

4. The temporary output data set is

   - to be called REFDATA.TESTFIRE and to be passed for use in a later job step
   - to use the device class TAPECLS
   - to be written on volume 2107, which is to remain mounted for use later
   - to have fixed length records, 80 bytes long, and a block size of 400 bytes

5. The second job step invokes the ALGOFCLG cataloged procedure to process and execute the ALGOL source program (ANALYZ) entered in the input stream

6. The SYSLMOD DD statement in the LKED step of the cataloged procedure is overridden to specify that the load module produced by the linkage editor is

   - to be a new member, ANALYZ, of temporary partitioned data set GOSET.TESTFIRE

7. The other input data sets are REFDATA. TESTFIRE and PARAMS. Both will be kept at the end of the job step

8. The temporary output data set is

   - to be called VALUES.TESTFIRE and is to be passed for use in a later job step
   - to use the device class TAPECLS
   - to be written on volume 2108
   - to have fixed length records, 68 bytes long, and a maximum block size of 204 bytes

9. The third job step invokes the ALGOFCLG cataloged procedure to process and execute the ALGOL source program (REPORT) entered in the input stream. The output data will be listed on the printer specified in the cataloged procedure

10. The SYSLMOD DD statement in the LKED step of the cataloged procedure is over-ridden to specify that the load module produced by the linkage editor is

   - to be a new member, REPORT, of temporary partitioned data set GOSET.TESTFIRE

11. The other input data set is VALUES.TESTFIRE which will be kept at the end of the job step

Example 3: Executing Two Load Modules

Statement of problem: Data on current weather conditions is to be read from cards and used by the program FILECR to create a new generation of a data set WEATHER, and also to print a report.

Then the new generation and the three immediately preceding generations of the WEATHER data set are to be used by the program FORCST to produce a printed weather forecast. The pro-

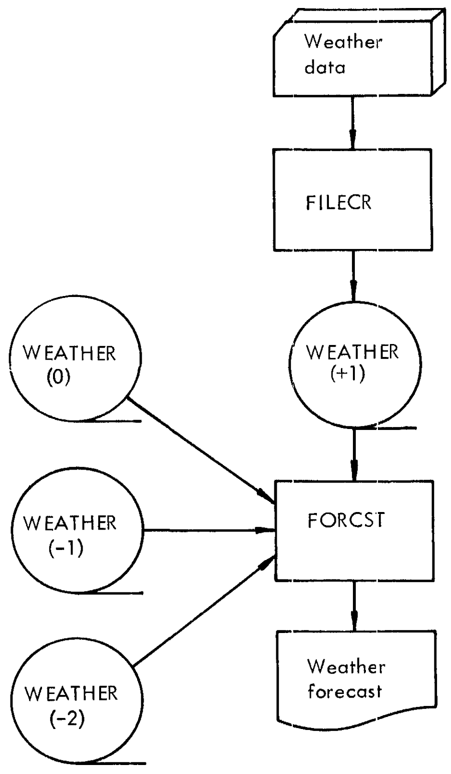


Figure 30. I/O flow for Example 3.

IEX007I W NNNNN LABEL BEGINNING WITH (up to six characters) CONTAINS INVALID CHARACTER. COLON DELETED.

Explanation: A label has been deleted because it contains a character of other than alphameric type.

IEX008I W NNNNN LABEL BEGINS WITH INVALID CHARACTER. COLON DELETED.

Explanation: A label has been deleted because it does not begin with an alphabetic character.

IEX010I S NNNNN SPECIFICATION PART OF PROCEDURE (identifier) INCOMPLETE.

Explanation: Not all of the formal parameters used in a procedure have been specified.

IEX011I S NNNNN PROGRAM STARTS WITH ILLEGAL DELIMITER.

Explanation: A program has been written not starting with one of the following:

1. 'BEGIN'
2. 'PROCEDURE'
3. 'REAL' 'PROCEDURE'
4. 'INTEGER' 'PROCEDURE'
5. 'BOOLEAN' 'PROCEDURE'

IEX012I W NNNNN TWO APOSTROPHES AFTER (six characters). FIRST APOSTROPHE DELETED.

Explanation: In this context, two apostrophes cannot be used together so one has been deleted.

IEX013I W NNNNN APOSTROPHE ASSUMED AFTER DELIMITER BEGINNING WITH (up to six characters).

Explanation: All delimiters involving words must begin and end with apostrophes. One has been left out of the program and has been inserted by the compiler.

IEX014I S NNNNN DELIMITER BEGINNING WITH (up to six characters) INVALID. FIRST APOSTROPHE DELETED.

Explanation: An invalid sequence of characters has been used after an apostrophe which apparently started a delimiter. The apostrophe is therefore deleted to remove the delimiter status from the characters but still include them in the program.

IEX015I W NNNNN MISSING SEMICOLON AFTER 'CODE'. SEMICOLON INSERTED.

Explanation: Self-explanatory.

IEX016I S NNNNN IDENTIFIER BEGINNING WITH (up to six characters) CONTAINS INVALID CHARACTER. IDENTIFIER DELETED.

Explanation: A character other than an alphameric type has been used in an identifier and so the identifier has been deleted.

IEX017I S NNNNN MORE THAN 65535 SEMICOLONS. SEMICOLON COUNTER RESET TO ZERO.

Explanation: Number of semicolons used exceeds capacity limitations. Duplicate numbers are allocated.

IEX018I W NNNNN DELIMITER 'COMMENT' IN ILLEGAL POSITION.

Explanation: 'COMMENT' has not been placed after a 'BEGIN' or a semicolon. Compilation continues normally.

IEX020I T NNNNN BLOCKS, COMPOUND STATEMENTS, FOR STATEMENTS, AND PROCEDURE DECLARATIONS NESTED TO TOO MANY LEVELS.

Explanation: Structure of program causes it to exceed capacity limitations (see Section 4).

IEX021I S NNNNN DECLARATOR (declarator) IN ILLEGAL POSITION.

Explanation: A declarator must come between either 'BEGIN' and the first statement of a block, or 'PROCEDURE' and the procedure body.

IEX022I T NNNNN MORE THAN 255 PROGRAM BLOCKS.

Explanation: Number of program blocks used exceeds capacity limitations.

IEX023I S NNNNN STRING POOL OVERFLOW.

Explanation: Total length of strings used exceeds capacity limitations (see Section 4).

IEX024I S NNNNN DELIMITER 'CODE' IN ILLEGAL POSITION. 'CODE' DELETED.

Explanation: 'CODE' has not been placed immediately after a procedure heading so it has been deleted.

IEX025I S NNNNN SPECIFIER 'STRING' OR 'LABEL' IN ILLEGAL POSITION. SPECIFICATION DELETED.

Explanation: 'STRING' and 'LABEL' have been used outside a procedure heading, so they have been deleted.

IEX026I S NNNNN PARAMETER (identifier) MULTIPLY SPECIFIED. FIRST SPECIFICATION USED.

Explanation: Self-explanatory.

IEX027I W NNNNN PARAMETER (identifier) MISSING FROM FORMAL PARAMETER LIST. SPECIFICATION IGNORED.

Explanation: A parameter has been specified in a procedure heading which does not exist in the formal parameter list, so it has been ignored.

IEX028I S NNNNN DELIMITER 'VALUE' IN ILLEGAL POSITION. VALUE PART DELETED.

Explanation: 'VALUE' has been placed outside a procedure heading so the value part has been deleted.

IEX029I W NNNNN SPECIFICATION PART PRECEDES VALUE PART.

Explanation: The specification part in a procedure heading has been incorrectly placed before the value part.

IEX030I W NNNNN PARAMETER (identifier) REPEATED IN VALUE PART.

Explanation: A parameter has been included in the value part of a procedure heading more than once.

IEX031I W NNNNN LEFT PARENTHESIS NOT FOLLOWED BY / AFTER ARRAY IDENTIFIER (identifier). SUBSCRIPT BRACKET ASSUMED.

Explanation: The subscript bounds after an array identifier have been preceded by a left parenthesis instead of a subscript bracket.

IEX032I S NNNNN MISSING RIGHT PARENTHESIS IN BOUND PAIR LIST OF ARRAY (identifier). DECLARATION DELETED.

Explanation: A right parenthesis has been omitted in the list of subscript bounds for an array identifier, so the declaration is deleted.

IEX033I T NNNNN MORE THAN 16 DIMENSIONS OR COMPONENTS IN DECLARATION OF (identifier).

Explanation: The number of dimensions or components used with an array or switch identifier exceeds the maximum allowed.

IEX034I S NNNNN ARRAY SEGMENT (identifier) NOT FOLLOWED BY SEMICOLON OR COMMA. CHARACTERS TO NEXT SEMICOLON DELETED.

Explanation: An array segment must be followed by a semicolon if it is the only or last segment of an array declaration; or a comma if it is followed by another segment.

IEX035I W NNNNN ILLEGAL PERIOD IN ARRAY OR SWITCH LIST. PERIOD DELETED.

Explanation: A period has been used wrongly in an array or switch list and deleted from the program. A period can be used only as a decimal point, or as part of a colon or semicolon.

IEX036I T NNNNN MORE THAN 15 PARAMETERS IN DECLARATION OF (identifier).

Explanation: The number of formal parameters specified for a procedure exceeds the maximum allowed.

IEX206I W NNNNN TOO MANY OPTION PARAMETER ERRORS. SUBSEQUENT PARAMETERS IGNORED.

Explanation: Too many incorrect parameters have been specified in the PARM parameter so the rest are ignored.

IEX207I W NNNNN POSSIBLE ERROR IN DD NAMES PARAMETER.

Explanation: An incorrect ddname may have been specified in the DD statement.

IEX208I W NNNNN SIZE PARAMETER INVALID. SIZE 45056 ASSUMED.

Explanation: The main storage size specified as being available to the compiler is less than the minimum required, so the minimum value is assumed.

IEX209I T NNNNN COMPILATION UNSUCCESSFUL DUE TO PROGRAM INTERRUPT. PSW (hexadecimal digits).

Explanation: A program interrupt has occurred causing termination of the job step. The program status word when the error occurred is given.

IEX210I T NNNNN UNRECOVERABLE I/O ERROR ON DATA SET (ddname).

Explanation: An I/O error has occurred on the data set specified causing termination of the job. This message is typed on the console typewriter when it concerns SYSPRINT. This is most likely to be a random error, so the user is recommended to rerun the program.

IEX211I T NNNNN PROGRAM INTERRUPT IN ERROR MESSAGE EDITING ROUTINE. PSW (hexadecimal digits).

Explanation: A program interrupt has occurred in the error message editing routine, ending the job.

IEX212I T NNNNN TOO MANY ERRORS.

Explanation: The total length of the error message patterns produced exceeds capacity limitations.

IEX213I T NNNNN INTERNAL OVERFLOW OF IDENTIFIER TABLE.

Explanation: The number of identifiers declared exceeds capacity limitations.

IEX214I S NNNNN DATA STORAGE AREA EXCEEDED. PROGRAM BLOCK NO. (number).

Explanation: The data storage area required by the program block specified exceeds 4096 bytes.

IEX215I T NNNNN SOURCE PROGRAM TOO LONG.

Explanation: The source program exceeds capacity limitations (see Section 4).

IEX216I S NNNNN TOO MANY LABELS. LABEL NUMBER RESET.

Explanation: The total number of labels used exceeds capacity limitations, so duplicated numbers are allocated (see Section 4).

LINKAGE EDITOR MESSAGES

Each message occupies one or more printed lines and contains:

- The message key, consisting of the letters IEW, a three digit decimal number identifying the message, and a final digit, either 1, 2, 3 or 4, indicating the severity code.
- The message text describing the error. For severity code 1 the message is preceded by 'WARNING'. For all other severity codes the message is preceded by 'ERROR'.

The severity codes have the following meaning:

1 indicates a condition that may cause an error during execution of the load module. A module map or cross-reference table is produced if it was required by the programmer. The output load module is marked as executable.

2 indicates an error that could make execution of the load module impossible. Processing continues. When possible, a module map or cross-reference table is produced if it was required. The load module is marked as not executable unless the LET option has been specified.

3 indicates an error that will make execution of the load module impossible. Processing continues. If possible a module map or cross-reference table is produced if it was required. The load module is marked as not executable.

4 indicates an error condition from which no recovery is possible. Processing terminates. The only output is diagnostic messages.

A full list of the linkage editor diagnostic messages is contained in IBM System/360 Operating System: Linkage Editor.

EXECUTION TIME MESSAGES

The list of diagnostic messages that may be produced by the load module is given below. Each message occupies one or more printed lines and contains:

- The message key, consisting of the letters IHI, a three digit decimal number identifying the message, and the letter I to indicate an informative message requiring no action from the operator.
- The characters SC = followed by the semicolon number (see Section 3). This number does not always indicate the statement in which the error occurred. For example, after a branch ('GOTO' or 'FOR'), if no semicolon has occurred before the error is detected, then the semicolon number preceding the branching instruction will be listed. For I/O errors, the semicolon number indicates the statement being executed when the error was detected, not the statement calling the I/O procedure.
- The message text describing the error. Where appropriate this begins by indicating the number of the data set (DSN) on which the error occurred, or the ddname if the data set does not have a number (that is, SYSUT1 and SYSUT2), or the program status word (PSW) held by the operating system when the error occurred. The PSW contains 16 hexadecimal digits. Message texts preceded by ** indicate that the program does not correspond with parameters specified in the job control cards.

IHI000I SC=NNNNN DATA SET NUMBER OUT OF RANGE

Explanation: A data set number must be in the range 0 to 15.

IHI001I SC=NNNNN DSN=NN. REAL NUMBER TO BE CONVERTED OUT OF INTEGER RANGE

Explanation: A real number has been included which exceeds capacity limitations when converted to integer. This message applies for input/output operations.

IHI002I SC=NNNNN DSN=NN. INCOMPATIBLE ACTIONS ON DATA SET

Explanation: The I/O procedure requested is not defined for this data set. For example, procedure SYSACT8 specifying data set number 0 is not allowed.

IHI003I SC=NNNNN DSN=NN. INPUT BEYOND LAST OUTPUT

Explanation: Before reading data which has just been written on the same data set, backward repositioning must be specified.

IHI004I SC=NNNNN TOO MANY REPOSITIONINGS IN DATA SETS. INTERNAL OVERFLOW

Explanation: Too many repositionings have caused an internal overflow of the Note Table (see Section 4).

IHI005I SC=NNNNN DSN=NN. INPUT REQUEST BEYOND END OF DATA SET

Explanation: Input has been requested to start beyond the end of the data set.

IHI006I SC=NNNNN DSN=NN. EXPONENT PART OF INPUT NUMBER CONSISTS OF MORE THAN TWO SIGNIFICANT DIGITS

Explanation: The length of the exponent part of an input number exceeds capacity limitations.

IHI007I SC=NNNNN DSN=NN. **NO CONTROL CHARACTER SPECIFIED IN RECORD FORMAT OF DATA SET. SPLITTING INTO SECTIONS IMPOSSIBLE

Explanation: A control character is required to define printing format.

IHI008I SC=NNNNN DSN=NN. SOURCE IN PROCEDURE OUTSYMBOL DOES NOT MATCH STRING

IBM Technical Newsletter

File Number	S360-26
Re: Form No.	C33-4000-0
This Newsletter No.	N33-8002
Date	January 16, 1968
Previous Newsletter Nos.	N33-8000

IBM System/360 Operating System
ALGOL Programmer's Guide

This Technical Newsletter relates to Release 15 and contains amendments to the IBM System/360 Operating System: ALGOL Programmer's Guide, Form C33-4000-0. The attached pages are replacements to be inserted in the publication, as indicated below. Corrections and additions to the text and/or illustrations are indicated by a vertical bar to the left of the affected text and by a bullet (•) to the left of the figure caption.

Pages to be Inserted	Pages to be Removed
11-12	11-12
13-14	13-14
15-16	15-16
17-18	17-18
25-26	25-26
31-32	31-32
33-34	33-34
35-36	35-36
45-46	45-46
49-50	49-50
51-52	51-52

Summary of Amendments

The amendments in this Newsletter primarily reflect the division of the execution time data set SYSPRINT into two data sets, namely ALGLDD01 and SYSPRINT, in order to provide MFT-II support for ALGOL under Release 15. This modification necessitates changes in certain cataloged procedures. The amendments also include various corrections and program maintenance adjustments.

Note: Please file this cover letter at the back of the publication, for use as a reference list.

IBM Nordic Laboratory, Technical Communications, Box 962, Lidingö 9, Sweden

PRINTED IN U.S.A.

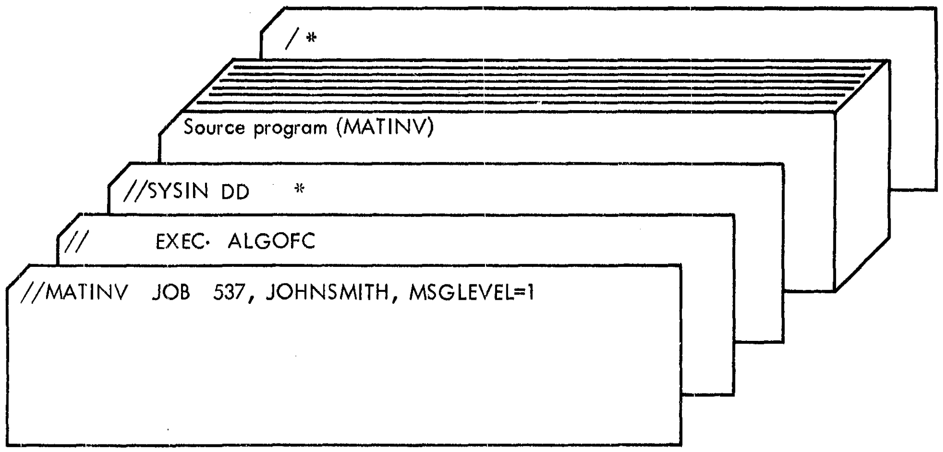


Figure 2. Sample deck for using ALGOFC cataloged procedure with a single source program. This job compiles the MATINV source program used in Example 1 of Appendix E.

If more than one source program is to be processed in the same job, then all job control statements except the JOB statement must be repeated for each source program.

If it is required to keep a load module for use in a later job (as in the case when the load module is a precompiled procedure), then the SYSLMOD DD statement in the cataloged procedure must be over-ridden to specify a permanent data set. This has to be done for each load module that is kept. The over-riding statement is placed at the end of the job step to which it applies, and has the form:

```
//LKED.SYSLMOD  DD  DSNAME=dsname(member),
                    DISP=(MOD,KEEP)
```

where "dsname" is the name of a partitioned data set and "member" is the member name assigned to the load module on the partitioned data set.

A sample deck of job control statements to compile and linkage edit two source programs is shown in Figure 3.

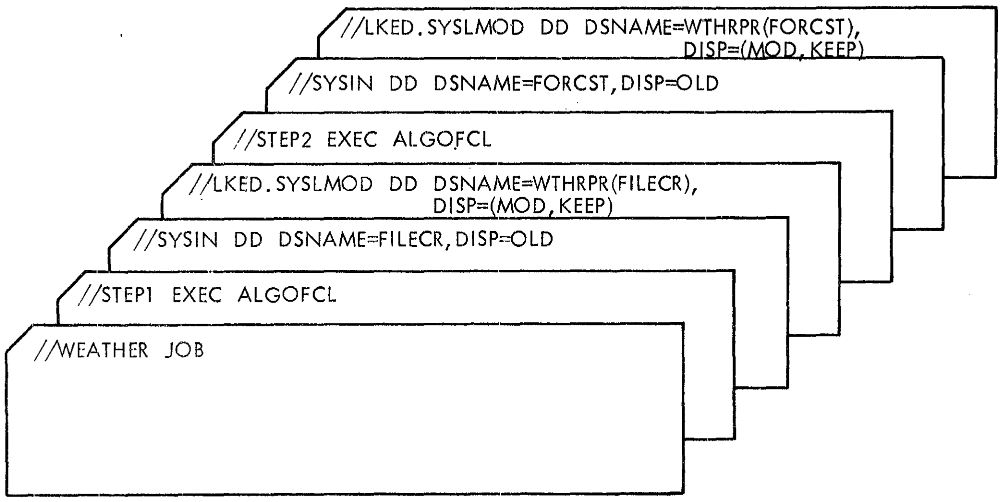

//LKED.SYSLMOD DD DSNAME=WTHRPR(FORCST),
DISP=(MOD, KEEP)
//SYSIN DD DSNAME=FORCST,DISP=OLD
//STEP2 EXEC ALGOFCL
//LKED.SYSLMOD DD DSNAME=WTHRPR(FILECR),
DISP=(MOD, KEEP)
//SYSIN DD DSNAME=FILECR,DISP=OLD
//STEP1 EXEC ALGOFCL
//WEATHER JOB

Figure 3. Sample deck for using ALGOFCL cataloged procedure with two source programs. These two job steps compile and linkage edit the two source programs used in Example 3 of Appendix E. Both source programs have been previously stored on intermediate I/O devices.

Compilation, Linkage Editing and Execution

The cataloged procedure used to compile an ALGOL source program, linkage edit the resulting object module, and execute the load module produced by the linkage editor is ALGOFCLG.

The statements used in this cataloged procedure are shown in Appendix B. The following statements can be used to invoke the ALGOFCLG cataloged procedure:

```
//jobname    JOB
//JOBLIB     DD  DSNAME=dsname1, DISP=OLD
//           EXEC  ALGOFCLG
//SYSIN      DD   {* or parameters defining an
                     input data set containing
                     the source program }
//GO.ALGLDD02  DD  DSNAME=dsname2
                                       .
                                       .
                                       .
//GO.ALGLDD15  DD  DSNAME=dsname15
```

where "jobname" is the name assigned to the job. "dsname1" is the name of a data set that contains a precompiled procedure (see Section 4) which is called by the load module being executed. The DD statement containing dsname1 need not be used if no precompiled procedure is used.

For a description of the correct use of the JOBLIB DD statement when more than one precompiled procedure is used in a job, or when a precompiled procedure resides on more than one data set, see "Data Set Concatenation" in Appendix E.

"dsname2"... "dsname15" are the names of input data sets required by the load module at execution time and output data sets to be created at execution time. In addition, two data sets for printed output (ddnames SYSPRINT and ALGLDD01) are supplied by the cataloged procedure, and a data set for input only can be specified by using the following statement after the invoking sequence just given.

```
//GO.SYSIN  DD  {* or parameters defining an
                   input data set }
```

If DD* is used then the data must follow immediately afterwards in the input stream. For sequential scheduling, the data must be followed by a delimiter statement (/*).

If more than one source program is to be processed and executed in the same job, then all job control statements except the JOB statement and the JOBLIB DD statement must be repeated for each source program.

A sample deck of job control statements required to compile, linkage edit and execute three source programs is shown in Figure 29.

Over-riding Cataloged Procedures

The programmer can change any of the statements in a cataloged procedure, except the name of the program in an EXEC statement.

These over-riding conditions are temporary, and will be in effect only until the next job step is started. The following text describes methods of temporarily modifying existing parameters and adding new parameters to the EXEC and DD statements used in the cataloged procedures. The full list of parameters available to the ALGOL programmer for these statements, and detailed explanations of the parameters, is given in Appendix E. The EXEC and DD statements used in the IBM-supplied cataloged procedures are shown in Appendix B.

Over-riding EXEC Statements

In the EXEC statement, the programmer can change or add any of the keyword parameters by using the following format:

```
keyword.procstep=option
```

where:

"keyword" is the parameter to be changed in, or added to, the specified procedure job step: either COND, PARM, ACCT, TIME or REGION. TIME and REGION are valid only for priority scheduling.

"procstep" is the procedure job step in which the change or addition is to occur: either ALGOL, LKED or GO.

"option" is the new option required.

For example, if the EXEC statement used to invoke the ALGOFCLG cataloged procedure was written as:

```
// EXEC  ALGOFCLG,PARM.ALGOL=DECK,
//                 PARM.LKED=XREF,
//                 COND.GO=(3,LT,ALGOL)
```

then the following changes would be made to the ALGOFCLG cataloged procedure:

1. In the PARM parameter of the job step ALGOL, the option DECK would be used instead of the default option NODECK (assuming that the standard default NODECK was not changed at system generation). Over-riding this option will not affect the other default options assumed for this parameter.

2. In the job step LKED, the option XREF is specified for the PARM parameter. Since the options specified in the cataloged procedure were XREF, LIST and LET, this statement has the effect of deleting the options LIST and LET since they were not default options.

3. In the job step GO, the COND parameter code is changed from 5, as it appears in the cataloged procedure, to 3. In this example, the code 3 causes the job step GO to be bypassed if a warning message is generated during the job step ALGOL. Note that although the other options (LT and ALGOL) are not to be altered, the entire parameter being modified must be respecified.

If "procstep" is not specified when overriding a multi-step cataloged procedure, the operating system makes the following assumptions:

- COND, ACCT and REGION parameters apply to all procedure job steps.
- A PARM parameter applies to the first procedure job step and any options already specified in the PARM parameters for the remaining procedure job steps are cancelled.
- A TIME parameter specifies the computing time for the entire job and any options already specified in the TIME parameters for individual procedure job steps are cancelled.

Over-riding DD Statements

An additional DD statement is used in the invoking sequence for each DD statement in the cataloged procedure that is to be over-ridden. The following format is used:

```
//procstep.ddname DD parameter-list
```

where:

"procstep" is the procedure job step containing the DD statement to be over-ridden: either ALGOL, LKED or GO. If "procstep" is omitted then the first procedure job step is assumed.

"ddname" is the name of the DD statement to be over-ridden.

"parameter-list" is the list of parameters that are being added or changed. In both cases the whole parameter must be specified. Unchanged parameters in the original statement need not be specified. For example, the statement:

```
//ALGOL.SYSLIN DD SPACE=(400,(80,10))
```

will change the SPACE parameter of the SYSLIN DD statement in the ALGOL job step so that space will be allocated for 80 physical records instead of 40.

DD statements that are used to over-ride other DD statements in the cataloged procedures must be placed immediately after the EXEC statement invoking the cataloged procedure, and must be in the same order as their corresponding DD statements in the cataloged procedures.

Adding DD Statements

Complete, new DD statements that are to be added to the cataloged procedure use the same format as over-riding DD statements. The "ddname" specified must not exist in the job step specified by "procstep". These new DD statements must follow immediately after the over-riding DD statements which apply to the same procedure job step.

USER-WRITTEN PROCEDURES

The information required by the programmer to write his own job control procedures is given in the following text, and in Appendix E. Cataloging user-written procedures, or permanently modifying the IBM-supplied cataloged procedures, is accomplished using the IEBUPDTE utility program, described in IBM System/360 Operating System: Utilities. The statements required in user-written procedures are:

- An EXEC statement to invoke the program.
- DD statements to define the data sets used by the program.

Compilation

Invoking Statement

The ALGOL compiler consists of ten load modules contained in the link library, SYS1. LINKLIB, of the operating system. The compiler is activated

by invoking its first load module, named ALGOL, which then internally invokes the other load modules of the compiler.

The usual method of invoking the compiler is by means of an EXEC statement of the form:

```
//stepname  EXEC  PGM=ALGOL
```

where "stepname" is the name assigned to the job step (optional).

Other EXEC statement parameters may be included if required (see Appendix E).

(A method of dynamically invoking the compiler within a job step, by means of the CALL, LINK, XCTL or ATTACH macro-instructions, is described in Section 4.)

Data Sets Used

The data sets used in the compilation process are illustrated in Figure 4, and described in Figure 5. These data sets must be specified by the programmer with suitable DD statements.

Blocksize DCB information may be specified by the user for SYSIN, SYSLIN, SYSPRINT and SYSPUNCH. The maximum blocking factor depends on the main storage size available (see Figure 6). Record length is fixed at 80 bytes for SYSIN, SYSLIN and SYSPUNCH, and 91 bytes for SYSPRINT.

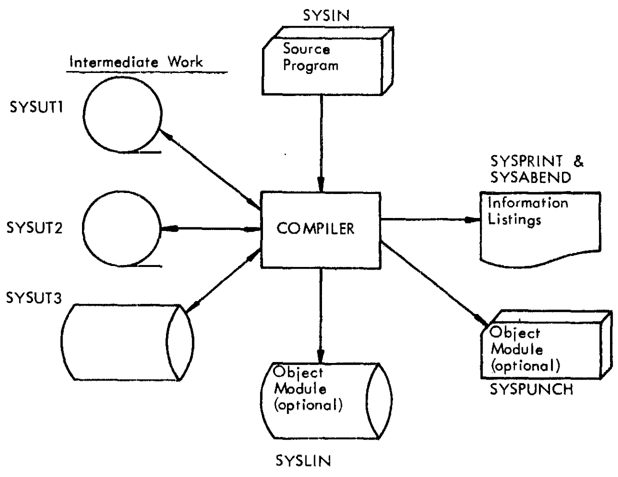


Figure 4. Flowchart showing data sets used by the compiler.

The space required for the compiler data sets depends on the size and structure of the source program, however it can be assumed that only in rare cases will the object module exceed four times the source program and usually much less will be required.

Purpose	Standard ddname	Devices required
For ALGOL source program	SYSIN	Card reader*
For object module to be used by linkage editor	SYSLIN	Direct access or magnetic tape
For compilation listings	SYSPRINT	Printer*
For object module (copied from SYSLIN)	SYSPUNCH	Card punch*
For the control program dump	SYSABEND	Printer*
For intermediate compiler working	SYSUT1	Direct access or magnetic tape
For intermediate compiler working	SYSUT2	Direct access or magnetic tape
For intermediate compiler working	SYSUT3	Direct access

* Some form of intermediate storage, such as magnetic tape, may be used to reduce I/O delay for the central processing unit.

Figure 5. Data sets used by the ALGOL compiler.

Also, as a rough estimate, SYSUT1, 2 and 3 must each be large enough to contain the number of valid characters in the source program.

SYSABEND is used for control program listings (see Section 3).

Processing of all data sets by the compiler is independent of the I/O device used except for the intermediate work data sets. These require magnetic tape or direct access devices.

Linkage Editing

Invoking Statement

The linkage editor is usually invoked with an EXEC statement of the form:

```
//stepname  EXEC  PGM=IEWL
```

where "stepname" is the name assigned to the job step (optional).

Other EXEC statement parameters may be included if required (see Appendix E). IEWL specifies the highest-level linkage editor in the installation's operating system.

(A method of dynamically invoking the linkage editor within a job step, by means of the CALL, LINK, XCTL or ATTACH instructions, is described in Section 4.)

Main storage size in bytes at which changes occur	Maximum blocking factor			
	SYSIN	SYSPRINT	SYSLIN	SYSPUNCH
45056 (44K)	5	5	5	1
51200 (50K)	5	5	5	5
59392 (58K)	5	5	5	5
67584 (66K)	5	5	5	5
77824 (76K)	5	5	5	5
90112 (88K)	20	20	40	20
104448 (102K)	20	20	40	20
120832 (118K)	20	20	40	20
139264 (136K)	20	20	40	20
159744 (156K)	20	20	40	20
184320 (180K)	40	40	40	40
212992 (208K)	40	40	40	40

Figure 6. Effect on compiler data sets if more than 44K bytes of main storage is available. The capacity of internal tables in the compiler is increased at each of the main storage sizes listed in this table, allowing, for example, a larger number of identifiers to be included in the source program. Therefore to get optimum performance, the user is recommended to use this list when specifying main storage size available to the compiler.

Data Sets Used

The data sets used by the linkage editor (see Figures 7 and 8) must be defined by the programmer with suitable DD statements.

Blocksize DCB information may be specified by the user for SYSLIN and SYSPRINT if the F level linkage editor is being used. Maximum blocking factor is 5 when 44K bytes of main storage size is available, and 40 when 88K bytes is available. Record length is fixed at 80 bytes for SYSLIN and 120 bytes for SYSPRINT.

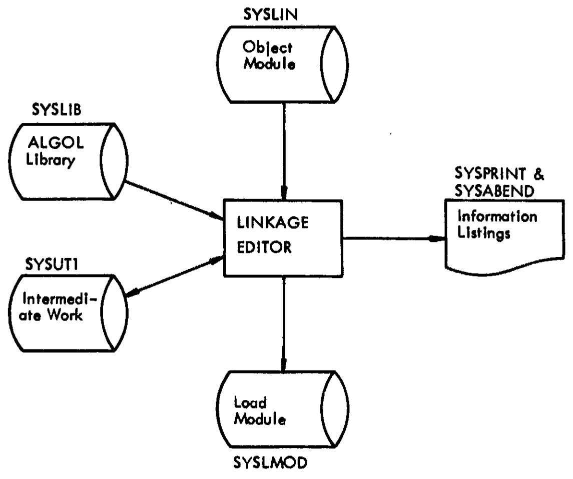


Figure 7. Flowchart showing data sets used by the linkage editor.

SYSABEND is used for control program listings (see Section 3).

Load Module Execution

Invoking Statement

The usual method of invoking the load module generated by the linkage editor is with an EXEC statement of the form:

```
//stepname EXEC PGM=member-name
```

Purpose	Standard ddname	Devices used
For object module input	SYSLIN	Direct access or magnetic tape
For load module output, stored as a member of a partitioned data set	SYSLMOD	Direct access
For ALGOL library, SYS1.ALGLIB. A partitioned data set containing routines in load module form	SYSLIB	Direct access
For linkage editing listings	SYSPRINT	Printer*
For intermediate linkage editor working	SYSUT1	Direct access or magnetic tape
For the control program dump	SYSABEND	Printer*

* Some form of intermediate storage, such as magnetic tape, may be used to reduce output delay for the central processing unit.

Figure 8. Data sets used by the linkage editor.

where "stepname" is the name assigned to the job step (optional).

"member-name" indicates the name of the partitioned data set member which contains the load module. This name is specified by the programmer in the SYSLMOD DD statement for the linkage editor. Other EXEC statement parameters may be included if required (see Appendix E).

(A method of dynamically invoking the load module within a job step, by means of the CALL, LINK, XCTL or ATTACH macro-instructions is described in Section 4.)

Data Sets Used

Up to 16 data sets for use at execution time may be specified by the programmer in the ALGOL source program by using the appropriate data set number. The numbers used and the corresponding names of their DD statements are listed below.

Data set number used in ALGOL source program	Corresponding ddname
0	SYSIN
1	ALGLDD01
2	ALGLDD02
3	ALGLDD03
4	ALGLDD04
5	ALGLDD05
6	ALGLDD06
7	ALGLDD07
8	ALGLDD08
9	ALGLDD09
10	ALGLDD10
11	ALGLDD11
12	ALGLDD12
13	ALGLDD13
14	ALGLDD14
15	ALGLDD15

Any reference to a data set number by an I/O procedure within an ALGOL source program is translated into a reference to a data control block using the corresponding ddname. It is the responsibility of the programmer to supply the DD statements which correspond to the data set numbers used in the ALGOL source program.

The execution time data sets are illustrated in Figure 9 and described in Figure 10. For ALGLDD02 to ALGLDD15, case 1 in the column showing device used, applies if the source program contains any of the following:

- A backward repositioning specification by the procedures SYSACT4 or SYSACT13 for this data set.
- Both input and output procedure statements for this data set.
- Procedure statements which prevent the compiler from recognizing whether either of these applies; for example, if the data set number or SYSACT function number is not an integer constant or if a precompiled procedure is used.

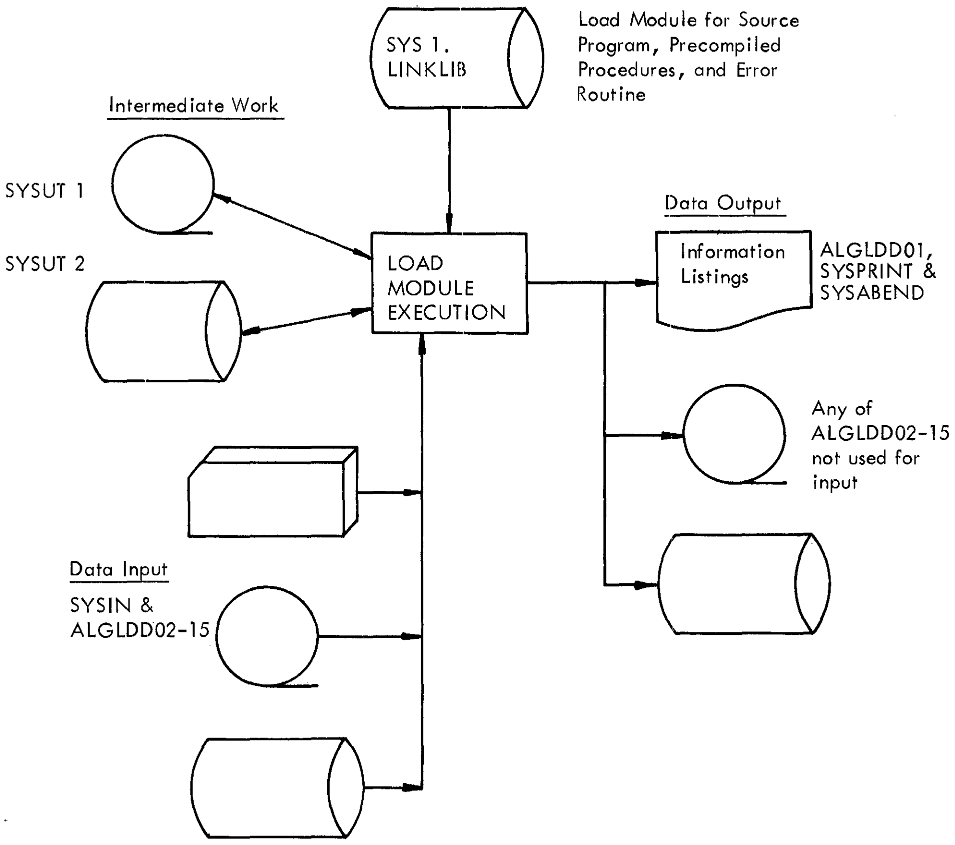
If the source program has already been compiled and linkage edited in a previous job, then the data set on which it has been stored (in load module form) must be concatenated to SYS1. LINKLIB. Data sets containing precompiled procedures called by the source program (see Section 4) must also be concatenated to SYS1. LINKLIB.

If the programmer specifies a TRACE, TRBEG or TREND option in the EXEC statement of the execution job step, the semicolon count (see Section 3) is stored intermediately on a data set with the ddname SYSUT1. The programmer must supply a corresponding DD statement if he uses this option. The semicolon count is converted to external form and transferred to the SYSPRINT data set as soon as the execution ends either by reaching the logical end of the source program or due to an error.

The space required for the semicolon count is:

For the main heading	6 bytes
For each semicolon	2 bytes
For each call of a precompiled procedure	12 bytes
For each physical record on SYSUT1	4 - 6 bytes

System/360 ALGOL permits data to be temporarily stored on and retrieved from external devices without conversion, using the ALGOL I/O procedures PUT and GET. If the programmer uses this facility in his source program, then he must supply a DD statement with the ddname SYSUT2. The device specified by this statement for storing such intermediate data should be a direct access device to guarantee reasonable performance, though programming is performed independently between magnetic tape and direct access devices. All data passed by a single PUT is



SYS 1. LINKLIB
Load Module for Source Program, Precompiled Procedures, and Error Routine
Intermediate Work
SYSUT 1
SYSUT 2
LOAD MODULE EXECUTION
Data Output
Information Listings
ALGLDD01, SYSPRINT & SYSABEND
Any of ALGLDD02-15 not used for input
Data Input
SYSIN & ALGLDD02-15

•Figure 9. Flowchart showing data sets used at load module execution. The data input and output requirements are variable.

stored as one record. This record will be as long as the data passed, plus 8 bytes. The maximum record length accepted is 2048 bytes.

The DCB information which may be specified by the user for execution time data sets is blocksize, record format and record length (see page 44 for details), except for the trace and PUT/GET data sets (ddnames SYSUT1 and SYSUT2) for which only blocksize may be specified (up to a maximum of 2048 bytes).

For information not provided, default values will be inserted by a routine in the ALGOL library. In particular, blocksize is assumed as 2048 bytes for SYSUT1 and SYSUT2 if none is specified.

SYSABEND is used for control program listings (see Section 3).

	Standard ddname	Device Used
For data input to load module	SYSIN	Any input device
For execution time listings	SYSPRINT	Printer*
For data output	ALGLDD01	Printer*
For data input or output	ALGLDD02 : ALGLDD15	1. Direct access or magnetic tape 2. Any
For intermediate storage of semicolon counter when TRACE is specified	SYSUT1	Direct access or magnetic tape
For temporary storage when PUT is specified	SYSUT2	Direct access or magnetic tape
For the control program dump	SYSABEND	Printer*

* Some form of intermediate storage, such as magnetic tape, may be used to reduce I/O delay for the central processing unit.

•Figure 10. Data sets used at execution time.

SECTION 3: INFORMATION LISTINGS

To assist the programmer to find the cause of any faults in the processing or execution of his program, various forms of information listings are produced for the compilation, linkage editing and execution operations. Some of these listings are optional. Examples are illustrated in Figures 11 to 16.

CONTROL PROGRAM LISTINGS

All three operations may produce listings generated by the control program. These are described in IBM System/360 Operating System: Messages, Completion Codes, and Storage Dumps. The ABEND macro-instruction for specifying the main storage dump is described in IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions.

COMPILATION LISTINGS

A successful compilation of an ALGOL source program produces the following information listings:

- Job control statement information according to which MSGLEVEL option was specified in the JOB statement.
- The source program supplemented by a count of the semicolons occurring in the program (optional).
- A table giving details of all identifiers used in the program (optional).
- Any warning diagnostic messages.
- Information on main storage requirements at execution time.

If a serious diagnostic message is produced (meaning that object module generation has ended) then the source program and identifier table listings will be printed in full if they have been requested, but the information on main storage requirements will not be printed. If a terminating diagnostic message is produced then the source program and identifier table listings can be printed only as far as they have been produced.

Source Program

If the SOURCE option has been specified, the source program is transferred by the compiler to an output data set in order to be listed by a printer. This source program is supplemented by a semicolon count, which is referred to in the diagnostic messages to help localize errors.

The compiler generates this semicolon count when scanning the source program, by counting all semicolons occurring in the source program outside strings, except those following the delimiter 'COMMENT'. The value of this semicolon count at the beginning of each record of the source program is printed at the left of that record. It is assigned by the compiler in order to have a clear, problem-oriented reference. Any reference to a particular semicolon number refers to the segment of source program following the specified semicolon, for example, the semicolon number 5 refers to the program segment between the fifth and sixth semicolons.

Identifier Table

If the SOURCE option has been specified, a list of all identifiers declared or specified within the source program is transferred by the compiler to the output data set for printing after the source program listing. This identifier table gives information about the characteristics and internal representation of all identifiers. The identifiers are grouped together within the identifier table according to their scopes.

All blocks and procedure declarations within the source program are numbered according to the order of occurrence of their opening delimiters 'BEGIN' or 'PROCEDURE'. Therefore, if the body of a procedure declaration is a block, then usually this block has the same number as the procedure declaration itself. These numbers are called program block numbers (even if they belong to a procedure declaration and not to a block).

Each line in the table contains entries for up to three identifiers and the line begins with the number of the program block in which the identifiers were declared or specified, the value of the semicolon count at the commencement of the program block, and the number of the immediately surrounding program block. Each identifier entry contains:

CAPACITY LIMITATIONS

In addition to those given in IBM System/360 Operating System: ALGOL Language, the following restrictions must be observed when writing an ALGOL source program:

Number of blocks and procedure declarations (NPB)	≤255
Number of for statements	≤255
Number of identifiers declared or specified in one block or procedure. F is at most twice the number of for statements occurring in that block	≤179-F for type procedures ≤180-F otherwise
Length of letter string serving as parameter delimiter	≤1024 letters when main storage size available is less than 50K, ≤2000 letters otherwise
Length of label identifer	≤1024 characters when main storage size available is less than 50K, ≤2000 characters otherwise
Number of valid characters	≤255K
Number of semicolons in the whole program	≤65535
Number of nested blocks, compound statements, for statements and procedure declarations	≤999
Number of labels declared or additionally generated by the compiler	≤1024

The compiler generates the following additional labels:

For each switch declaration	2
For each procedure declaration	2
For each procedure activation (including function designators)	1
For each 'THEN' and each 'ELSE'	1
For each for statement	at most L + 3 where L is the number of for list elements

Length of constant pool	≤(256 - NPB) x 4096 bytes

The requirements of components within the pool are

Integer constant	4 bytes
Real constant (SHORT)	4 bytes
Real constant (LONG)	8 bytes
String (in bytes)	2 + number of symbols of open string between the outermost string quotes

The constant pool is divided into blocks of 4096 bytes each. The first block contains the integer constants 0 to 15 (64 bytes). All strings together are restricted to fill not more than the rest of this block (4096 - 64 - 2S bytes, where S = number of strings).

No constant occurring more than once in the source program is stored twice in the same block; however, it may possibly be stored more than once in different blocks. Up to seven bytes may be left unused.

Length of data storage area for each block or procedure declaration	≤4096 bytes
Number of blank spaces serving as delimiters on I/O data sets	≤255

Number of records per section	≤255
Number of entries in the Note Table	≤127

(The Note Table stores information to retrieve records which may be required again later. An entry for a record is made each time the ALGOL I/O procedures PUT and SYSACT13 are executed, and each time an input operation, with backward repositioning, follows an output operation on the same data set.)

Identification number (N) used by PUT or GET	0≤N≤65535

INVOKING A PROGRAM WITHIN A JOB STEP

Any one of the four macro-instructions, CALL, LINK, XCTL or ATTACH, may be used to dynamically invoke the compiler, linkage editor and load module within a job step. This is an alternative to the more usual method of invoking a program by starting a job step with an EXEC statement. Full details of the four macro-instructions are given in IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions.

To invoke a program with the CALL macro-instruction, the program must first be loaded into main storage, using the LOAD macro-instruction. This returns, in general register 15, the entry address which is used by the CALL macro-instruction. The instructions used could be:

```
LOAD      EP=member-name

LR        15,0

CALL      (15), (option-address), VL
```

To invoke a program with one of the LINK, XCTL or ATTACH macro-instructions would need instructions such as:

```
LINK      EP=member-name,

          PARAM=(option-address), VL=1

XCTL      EP=member-name

ATTACH    EP=member-name,

          PARAM=(option-address), VL=1
```

"member-name" specifies the name of the member of a partitioned data set which contains the program required.

For the compiler, member-name=ALGOL

For the linkage editor, member-name=IEWL

For the load module, member-name is specified by the programmer in the SYSLMOD DD statement for the linkage editor.

"option-address" specifies the address of a list containing the options required by the user. An address must be given even if no options are specified. The list must begin on a half-word boundary. The first two bytes contain a number giving the number of bytes in the remainder of the list. (If no options are specified this number must be zero). The list itself contains any of the options available to the PARM parameter in an EXEC statement (see Appendix E).

When using CALL, LINK or ATTACH to invoke the compiler, other ddnames may be used in place of the standard ddnames given in Section 2 for the data sets (except for SYSABEND), and an alternative page number (instead of the normal 001) may be specified for the start of output listings.

If alternative ddnames are used, then in the statement invoking the compiler, "option-address" must be followed by "ddname-address" giving the address of a list containing the alternative ddnames. If alternative page numbers are used, then "page-address" giving the address of a location containing the alternative page number must be placed after "ddname-address"; though if alternative ddnames are not required "ddname-address" may be replaced by a comma.

The ddname list must begin on a half-word boundary. The first two bytes contain a number giving the number of bytes in the remainder of the list. The list itself contains up to ten 8-byte fields, separated by commas, for specifying alternative ddnames for the data sets. As only seven data sets are used by the compiler, three of the fields are left blank. The alternative ddnames must be listed in the following order:

Purpose of data set	Standard ddname
Output of object module for linkage editor	SYSLIN

When processing the source program, the compiler detects and specifies any routines that need to be combined with the generated object module before it can be executed. These routines are contained in the System/360 Operating System ALGOL library - a partitioned data set with the external name SYS1.ALGLIB. The routines are in load module form and the linkage editor combines them with the object module to produce an executable load module. There are three types of routines - fixed storage area routines, mathematical routines and input/output routines. Additionally, an error routine, stored on the operating system link library, SYS1.LINKLIB, is called at execution time if an error occurs.

Initialization and termination of the library routines is performed using the standard method (see "Supervisor" in Section 1).

FIXED STORAGE AREA

General routines required to some degree by all object modules are combined into a single load module known as the fixed storage area (IHIFSA). These routines are used to initialize and terminate execution of the ALGOL program, to handle the DSA when entering or leaving a program block or procedure, to produce the program trace, to load precompiled procedures, to get main storage for arrays, to convert values from real to integer and integer to real, to call actual parameters, to handle branches in the program, to handle program interrupts, etc....

MATHEMATICAL ROUTINES

Standard mathematical functions contained in ALGOL have corresponding mathematical routines in the library, except for ABS, SIGN and LENGTH which are handled by the compiler, and ENTIER which is contained in the fixed storage area. Routines exist in each case for both long and short precision of real numbers.

These mathematical routines are taken from the System/360 Operating System FORTRAN IV library and modified to conform to the ALGOL language requirements without affecting the mathematical methods used. Full details of these routines are contained in IBM System/360 Operating System: FORTRAN IV Library Sub-programs.

INPUT/OUTPUT ROUTINES

Data transfer between the load module and external data sets is performed by input/output routines. These routines correspond to the ALGOL I/O procedures and are mostly contained on separate load modules (see Figure 18). In addition there is a single load module, IHIIOR, which contains a number of commonly-used subroutines.

ERROR ROUTINE

If an error is detected during execution of the load module, an error routine (in SYS1.LINKLIB) is invoked. Its main purpose is to construct the error message and produce the data storage area listing before passing to the termination routine in the FSA. If a second error occurs while the first is being handled (due, for example, to an I/O error or because the object module has overwritten part of the ALGOL library or control program), then termination takes place immediately and incomplete information listings may be produced.

Module Name		When used	Storage estimate (bytes)
ALGOL	FORTRAN IV		
IHIERR		When an error is detected at execution time	4290
IHIFDD	IHCFDXPD	For an exponentiation (** or 'POWER') using long precision base and long precision exponent	200
IHIFDI	IHCFDXPI	For an exponentiation (** or 'POWER') using long precision base and integer exponent	140
IHIFII	IHCFIXPI	For an exponentiation (** or 'POWER') using integer base and integer exponent	170
IHIFRI	IHCFRXPI	For an exponentiation (** or 'POWER') using hort precision base and integer exponent	140
IHIFRR	IHCFRXPR	For an exponentiation (** or 'POWER') using short precision base and short precision exponent	200
IHIFSA		For every object module (except those for precompiled procedures)	5210
IHIGPR		For either GET or PUT	2430
IHIIAR		For INARRAY or INTARRAY	120
IHIIBA		For INBARRAY	70
IHIIBO		For INBOOLEAN	560
IHIIDE		For either INREAL or ININTEGER	1610
IHIIOR		For every object module	2980
IHIISY		For INSYMBOL	320
IHILAT	IHCLATAN	For a long precision arctangent operation (ARCTAN)	320
IHILEX	IHCLEXP	For a long precision exponential operation (EXP)	450
IHILLO	IHCLLOG	For a long precision logarithmic operation (LN)	310
IHILOR		For a long precision OUTREAL operation	730
IHILSC	IHCLSCN	For a long precision sine or cosine operation (SIN or COS)	370
IHILSQ	IHCLSQRT	For a long precision square root operation (SQRT)	140
IHIOAR		For OUTARRAY	120

IHIOBA		For OUTBARRAY	70
IHIOBO		For OUTBOOLEAN	400
IHIOIN		For OUTINTEGER	420
IHIOST		For OUTSTRING	300
IHIOSY		For OUTSYMBOL	290
IHIOTA		For OUTTARRAY	120
IHIPTT		For INREAL, OUTREAL, ININTEGER or OUTINTEGER	270
IHISAT	IHCSATAN	For a short precision arctangent operation (ARCTAN)	200
IHISEX	IHCSEXP	For a short precision exponential operation (EXP)	280
IHISLO	IHCSLOG	For a short precision logarithmic operation (LN)	210
IHISOR		For a short precision OUTREAL operation	810
IHISSC	IHCSSCN	For a short precision sine or cosine operation (SIN or COS)	260
IHISSQ	IHCSSQRT	For a short precision square root operation (SQRT)	170
IHISYS		For SYSACT	1890

• Figure 18. Table of ALGOL library modules. All are contained in SYS1.ALGLIB except IHIERR which is in SYS1.LINKLIB. For mathematical routines, the corresponding name in the FORTRAN IV library is also given.

APPENDIX B: IBM-SUPPLIED CATALOGED PROCEDURES

The three cataloged procedures for ALGOL that were introduced in Section 2 are contained in the procedure library, SYS1.PROCLIB, of the operating system. They consist of the job control statements listed below.

These procedures have been designed for an optimum job, and can be over-ridden by the user if he requires different or additional system support to that provided (see Section 2). In particular it should be noted that in these procedures the object or load module produced is stored on a temporary data set and will therefore be deleted at the end of the job.

Compilation, ALGOFC

```
//ALGOL EXEC PGM=ALGOL
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD SYSOUT=B
//SYSLIN DD DSNAME=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,DISP=(MOD,PASS),     X
//             SPACE=(4ØØ,(4Ø,1Ø))
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT2 DD UNIT=SYSSQ,SEP=(SYSUT1,SYSLIN,SYSPUNCH),                    X
//             SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT3 DD UNIT=SYSDA,SPACE=(2ØØØ,(2Ø,5)),                             X
//             SEP=(SYSUT1,SYSUT2,SYSLIN,SYSPUNCH)
//SYSABEND DD SYSOUT=A
```

Compilation and Linkage Editing, ALGOFCL

```
//ALGOL EXEC PGM=ALGOL
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD SYSOUT=B
//SYSLIN DD DSNAME=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,DISP=(NEW,PASS),     X
//             SPACE=(4ØØ,(4Ø,1Ø))
//SYSLMOD DD DSNAME=&GOSET,UNIT=SYSDA,DISP=(MOD,PASS),                  X
//             SPACE=(1Ø24,(5Ø,2Ø,1))
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT2 DD UNIT=SYSSQ,SEP=(SYSUT1,SYSLIN,SYSPUNCH),                    X
//             SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT3 DD UNIT=SYSDA,SPACE=(2ØØØ,(2Ø,5))                              X
//             SEP=(SYSUT1,SYSUT2,SYSLIN,SYSPUNCH)
//SYSABEND DD SYSOUT=A
//LKED EXEC PGM=IEWL,PARM=(XREF,LIST,LET),COND=(5,LT,ALGOL)
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSNAME=*.ALGOL.SYSLIN,DISP=(OLD,DELETE)
//SYSLIB DD DSNAME=SYS1.ALGLIB,DISP=OLD
//SYSLMOD DD DSNAME=&GOSET(GO),UNIT=SYSDA,DISP=(MOD,PASS),              X
//             SPACE=(1Ø24,(5Ø,2Ø,1))
//SYSUT1 DD UNIT=SYSDA,SEP=(SYSLIN,SYSLIB,SYSLMOD),                     X
//             SPACE=(1Ø24,(5Ø,2Ø))
//SYSABEND DD SYSOUT=A
```

```
//ALGOL EXEC PGM=ALGOL
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD SYSOUT=B
//SYSLIN DD DSNAME=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,DISP=(NEW,PASS),     X
//               SPACE=(400,(40,10))
//SYSLMOD DD DSNAME=&GOSET,UNIT=SYSDA,DISP=(MOD,PASS),                   X
//               SPACE=(1024,(50,20,1))
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(1024,(50,10))
//SYSUT2 DD UNIT=SYSSQ,SEP=(SYSUT1,SYSLIN,SYSPUNCH),                     X
//               SPACE=(1024,(50,10))
//SYSUT3 DD UNIT=SYSDA,SPACE=(2000,(20,5)),                              X
//               SEP=(SYSUT1,SYSUT2,SYSLIN,SYSPUNCH)
//SYSABEND DD SYSOUT=A
//LKED EXEC PGM=IEWL,PARM=(XREF,LIST,LET),COND=(5,LT,ALGOL)
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSNAME=*.ALGOL.SYSLIN,DISP=(OLD,DELETE)
//SYSLIB DD DSNAME=SYS1.ALGLIB,DISP=OLD
//SYSLMOD DD DSNAME=&GOSET(GO),UNIT=SYSDA,DISP=(OLD,PASS),               X
//               SPACE=(1024,(50,20,1))
//SYSUT1 DD UNIT=SYSDA,SEP=(SYSLIN,SYSLIB,SYSLMOD),                      X
//               SPACE=(1024,(50,20))
//SYSABEND DD SYSOUT=A
//GO EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,ALGOL),(5,LT,LKED))
//ALGLDD01 DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(TRK,(5,2))
//SYSABEND DD SYSOUT=A
```

APPENDIX C: CARD CODES

The card deck of the source program is punched line for line from the text written on the coding sheets. The card code used can be either a 53 character set in Extended Binary Coded Decimal Interchange Code (EBCDIC), or a 46 character set in Binary Coded Decimal (BCD). This latter character set has been established as standard for ALGOL by the International Standards Organization (ISO) and Deutsche Industrie Normen (DIN). Figure 19 shows these two codes.

Characters	Card Codes	
	EBCDIC	ISO/DIN
A to Z	12-1 to 0-9	12-1 to 0-9
0 to 9	0 to 9	0 to 9
+	12-8-6	12
-	11	11
*	11-8-4	11-8-4
/	0-1	0-1
=	8-6	8-3
,	0-8-3	0-8-3
.	12-8-3	12-8-3
´	8-5	8-4
(	12-8-5	0-8-4
)	11-8-5	12-8-4
blank	no punch	no punch
<	12-8-4	
>	0-8-6	
|	12-8-7	
&	12	
→	11-8-7	
:	8-2	
;	11-8-6	

Figure 19. Source program card codes.

reached the end of its tape reel, output would be automatically continued on one of the additional units, and the first tape reel would be rewound and then replaced by the operator with a new reel so that the unit would be available for other data sets. The pool would be established by using the first form of the UNIT "subparameter-list" in a DD statement. Only the AFF or SEP parameters may be used with the UNIT parameter in this statement.

1 or 0 indicates that an extra tape unit is either to be added to the pool, or not to be added to the pool.

AFF=ddname indicates that the data set is to use the same I/O devices as the data set specified in the DD statement named "ddname" in the same job step.

SPACE=subparameter-list
indicates the space required when a direct access device is specified in the UNIT parameter. The "subparameter-list" contains only positional subparameters. The list is:

$$\left\{\begin{array}{l}\text{TRK}\\ \text{CYL}\\ \text{average-record-length}\end{array}\right\} \text{primary-quantity}$$

[secondary-quantity][directory-or-index-quantity]

$$\text{[RLSE][}\left\{\begin{array}{l}\text{MXIG}\\ \text{ALX}\\ \text{CONTIG}\end{array}\right\}\text{][ROUND]}$$

The first subparameter specifies the units in which the space requirements are expressed, that is, tracks, cylinders or records (with length given in bytes).

The next subparameter specifies the space required. It has three parts (of which the second and third are optional) and is enclosed in parentheses if more than one part is specified. If the second part is omitted, then it must be substituted by a comma if the third part is included. The initial space to be allocated is given by "primary-quantity". Each time this initial space is filled, additional space is to be provided as specified by "secondary-quantity". The number of 256 byte records to be allocated for the directory of a new partitioned data set, or the number of cylinders, taken from the initial space reserved, to be allocated for the index of an indexed sequential data set, is given by "directory-or-index-quantity".

RLSE indicates that any unused space assigned to the data set is to be released.

MXIG requests that the largest single block of storage available is to be allocated to the data set.

ALX requests that extra blocks of storage (in track units) are to be allocated to the data set. As many available blocks that are equal to or greater than "primary-quantity", up to a maximum of five, will be allocated.

CONTIG specifies that the space specified by "primary-quantity" is to be in a single block.

ROUND requests that when records are used to express the space required on the direct access device, the space is to begin and end on cylinder boundaries.

DISP=subparameter-list
indicates the status of the data set and specifies its disposition at the end of the job step. The "subparameter-list" consists of the following positional subparameters:

$$\left\{\begin{array}{l}\underline{\text{NEW}}\\ \text{OLD}\\ \text{MOD}\end{array}\right\} \left[\left\{\begin{array}{l}\text{DELETE}\\ \text{KEEP}\\ \text{PASS}\\ \text{CATLG}\\ \text{UNCATLG}\end{array}\right\}\right]$$

NEW specifies that the data set is to be generated in this job step, and would be deleted at the end of the job step unless KEEP, PASS or CATLG is specified.

OLD specifies that the data set already exists, and would be kept at the end of the job step unless PASS or DELETE is specified.

MOD specifies that the data set already exists and is to be modified in this job step. If the data set cannot be found by the operating system then this parameter is equivalent to NEW.

DELETE specifies that the space used by the data set (including that in the data set catalog, etc.) is to be released at the end of the job step.

KEEP specifies that the data set is to be kept at the end of the job step.

PASS specifies that the data set is to be referred to in a later step of this job, at which

time its final disposition, or a further pass, will be specified.

CATLG specifies that the data set is to be cataloged at the end of the job step. Thus KEEP is implied. The catalog structure must already exist.

UNCATLG specifies that the data set is to be deleted from the catalog at the end of the job step. KEEP is implied.

SYSOUT=subparameter-list
specifies the printing or punching operation to be used for the data set. The "subparameter-list" is:

classname [progname][number]

"classname specifies the system output class to be used. Up to 36 different classes (A to Z, 0 to 9) may be fixed by the user for his installation, according to device type, priority, destination, etc. The standard classname is A.

"progname" can be used to specify the name of a user-written output routine.

"number" can be used to specify an installation form number to be assigned to the output.

For sequential scheduling, the "subparameter-list" consists of only the standard class-names A and B. SYSOUT=B is interpreted as UNIT=SYSCP.

VOLUME=subparameter-list
indicates the volume or volumes assigned to the data set. If the data set is cataloged this parameter is not necessary. The "subparameter-list" is:

Positional subparameters	[RETAIN][number][value]

Keyword subparameters	SER=list-of-serial-numbers
	REF= { dsname | *. ddname | *. stepname. ddname | *. stepname. procstep. ddname }

RETAIN specifies that, if possible, the volume is to remain mounted until referred to in a later DD statement, or until the end of the job, whichever is first. "number" is any number between 2 and 9999, and is used if an input or output operation, on a cataloged data set residing on more than one volume, does not start on the first volume of the data set. The number specifies the position of the volume on which input or output does start (for example, 3 indicates the third volume of the data set).

"value" specifies the number of volumes required by an output data set. It is not required if SER or REF is used.

SER=list-of-serial-numbers, specifies the serial numbers allocated by the user to the volumes required by the data set. These serial numbers can consist of between one and six alphameric characters.

```
     ⎧ dsname                          ⎫
REF= ⎨ *. ddname                       ⎬
     ⎨ *. stepname. ddname             ⎬
     ⎩ *. stepname. procstep. ddname   ⎭
```

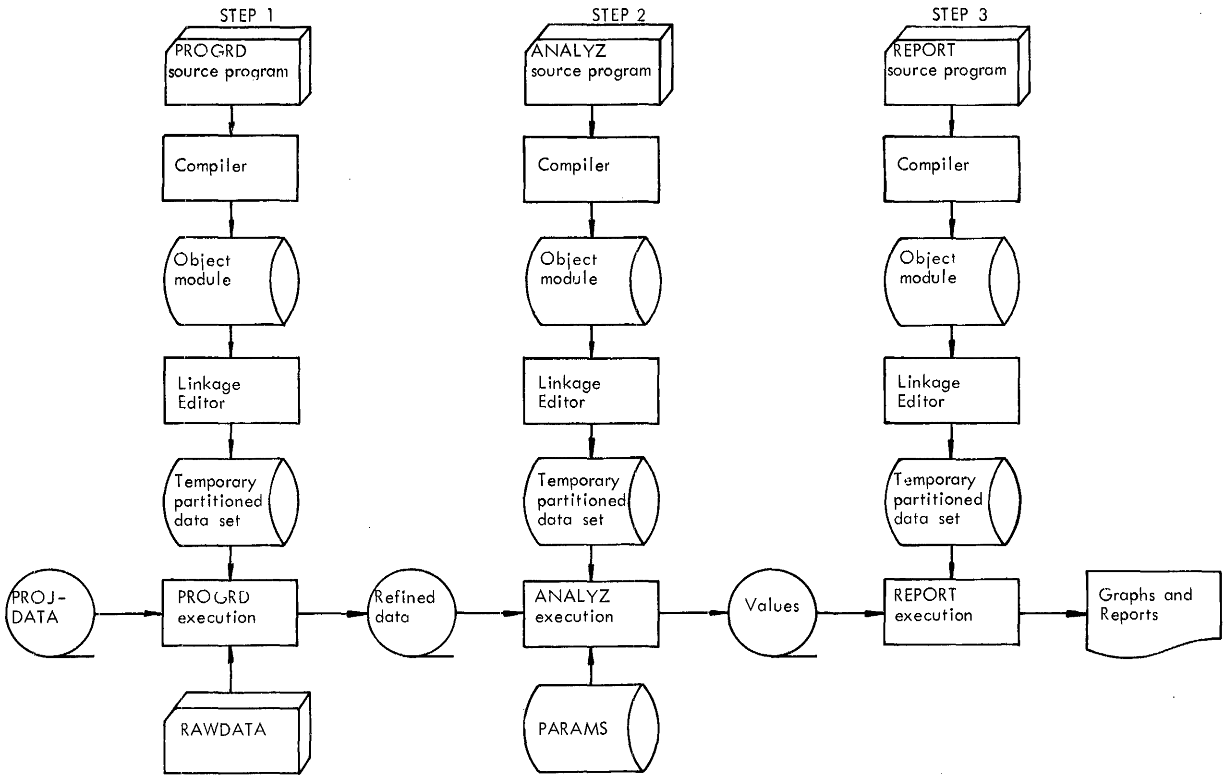
specifies that this data set is to use the same volume or volumes as the data set specified by one of the alternative sub-subparameter forms. If the latter data set resides on more than one tape volume, then only the last volume (as specified in the SER subparameter) can be used.

LABEL=subparameter-list
indicates the type of label or labels associated with the data set. If the data set is cataloged this parameter is not necessary. The "subparameter-list" is:

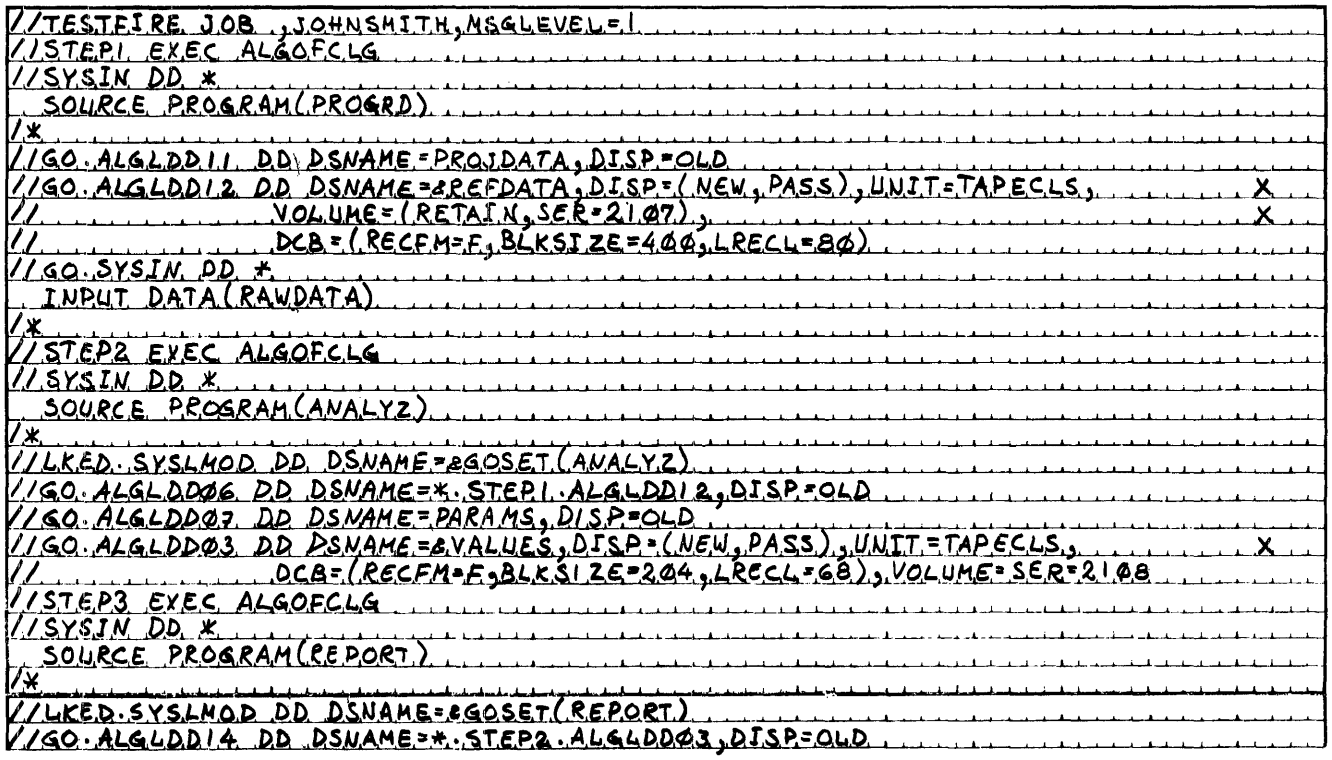
Positional subparameters	[number]	{ NL | <u>SL</u> | NSL | SUL | BLP }
Keyword subparameters	{ EXPDT=yyddd | RETPD=dddd }	

"number" is any number between 2 and 9999, and specifies the position of the data set on the volume (for example, 3 would indicate the third data set on the volume).

NL, SL, NSL, and SUL specify the type of label or labels to be used, that is, no labels, standard labels, non-standard labels, and standard



•Figure 28. Basic I/O flow for Example 2. The data sets for information listings, ALGOL library routines intermediate work and the execution time error routine are not shown.



```
//TESTFIRE JOB ,JOHNSMITH,MSGLEVEL=1
//STEP1 EXEC ALGOFCLG
//SYSIN DD *
  SOURCE PROGRAM(PROGRD)
/*
//GO.ALGLDD11 DD DSNAME=PROJDATA,DISP=OLD
//GO.ALGLDD12 DD DSNAME=&REFDATA,DISP=(NEW,PASS),UNIT=TAPECLS,           X
//                 VOLUME=(RETAIN,SER=21Ø7),                             X
//                 DCB=(RECFM=F,BLKSIZE=4ØØ,LRECL=8Ø)
//GO.SYSIN DD *
  INPUT DATA(RAWDATA)
/*
//STEP2 EXEC ALGOFCLG
//SYSIN DD *
  SOURCE PROGRAM(ANALYZ)
/*
//LKED.SYSLMOD DD DSNAME=&GOSET(ANALYZ)
//GO.ALGLDDØ6 DD DSNAME=*.STEP1.ALGLDD12,DISP=OLD
//GO.ALGLDDØ7 DD DSNAME=PARAMS,DISP=OLD
//GO.ALGLDDØ3 DD DSNAME=&VALUES,DISP=(NEW,PASS),UNIT=TAPECLS,            X
//                 DCB=(RECFM=F,BLKSIZE=2Ø4,LRECL=68),VOLUME=SER=21Ø8
//STEP3 EXEC ALGOFCLG
//SYSIN DD *
  SOURCE PROGRAM(REPORT)
/*
//LKED.SYSLMOD DD DSNAME=&GOSET(REPORT)
//GO.ALGLDD14 DD DSNAME=*.STEP2.ALGLDDØ3,DISP=OLD
```

Figure 29. Job control statements for Example 2.

occurs) printed on the normal system output device for information listings

2. The first job step invokes the ALGOFCLG cataloged procedure (see Appendix B) to process and execute the ALGOL source program (PROGRD) entered in the input stream

3. The other input data sets are RAWDATA and PROJDATA. RAWDATA is also entered in the input stream

4. The temporary output data set is

   - to be called REFDATA. TESTFIRE and to be passed for use in a later job step
   - to use the device class TAPECLS
   - to be written on volume 2107, which is to remain mounted for use later
   - to have fixed length records, 80 bytes long, and a maximum block size of 400 bytes

5. The second job step invokes the ALGOFCLG cataloged procedure to process and execute the ALGOL source program (ANALYZ) entered in the input stream

6. The SYSLMOD DD statement in the LKED step of the cataloged procedure is overridden to specify that the load module produced by the linkage editor is

   - to be a new member, PROGRD, of the temporary partitioned data set FIRING

7. The other input data sets are REFDATA. TESTFIRE and PARAMS. Both will be kept at the end of the job step

8. The temporary output data set is

   - to be called VALUES. TESTFIRE and is to be passed for use in a later job step
   - to use the device class TAPECLS
   - to be written on volume 2108
   - to have fixed length records, 68 bytes long, and a maximum block size of 204 bytes

9. The third job step invokes the ALGOFCLG cataloged procedure to process and execute the ALGOL source program (REPORT) entered in the input stream. The output data will be listed on the printer specified in the cataloged procedure

10. The SYSLMOD DD statement in the LKED step of the cataloged procedure is over-ridden to specify that the load module produced by the linkage editor is

    - to be a new member, REPORT, of the temporary partitioned data set FIRING

11. The other input data set is VALUES. TESTFIRE which will be kept at the end of the job step

Example 3: Executing Two Load Modules

Statement of problem: Data on current weather conditions is to be read from cards and used by the program FILECR to create a new generation of a data set WEATHER, and also to print a report.

Then the new generation and the three immediately preceding generations of the WEATHER data set are to be used by the program FORCST to produce a printed weather forecast. The pro-

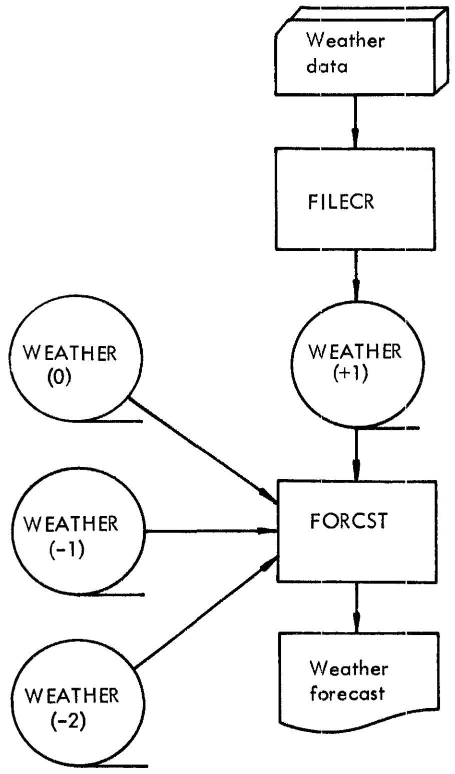
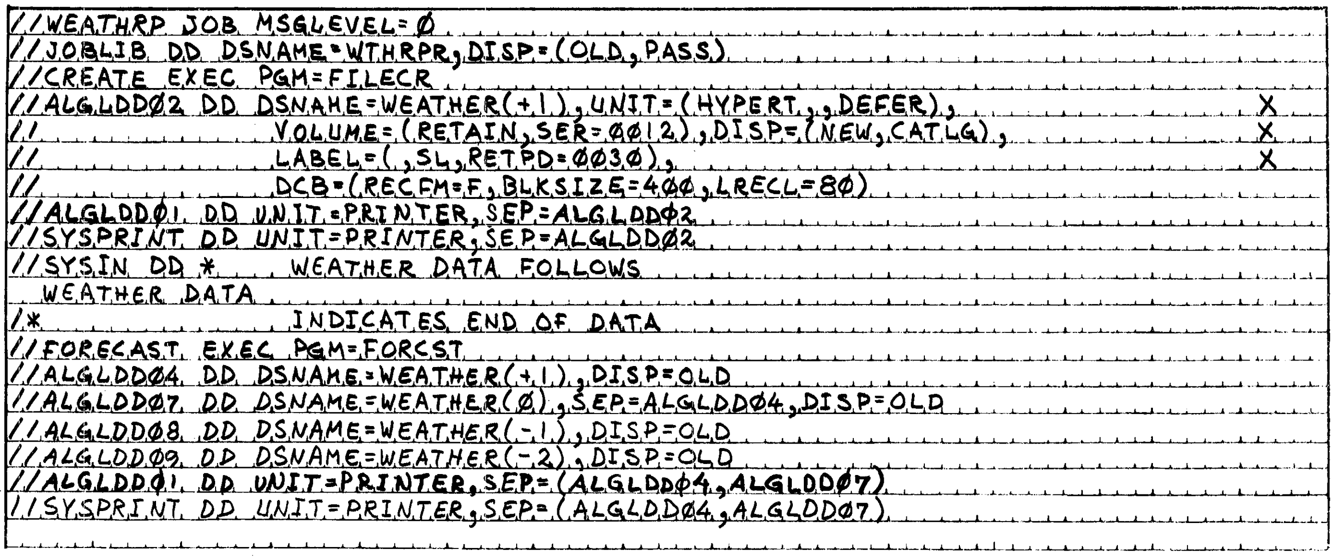


Figure 30. I/O flow for Example 3.



```
//WEATHRP JOB MSGLEVEL=Ø
//JOBLIB DD DSNAME=WTHRPR,DISP=(OLD,PASS)
//CREATE EXEC PGM=FILECR
//ALGLDDØ2 DD DSNAME=WEATHER(+1),UNIT=(HYPERT,,DEFER),                  X
//             VOLUME=(RETAIN,SER=ØØ12),DISP=(NEW,CATLG),                X
//             LABEL=(,SL,RETPD=ØØ3Ø),                                   X
//             DCB=(RECFM=F,BLKSIZE=4ØØ,LRECL=8Ø)
//ALGLDDØ1 DD UNIT=PRINTER,SEP=ALGLDDØ2
//SYSPRINT DD UNIT=PRINTER,SEP=ALGLDDØ2
//SYSIN DD *     WEATHER DATA FOLLOWS
 WEATHER DATA
/*               INDICATES END OF DATA
//FORECAST EXEC PGM=FORCST
//ALGLDDØ4 DD DSNAME=WEATHER(+1),DISP=OLD
//ALGLDDØ7 DD DSNAME=WEATHER(Ø),SEP=ALGLDDØ4,DISP=OLD
//ALGLDDØ8 DD DSNAME=WEATHER(-1),DISP=OLD
//ALGLDDØ9 DD DSNAME=WEATHER(-2),DISP=OLD
//ALGLDDØ1 DD UNIT=PRINTER,SEP=(ALGLDDØ4,ALGLDDØ7)
//SYSPRINT DD UNIT=PRINTER,SEP=(ALGLDDØ4,ALGLDDØ7)
```

•Figure 31. Job control statements for Example 3.

grams FILECR and FORCST are contained in a partitioned data set WTHRPR.

Explanation of coding: The job control statements used in Figure 31 specify that:

1. The job is to have control statement messages plus the relevant control statement printed on the normal system output device only if an error occurs

2. The partitioned data set WTHRPR is concatenated to the operating system library, SYS1. LINKLIB

3. The first job step executes the program FILECR

4. The output data set is

   - a new generation of the data set WEATHER
   - to use the device class HYPERT
   - to be written on volume 0012 which need not be mounted until the data set is opened, and is then to remain mounted for later use
   - to be cataloged and have standard labels
   - to be retained for 30 days
   - to have fixed length records, 80 bytes long, and a maximum block size of 400 bytes

5. The printed output is

   - to use the device class PRINTER
   - to use a separate channel to the output data set

6. The input data is included in the input stream

7. The second job step executes the program FORCST

8. The input data sets are the last four generations of WEATHER, all of which are to be kept at the end of the job step

9. The output data set is

   - to use the device class PRINTER
   - to use a separate channel to the last two generations of WEATHER

APPENDIX F: DIAGNOSTIC MESSAGES

Each of the three operations-compilation, linkage editing and execution - may produce diagnostic messages.

COMPILER MESSAGES

The diagnostic messages that may be produced by the ALGOL compiler are given below. Each diagnostic message occupies one or more printed lines and contains:

- The message key, consisting of the letters IEX, a three digit decimal number identifying the message, and the letter I to indicate an informative message requiring no action from the operator.
- The severity code W, S or T (see below)
- The semicolon number (see Section 3). This number is sometimes omitted if the error cannot be directly related to a point in the program. The semicolon number is indicated in the list below by the sequence NNNNN
- The message text describing the error and, in the case of some W or S type errors, the modification performed on the program by the compiler. In the message text listed below the words in parentheses, together with the parentheses themselves, will be replaced in the actual message that is printed, by specific information taken from the program. The word "operator" usually refers to all delimiters defined in IBM System/360 Operating System: ALGOL Language, but an internal compiler operator may sometimes be listed. The word "operand" refers to an identifier or an expression.

The three severity codes for errors and their corresponding compiler action are as follows:

W (Warning): The program is modified internally and the compilation is continued. The modification may not make the program correct but it allows object module generation to continue. A diagnostic message is produced.

S (Serious): An attempt is made to modify the program internally, including skipping or changing parts of it. Generation of the object module is stopped, but syntax checking continues. A diagnostic message is produced.

T (Terminating): A diagnostic message is produced and the compilation is terminated.

IEX001I W NNNNN INVALID CHARACTER DELETED.

Explanation: A character not recognized by the compiler has been deleted from the program.

IEX002I W NNNNN ILLEGAL PERIOD. PERIOD DELETED.

Explanation: The character period has been used wrongly and deleted from the program. It can be used only as a decimal point, or as part of a colon or semicolon.

IEX003I W NNNNN INVALID COLON AFTER (six characters). COLON DELETED.

Explanation: The character colon has been used wrongly and has been deleted from the program. It can be used only after a label, between subscript bounds, within a parameter delimiter or as part of an assign symbol.

IEX004I T NNNNN LETTER STRING TOO LONG.

Explanation: A letter string used to supply explanatory information exceeds capacity limitations (see Section 4).

IEX005I S NNNNN IDENTIFIER BEGINS WITH INVALID CHARACTER. IDENTIFIER DELETED.

Explanation: An identifier has been deleted because it does not begin with an alphabetic character.

IEX006I T NNNNN LABEL CONTAINS TOO MANY CHARACTERS.

Explanation: A label identifier has been used whose length exceeds capacity limitations (see Section 4).

IBM Technical Newsletter

File Number	S360-26
Re: Form No.	C33-4000-0
This Newsletter No.	N33-8013
Date	June 26, 1968
Previous Newsletter Nos.	N33-8000, N33-8002, N33-8012

IBM System/360 Operating System ALGOL Programmer's Guide

This Technical Newsletter relates to Release 16 of the Operating System and contains amendments to the IBM System/360 Operating System: ALGOL Programmer's Guide, Form C33-4000-0. The attached pages are replacements to be inserted in the publication, as indicated below. Corrections and additions to the text and/or illustrations are indicated by a vertical bar to the left of the affected text and by a bullet (•) to the left of the figure caption.

Pages to be Removed	Pages to be Inserted
1-4	1-4
11-18	11-18
25-30	25-28
	29-29.1
	29.2-30
	30.1-30.2
33-36	33-36
45-46	45-46
51-52	51-51.1
	51.2-52

Summary of Amendments

The amendments in this newsletter reflect changes in IBM-supplied cataloged procedures in order to provide MVT support for the ALGOL compiler under Release 16. The text relating to precompiled procedures is also amended and examples have been added to illustrate an Assembler language procedure and the use of job control statements.

File this cover letter at the back of the publication. It will serve as a record of the changes received and incorporated.

IBM Nordic Laboratory, Technical Communications, Box 962, Lidingö 9, Sweden

File No. S360-26
Form C33-4000-0

OS

IBM Systems Reference Library

IBM System/360 Operating System

ALGOL Programmer's Guide

Program Number 360S-AL-531....Compiler
360S-LM-532....Library Routines

This publication describes how to compile, linkage edit and execute a program written in the System/360 Operating System Algorithmic Language (ALGOL). It includes an introduction to the operating system and a description of the information listings that can be produced, the job control language, and the subroutine library.

PREFACE

This publication is intended for use by Application Programmers, Systems Programmers and IBM Systems Engineers. A knowledge of ALGOL is assumed, and the reader is expected to be familiar with the prerequisite publication:

IBM System/360 Operating System: ALGOL Language. Form C28-6615.

In Section 2, the description "IBM-Supplied Cataloged Procedures" provides sufficient information to process and execute an ALGOL program that can use the IBM-supplied cataloged procedures without modification.

The rest of Section 2, together with information in Section 1 and the Appendices, will be required for programs that cannot use the IBM-supplied cataloged procedures without modification.

The description of information listings in Section 3 and the list of diagnostic messages given in Appendix F will be helpful in interpreting system output, especially for debugging.

An extensive index has been provided to assist the reader in using the manual for reference purposes.

This publication contains most of the information required by the Applications Programmer.

The following publications are referred to within the text for information beyond the scope of this publication.

IBM System/360 Operating System:

Assembler Language, Form C28-6514

Linkage Editor, Form C28-6538

Job Control Language, Form C28-6539

Operator's Guide, Form C28-6540

Utilities, Form C28-6586

FORTRAN IV Library Subprograms, Form C28-6596

Message Completion Codes, and Storage Dumps, Form C28-6631

Supervisor and Data Management Services, Form C28-6646

Supervisor and Data Management Macro-Instructions, Form C28-6647

System Programmer's Guide, Form C28-6550

First Edition (April 1967)
Significant changes or additions to the specifications contained in this publication will be reported in subsequent revisions or Technical Newsletters.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Department 813, 112 East Post Road, White Plains, N.Y. 10601, or IBM Nordic Laboratory, Technical Communications, Vesslevägen 3, Lidingö, Sweden.

CONTENTS

FIGURES

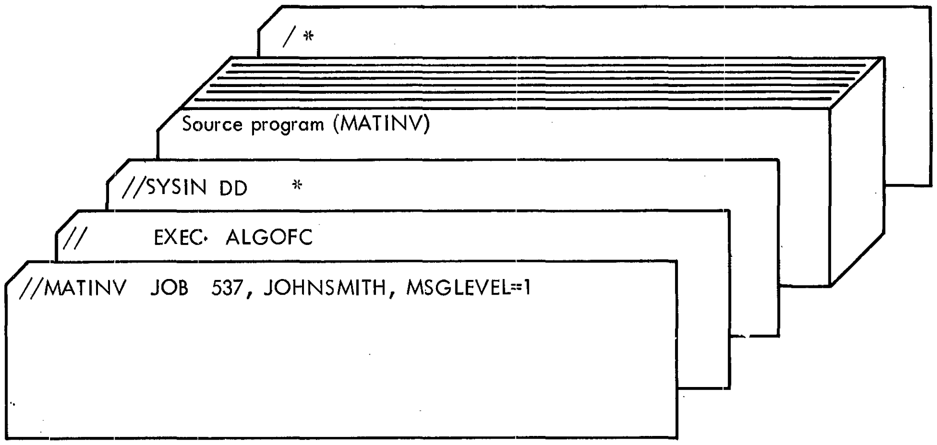


Figure 2. Sample deck for using ALGOFC cataloged procedure with a single source program. This job compiles the MATINV source program used in Example 1 of Appendix E.

If more than one source program is to be processed in the same job, then all job control statements except the JOB statement must be repeated for each source program.

If it is required to keep a load module for use in a later job (as in the case when the load module is a precompiled procedure), then the SYSLMOD DD statement in the cataloged procedure must be over-ridden to specify a permanent data set. This has to be done for each load module that is kept. The over-riding statement is placed at the end of the job step to which it applies, and has the form:

```
//LKED.SYSLMOD DD DSNAME=dsname(member),
                  DISP=(MOD,KEEP)
```

where "dsname" is the name of a partitioned data set and "member" is the member name assigned to the load module on the partitioned data set.

Figure 32 shows the job control statements needed to compile and linkage edit a precompiled procedure.

A sample deck of job control statements to compile and linkage edit two source programs is shown in Figure 3.

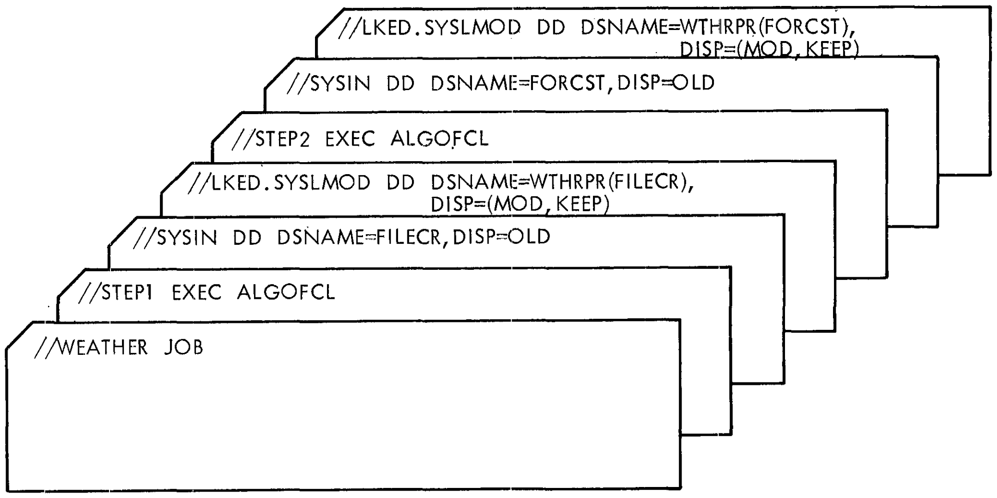

//LKED.SYSLMOD DD DSNAME=WTHRPR(FORCST),
DISP=(MOD,KEEP)
//SYSIN DD DSNAME=FORCST,DISP=OLD
//STEP2 EXEC ALGOFCL
//LKED.SYSLMOD DD DSNAME=WTHRPR(FILECR),
DISP=(MOD,KEEP)
//SYSIN DD DSNAME=FILECR,DISP=OLD
//STEP1 EXEC ALGOFCL
//WEATHER JOB

Figure 3. Sample deck for using ALGOFCL cataloged procedure with two source programs. These two job steps compile and linkage edit the two source programs used in Example 3 of Appendix E. Both source programs have been previously stored on intermediate I/O devices.

Compilation, Linkage Editing and Execution

The cataloged procedure used to compile an ALGOL source program, linkage edit the resulting object module, and execute the load module produced by the linkage editor is ALGOFCLG.

The statements used in this cataloged procedure are shown in Appendix B. The following statements can be used to invoke the ALGOFCLG cataloged procedure:

```
//jobname    JOB
//JOBLIB     DD  DSNAME=dsname1, DISP=OLD
//           EXEC  ALGOFCLG
//SYSIN      DD    {* or parameters defining an
                      input data set containing
                      the source program }
//GO.ALGLDD02  DD  DSNAME=dsname2
                                      .
                                      .
                                      .
//GO.ALGLDD15  DD  DSNAME=dsname15
```

where "jobname" is the name assigned to the job. "dsname1" is the name of a data set that contains a precompiled procedure (see Section 4) which is called by the load module being executed. The DD statement containing dsname1 need not be used if no precompiled procedure is used.

For a description of the correct use of the JOBLIB DD statement when more than one precompiled procedure is used in a job, or when a precompiled procedure resides on more than one data set, see "Data Set Concatenation" in Appendix E.

"dsname2"... "dsname15" are the names of input data sets required by the load module at execution time and output data sets to be created at execution time. In addition, two data sets for printed output (ddnames SYSPRINT and ALGLDD01) are supplied by the cataloged procedure, and a data set for input only can be specified by using the following statement after the invoking sequence just given.

```
//GO.SYSIN  DD  {* or parameters defining an
                   input data set }
```

If DD* is used then the data must follow immediately afterwards in the input stream. For sequential scheduling, the data must be followed by a delimiter statement (/*).

If more than one source program is to be processed and executed in the same job, then all job control statements except the JOB statement and the JOBLIB DD statement must be repeated for each source program.

A sample deck of job control statements required to compile, linkage edit and execute three source programs is shown in Figure 29.

Over-riding Cataloged Procedures

The programmer can change any of the statements in a cataloged procedure, except the name of the program in an EXEC statement.

These over-riding conditions are temporary, and will be in effect only until the next job step is started. The following text describes methods of temporarily modifying existing parameters and adding new parameters to the EXEC and DD statements used in the cataloged procedures. The full list of parameters available to the ALGOL programmer for these statements, and detailed explanations of the parameters, is given in Appendix E. The EXEC and DD statements used in the IBM-supplied cataloged procedures are shown in Appendix B.

Over-riding EXEC Statements

In the EXEC statement, the programmer can change or add any of the keyword parameters by using the following format:

```
keyword.procstep=option
```

where:

"keyword" is the parameter to be changed in, or added to, the specified procedure job step: either COND, PARM, ACCT, TIME or REGION. TIME and REGION are valid only for priority scheduling.

"procstep" is the procedure job step in which the change or addition is to occur: either ALGOL, LKED or GO.

"option" is the new option required.

For example, if the EXEC statement used to invoke the ALGOFCLG cataloged procedure was written as:

```
// EXEC  ALGOFCLG,PARM.ALGOL=DECK,
//                 PARM.LKED=XREF,
//                 COND.GO=(3,LT,ALGOL)
```

then the following changes would be made to the ALGOFCLG cataloged procedure:

1. In the PARM parameter of the job step ALGOL, the option DECK would be used instead of the default option NODECK (assuming that the standard default NODECK was not changed at system generation). Over-riding this option will not affect the other default options assumed for this parameter.

2. In the job step LKED, the option XREF is specified for the PARM parameter. Since the options specified in the cataloged procedure were XREF, LIST and LET, this statement has the effect of deleting the options LIST and LET since they were not default options.

3. In the job step GO, the COND parameter code is changed from 5, as it appears in the cataloged procedure, to 3. In this example, the code 3 causes the job step GO to be bypassed if a warning message is generated during the job step ALGOL. Note that although the other options (LT and ALGOL) are not to be altered, the entire parameter being modified must be respecified.

If "procstep" is not specified when over-riding a multi-step cataloged procedure, the operating system makes the following assumptions:

- COND, ACCT and REGION parameters apply to all procedure job steps.
- A PARM parameter applies to the first procedure job step and any options already specified in the PARM parameters for the remaining procedure job steps are cancelled.
- A TIME parameter specifies the computing time for the entire job and any options already specified in the TIME parameters for individual procedure job steps are cancelled.

Over-riding DD Statements

An additional DD statement is used in the invoking sequence for each DD statement in the cataloged procedure that is to be over-ridden. The following format is used:

```
//procstep.ddname  DD  parameter-list
```

where:

"procstep" is the procedure job step containing the DD statement to be over-ridden: either ALGOL, LKED or GO. If "procstep" is omitted then the first procedure job step is assumed.

"ddname" is the name of the DD statement to be over-ridden.

"parameter-list" is the list of parameters that are being added or changed. In both cases the whole parameter must be specified. Unchanged parameters in the original statement need not be specified. For example, the statement:

```
//ALGOL.SYSLIN  DD  SPACE=(400,(80,10))
```

will change the SPACE parameter of the SYSLIN DD statement in the ALGOL job step so that space will be allocated for 80 physical records instead of 40.

DD statements that are used to over-ride other DD statements in the cataloged procedures must be placed immediately after the EXEC statement invoking the cataloged procedure, and must be in the same order as their corresponding DD statements in the cataloged procedures.

Adding DD Statements

Complete, new DD statements that are to be added to the cataloged procedure use the same format as over-riding DD statements. The "ddname" specified must not exist in the job step specified by "procstep". These new DD statements must follow immediately after the over-riding DD statements which apply to the same procedure job step.

USER-WRITTEN PROCEDURES

The information required by the programmer to write his own job control procedures is given in the following text, and in Appendix E. Cataloging user-written procedures, or permanently modifying the IBM-supplied cataloged procedures, is accomplished using the IEBUPDTE utility program, described in IBM System/360 Operating System: Utilities. The statements required in user-written procedures are:

- An EXEC statement to invoke the program.
- DD statements to define the data sets used by the program.

Compilation

Invoking Statement

The ALGOL compiler consists of ten load modules contained in the link library, SYS1.LINKLIB, of the operating system. The compiler is activated

by invoking its first load module, named ALGOL, which then internally invokes the other load modules of the compiler.

The usual method of invoking the compiler is by means of an EXEC statement of the form:

```
//stepname  EXEC  PGM=ALGOL
```

where "stepname" is the name assigned to the job step (optional).

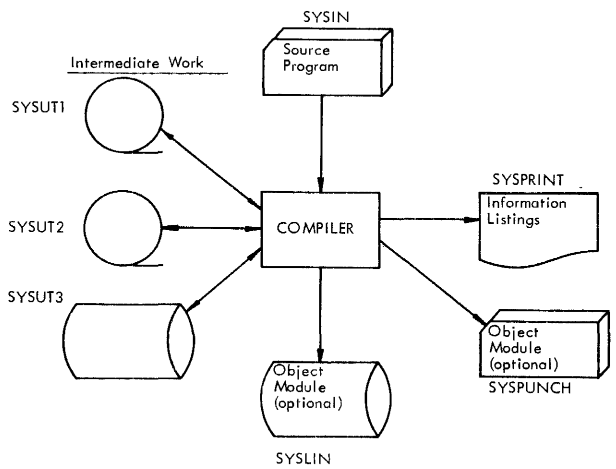
Other EXEC statement parameters may be included if required (see Appendix E).

(A method of dynamically invoking the compiler within a job step, by means of the CALL, LINK, XCTL or ATTACH macro-instructions, is described in Section 4.)

Data Sets Used

The data sets used in the compilation process are illustrated in Figure 4, and described in Figure 5. These data sets must be specified by the programmer with suitable DD statements.

Blocksize DCB information may be specified by the user for SYSIN, SYSLIN, SYSPRINT and SYSPUNCH. The maximum blocking factor depends on the main storage size available (see Figure 6). Record length is fixed at 80 bytes for SYSIN, SYSLIN and SYSPUNCH, and 91 bytes for SYSPRINT.



•Figure 4. Flowchart showing data sets used by the compiler.

The space required for the compiler data sets depends on the size and structure of the source program, however it can be assumed that only in rare cases will the object module exceed four times the source program and usually much less will be required.

Purpose	Standard ddname	Devices required
For ALGOL source program	SYSIN	Card reader*
For object module to be used by linkage editor	SYSLIN	Direct access or magnetic tape
For compilation listings	SYSPRINT	Printer*
For object module (copied from SYSLIN)	SYSPUNCH	Card punch*
For intermediate compiler working	SYSUT1	Direct access or magnetic tape
For intermediate compiler working	SYSUT2	Direct access or magnetic tape
For intermediate compiler working	SYSUT3	Direct access

* Some form of intermediate storage, such as magnetic tape, may be used to reduce I/O delay for the central processing unit.

•Figure 5. Data sets used by the ALGOL compiler.

The primary quantity specified in the SPACE parameter of the DD statements for SYSUT1, SYSUT2 and SYSUT3 must be large enough to contain the entire data set. The use of a secondary quantity for any of these data sets will increase the need for main storage by 40%. The following estimates can be used to allocate space on a 2311 direct access device:

SYSUT1	-	1 track per 100 source cards
SYSUT2	-	1 track per 100 source cards
SYSUT3	-	1 track per 200 source cards.

Processing of all data sets by the compiler is independent of the I/O device used except for the intermediate work data sets. These require magnetic tape or direct access devices.

Linkage Editing

Invoking Statement

The linkage editor is usually invoked with an EXEC statement of the form:

```
//stepname  EXEC  PGM=IEWL
```

where "stepname" is the name assigned to the job step (optional).

Other EXEC statement parameters may be included if required (see Appendix E). IEWL specifies the highest-level linkage editor in the installation's operating system.

(A method of dynamically invoking the linkage editor within a job step, by means of the CALL, LINK, XCTL or ATTACH instructions, is described in Section 4.)

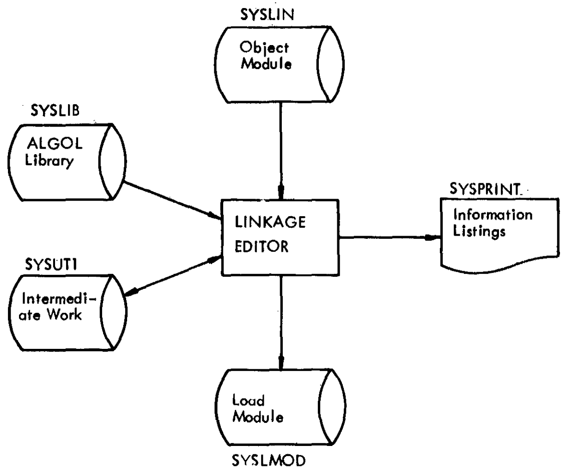
Main storage size in bytes at which changes occur	Maximum blocking factor SYSIN	SYSPRINT	SYSLIN	SYSPUNCH
45056 (44K)	5	5	5	1
51200 (50K)	5	5	5	5
59392 (58K)	5	5	5	5
67584 (66K)	5	5	5	5
77824 (76K)	5	5	5	5
90112 (88K)	20	20	40	20
104448 (102K)	20	20	40	20
120832 (118K)	20	20	40	20
139264 (136K)	20	20	40	20
159744 (156K)	20	20	40	20
184320 (180K)	40	40	40	40
212992 (208K)	40	40	40	40

Figure 6. Effect on compiler data sets if more than 44K bytes of main storage is available. The capacity of internal tables in the compiler is increased at each of the main storage sizes listed in this table, allowing, for example, a larger number of identifiers to be included in the source program. Therefore to get optimum performance, the user is recommended to use this list when specifying main storage size available to the compiler.

Data Sets Used

The data sets used by the linkage editor (see Figures 7 and 8) must be defined by the programmer with suitable DD statements.

Blocksize DCB information may be specified by the user for SYSLIN and SYSPRINT if the F level linkage editor is being used. Maximum blocking factor is 5 when 44K bytes of main storage size is available, and 40 when 88K bytes is available. Record length is fixed at 80 bytes for SYSLIN and 121 bytes for SYSPRINT.



•Figure 7. Flowchart showing data sets used by the linkage editor.

Load Module Execution

Invoking Statement

The usual method of invoking the load module generated by the linkage editor is with an EXEC statement of the form:

//stepname EXEC PGM=member-name

Purpose	Standard ddname	Devices used
For object module input	SYSLIN	Direct access or magnetic tape
For load module output, stored as a member of a partitioned data set	SYSLMOD	Direct access
For ALGOL library, SYS1.ALGLIB. A partitioned data set containing routines in load module form	SYSLIB	Direct access
For linkage editing listings	SYSPRINT	Printer*
For intermediate linkage editor working	SYSUT1	Direct access or magnetic tape

* Some form of intermediate storage, such as magnetic tape, may be used to reduce output delay for the central processing unit.

•Figure 8. Data sets used by the linkage editor.

where "stepname" is the name assigned to the job step (optional).

"member-name" indicates the name of the partitioned data set member which contains the load module. This name is specified by the programmer in the SYSLMOD DD statement for the linkage editor. Other EXEC statement parameters may be included if required (see Appendix E).

(A method of dynamically invoking the load module within a job step, by means of the CALL, LINK, XCTL or ATTACH macro-instructions is described in Section 4.)

Data Sets Used

Up to 16 data sets for use at execution time may be specified by the programmer in the ALGOL source program by using the appropriate data set number. The numbers used and the corresponding names of their DD statements are listed below.

Data set number used in ALGOL source program	Corresponding ddname
0	SYSIN
1	ALGLDD01
2	ALGLDD02
3	ALGLDD03
4	ALGLDD04
5	ALGLDD05
6	ALGLDD06
7	ALGLDD07
8	ALGLDD08
9	ALGLDD09
10	ALGLDD10
11	ALGLDD11
12	ALGLDD12
13	ALGLDD13
14	ALGLDD14
15	ALGLDD15

Any reference to a data set number by an I/O procedure within an ALGOL source program is translated into a reference to a data control block using the corresponding ddname. It is the responsibility of the programmer to supply the DD statements which correspond to the data set numbers used in the ALGOL source program.

The execution time data sets are illustrated in Figure 9 and described in Figure 10. For ALGLDD02 to ALGLDD15, case 1 in the column showing device used, applies if the source program contains any of the following:

- A backward repositioning specification by the procedures SYSACT4 or SYSACT13 for this data set.
- Both input and output procedure statements for this data set.
- Procedure statements which prevent the compiler from recognizing whether either of these applies; for example, if the data set number or SYSACT function number is not an integer constant or if a precompiled procedure is used.

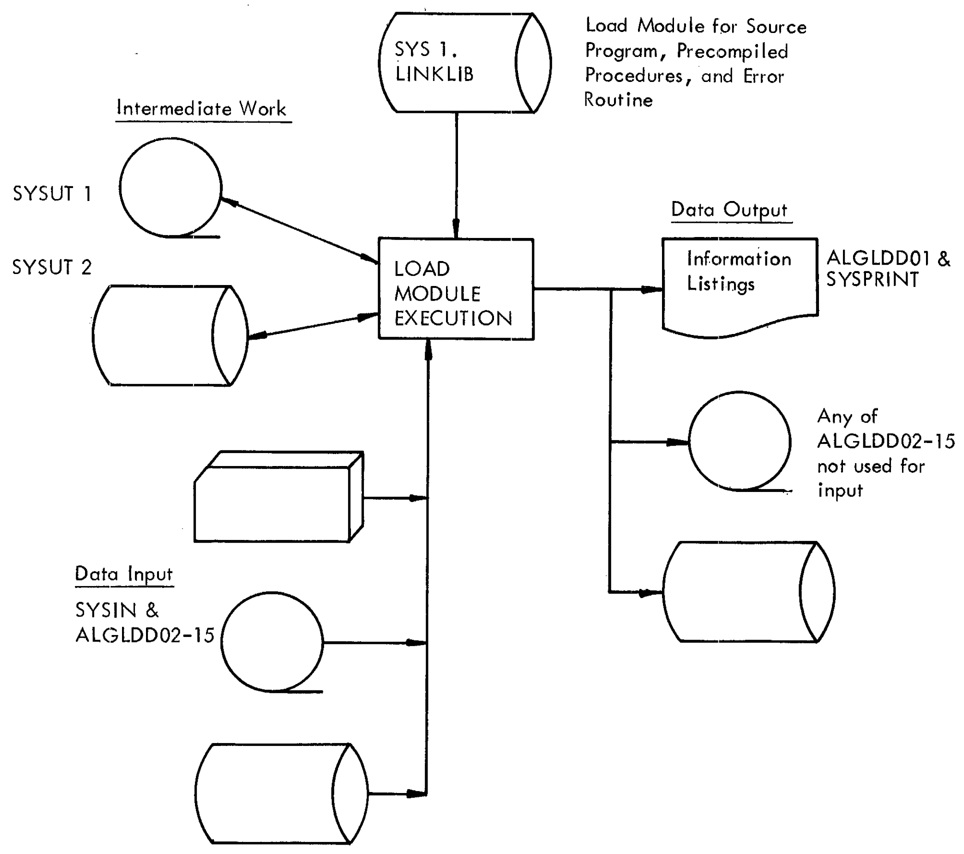
If the source program has already been compiled and linkage edited in a previous job, then the data set on which it has been stored (in load module form) must be concatenated to SYS1. LINKLIB. Data sets containing precompiled procedures called by the source program (see Section 4) must also be concatenated to SYS1. LINKLIB.

If the programmer specifies a TRACE, TRBEG or TREND option in the EXEC statement of the execution job step, the semicolon count (see Section 3) is stored intermediately on a data set with the ddname SYSUT1. The programmer must supply a corresponding DD statement if he uses this option. The semicolon count is converted to external form and transferred to the SYSPRINT data set as soon as the execution ends either by reaching the logical end of the source program or due to an error.

The space required for the semicolon count is:

For the main heading	6 bytes
For each semicolon	2 bytes
For each call of a precompiled procedure	12 bytes
For each physical record on SYSUT1	4 - 6 bytes

System/360 ALGOL permits data to be temporarily stored on and retrieved from external devices without conversion, using the ALGOL I/O procedures PUT and GET. If the programmer uses this facility in his source program, then he must supply a DD statement with the ddname SYSUT2. The device specified by this statement for storing such intermediate data should be a direct access device to guarantee reasonable performance, though programming is performed independently between magnetic tape and direct access devices. All data passed by a single PUT is



•Figure 9. Flowchart showing data sets used at load module execution. The data input and output requirements are variable.

stored as one record. This record will be as long as the data passed, plus 8 bytes. The maximum record length accepted is 2048 bytes.

The DCB information which may be specified by the user for execution time data sets is block-size, record format and record length (see page 44 for details), except for the trace and PUT/GET data sets (ddnames SYSUT1 and SYSUT2) for which only blocksize may be specified (up to a maximum of 2048 bytes).

For information not provided, default values will be inserted by a routine in the ALGOL library. In particular, blocksize is assumed as 2048 bytes for SYSUT1 and SYSUT2 if none is specified.

	Standard ddname	Device Used
For data input to load module	SYSIN	Any input device
For execution time listings	SYSPRINT	Printer*
For data output	ALGLDD01	Printer*
For data input or output	ALGLDD02 : ALGLDD15	1. Direct access or magnetic tape 2. Any
For intermediate storage of semicolon counter when TRACE is specified	SYSUT1	Direct access or magnetic tape
For temporary storage when PUT is specified	SYSUT2	Direct access or magnetic tape

* Some form of intermediate storage, such as magnetic tape, may be used to reduce I/O delay for the central processing unit.

•Figure 10. Data sets used at execution time.

SECTION 3: INFORMATION LISTINGS

To assist the programmer to find the cause of any faults in the processing or execution of his program, various forms of information listings are produced for the compilation, linkage editing and execution operations. Some of these listings are optional. Examples are illustrated in Figures 11 to 16.

CONTROL PROGRAM LISTINGS

All three operations may produce listings generated by the control program. These are described in IBM System/360 Operating System: Messages, Completion Codes, and Storage Dumps. The ABEND macro-instruction for specifying the main storage dump is described in IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions.

COMPILATION LISTINGS

A successful compilation of an ALGOL source program produces the following information listings:

- Job control statement information according to which MSGLEVEL option was specified in the JOB statement.
- The source program supplemented by a count of the semicolons occurring in the program (optional).
- A table giving details of all identifiers used in the program (optional).
- Any warning diagnostic messages.
- Information on main storage requirements at execution time.

If a serious diagnostic message is produced (meaning that object module generation has ended) then the source program and identifier table listings will be printed in full if they have been requested, but the information on main storage requirements will not be printed. If a terminating diagnostic message is produced then the source program and identifier table listings can be printed only as far as they have been produced.

Source Program

If the SOURCE option has been specified, the source program is transferred by the compiler to an output data set in order to be listed by a printer. This source program is supplemented by a semicolon count, which is referred to in the diagnostic messages to help localize errors.

The compiler generates this semicolon count when scanning the source program, by counting all semicolons occurring in the source program outside strings, except those following the delimiter 'COMMENT'. The value of this semicolon count at the beginning of each record of the source program is printed at the left of that record. It is assigned by the compiler in order to have a clear, problem-oriented reference. Any reference to a particular semicolon number refers to the segment of source program following the specified semicolon, for example, the semicolon number 5 refers to the program segment between the fifth and sixth semicolons.

Identifier Table

If the SOURCE option has been specified, a list of all identifiers declared or specified within the source program is transferred by the compiler to the output data set for printing after the source program listing. This identifier table gives information about the characteristics and internal representation of all identifiers. The identifiers are grouped together within the identifier table according to their scopes.

All blocks and procedure declarations within the source program are numbered according to the order of occurrence of their opening delimiters 'BEGIN' or 'PROCEDURE'. Therefore, if the body of a procedure declaration is a block, then usually this block has the same number as the procedure declaration itself. These numbers are called program block numbers (even if they belong to a procedure declaration and not to a block).

Each line in the table contains entries for up to three identifiers and the line begins with the number of the program block in which the identifiers were declared or specified, the value of the semicolon count at the commencement of the program block, and the number of the immediately surrounding program block. Each identifier entry contains:

CAPACITY LIMITATIONS

In addition to those given in IBM System/360 Operating System: ALGOL Language, the following restrictions must be observed when writing an ALGOL source program:

Number of blocks and procedure declarations (NPB)	≤255
Number of for statements	≤255
Number of identifiers declared or specified in one block or procedure. F is at most twice the number of for statements occurring in that block	≤179-F for type procedures ≤180-F otherwise
Length of letter string serving as parameter delimiter	≤1024 letters when main storage size available is less than 50K, ≤2000 letters otherwise
Length of label identifer	≤1024 characters when main storage size available is less than 50K, ≤2000 characters otherwise
Length of source program	≤255K
Number of semicolons in the whole program	≤65535
Number of nested blocks, compound statements, for statements and procedure declarations	≤999
Number of labels declared or additionally generated by the compiler	≤1024

The compiler generates the following additional labels:

For each switch declaration	2
For each procedure declaration	2
For each procedure activation (including function designators)	1
For each 'THEN' and each 'ELSE'	1
For each for statement	at most L + 3 where L is the number of for list elements

Length of constant pool	≤(256 - NPB) x 4096 bytes

The requirements of components within the pool are

Integer constant	4 bytes
Real constant (SHORT)	4 bytes
Real constant (LONG)	8 bytes
String (in bytes)	2 + number of symbols of open string between the outermost string quotes

The constant pool is divided into blocks of 4096 bytes each. The first block contains the integer constants 0 to 15 (64 bytes). All strings together are restricted to fill not more than the rest of this block (4096 - 64 - 2S bytes, where S = number of strings).
No constant occurring more than once in the source program is stored twice in the same block; however, it may possibly be stored more than once in different blocks. Up to seven bytes may be left unused.

Length of data storage area for each block or procedure declaration	≤4096 bytes
Number of blank spaces serving as delimiters on I/O data sets	≤255
Number of records in a data set	≤32760

Number of records per section	≤255
Number of entries in the Note Table	≤127

(The Note Table stores information to retrieve records which may be required again later. An entry for a record is made each time the ALGOL I/O procedures PUT and SYSACT13 are executed, and each time an input operation, with backward repositioning, follows an output operation on the same data set.)

Identification number (N) used by PUT or GET	0≤N≤65535

INVOKING A PROGRAM WITHIN A JOB STEP

Any one of the four macro-instructions, CALL, LINK, XCTL or ATTACH, may be used to dynamically invoke the compiler, linkage editor and load module within a job step. This is an alternative to the more usual method of invoking a program by starting a job step with an EXEC statement. Full details of the four macro-instructions are given in IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions.

To invoke a program with the CALL macro-instruction, the program must first be loaded into main storage, using the LOAD macro-instruction. This returns, in general register 15, the entry address which is used by the CALL macro-instruction. The instructions used could be:

```
LOAD     EP=member-name

LR       15,0

CALL     (15), (option-address), VL
```

To invoke a program with one of the LINK, XCTL or ATTACH macro-instructions would need instructions such as:

```
LINK     EP=member-name,

         PARAM=(option-address), VL=1

XCTL     EP=member-name

ATTACH   EP=member-name,

         PARAM=(option-address), VL=1
```

"member-name" specifies the name of the member of a partitioned data set which contains the program required.

For the compiler, member-name=ALGOL

For the linkage editor, member-name=IEWL

For the load module, member-name is specified by the programmer in the SYSLMOD DD statement for the linkage editor.

"option-address" specifies the address of a list containing the options required by the user. An address must be given even if no options are specified. The list must begin on a half-word boundary. The first two bytes contain a number giving the number of bytes in the remainder of the list. (If no options are specified this number must be zero). The list itself contains any of the options available to the PARM parameter in an EXEC statement (see Appendix E).

When using CALL, LINK or ATTACH to invoke the compiler, other ddnames may be used in place of the standard ddnames given in Section 2 for the data sets and an alternative page number (instead of the normal 001) may be specified for the start of output listings.

If alternative ddnames are used, then in the statement invoking the compiler, "option-address" must be followed by "ddname-address" giving the address of a list containing the alternative ddnames. If alternative page numbers are used, then "page-address" giving the address of a location containing the alternative page number must be placed after "ddname-address"; though if alternative ddnames are not required "ddname-address" may be replaced by a comma.

The ddname list must begin on a half-word boundary. The first two bytes contain a number giving the number of bytes in the remainder of the list. The list itself contains up to ten 8-byte fields, separated by commas, for specifying alternative ddnames for the data sets. As only seven data sets are used by the compiler, three of the fields are left blank. The alternative ddnames must be listed in the following order:

Purpose of data set	Standard ddname
Output of object module for linkage editor	SYSLIN

-- Three blank fields --

Source program input	SYSIN
Information listings	SYSPRINT
Output of object module for card deck	SYSPUNCH
Intermediate work	SYSUT1
Intermediate work	SYSUT2
Intermediate work	SYSUT3

The field for a data set which does not use an alternative ddname must be left blank if there is an alternative ddname following. Otherwise the field is omitted.

The location containing the page number must begin on a half-word boundary. The first two bytes contain a number giving the number of bytes in the remainder of the location (namely, four). These four bytes contain the number for the first page of the output listings, and on return to the invoking program they will contain the number of the last page.

An example of an invoking statement and the associated lists, for the compiler, is:

```
COMPILE   LINK   EP=ALGOL,PARAM=
                 (OPTIONS,DDNAMES,PAGE),
                 VL=1

OPTIONS   DC     H'25',C'PROCEDURE,DECK,
                 SIZE=90112'

DDNAMES   DC     H'35',C'OUTPUTbb,3CL8'b',
                 C'INPUTbbb',CL8'b',
                 C'CARDDECK'

PAGE      DC     H'04',F'62'

          b = blank
```

In this case, the PROCEDURE and DECK options are specified and 88K bytes of main storage are made available. Alternative ddnames are specified for SYSLIN, SYSIN and SYSPUNCH, and 62 is specified as the first page number for the output listings.

PRECOMPILED PROCEDURES

An ALGOL program may invoke one or more subprograms, written in the ALGOL language or in the Assembler language and stored on a partitioned data set in load module form. Subprograms of this type are known as precompiled procedures.

A precompiled procedure to be invoked by an ALGOL program must be nominally declared in the calling program. The declaration consists of a normal procedure heading, followed by the delimiter 'CODE' representing the procedure body. The name of the precompiled procedure declared in the calling program must be the load module name of the precompiled procedure.

A precompiled procedure is loaded into main storage when control passes to the program block in which the precompiled procedure is declared, and is deleted when control leaves that block. Where possible, a precompiled procedure should be nominally declared in the outermost block of the calling ALGOL program. The declaration of a precompiled procedure in another precompiled procedure which is frequently invoked, should be avoided. This saves execution time by reducing the number of loadings of the precompiled procedure.

The precision of real values must be the same, SHORT or LONG, in the calling ALGOL program and the precompiled procedure. If the installation allows multiprogramming, the REUS option (Appendix E) may not be specified for the precompiled procedure load module, in the statement invoking the linkage editor.

ALGOL Language Procedures

A precompiled procedure written in the ALGOL language must satisfy the rules, as stated in IBM System/360 Operating System: ALGOL Language, governing any normal procedure declaration. That is to say, the source module should comprise a procedure heading and a procedure body. The source module should not be enclosed by the delimiters 'BEGIN' and 'END'.

An ALGOL procedure to be invoked in a later program must be compiled, linkage edited and stored on a partitioned data set. In the invoking statement, the source module must be identified as a precompiled procedure by specifying the option PROCEDURE.

An example of the job control statements needed to compile and linkage edit a precompiled procedure is provided in Figure 32. Figure 33 illustrates the job control statements needed to compile, linkage edit and execute an ALGOL program in which a precompiled procedure is called.

Assembler Language Procedures

A sample Assembler language procedure, and an ALGOL program in which the procedure is nominally declared and called, are shown in Figure 17.1. Figure 33 contains an example of the job control statements needed to compile, linkage edit and execute an ALGOL program in which a precompiled procedure is called.

In writing an Assembler language procedure, certain rules must be observed. These rules are outlined below under the headings Entry and Start, Definitions, Register Use, Parameter Handling, and Termination.

In the instructions given below the programmer may specify any valid names in the name fields, provided the appropriate name is used in all references.

Entry and Start

The entry point of the module must be defined as follows (the names shown are examples only):

```
ENTRY     DC     A(PBTAB, 0, PARMDEF)
```

where 'ENTRY' is the location specified in the END statement; 'PBTAB' references a Program Block Table (see "Definitions", item 1); 0 represents a dummy label; and PARMDEF references a list of two-byte parameter definition constants or characteristics (Figure 17), as follows:

```
PARMDEF   DC     XL2'characteristic 1'
          DC     XL2'characteristic 2'
                 .
                 .
                 .
          DC     XL2'characteristic n'
          (First instruction executed)
```

The list must include a characteristic for each formal parameter and must be followed by the first instruction to be executed in the module. If the procedure has no parameters, PARMDEF must reference the initial instruction.

Definitions

The following data must be defined in the Assembler language procedure.

1. A 16-byte table, called the Program Block Table, must be defined:

```
PBTAB     DS     F
          DC     CL4 '(proc. name)'
          DS     F
          DC     H '(DSA length)'
          DC     X '04' ['08' if type-
                      procedure]
          DC     X '0p' p = no. of formal
                      parameters
```

"proc. name" represents the first four characters of the module name. "DSA length" represents the length of the procedure's data storage area. The length is 24 (+8 if the procedure is type-qualified), + 8 x number of formal parameters. The Program Block Table must be addressed by an address constant at the procedure entry point (see "Entry and Start") and should preferably be defined at the base address of the procedure (see "Register Use", item 4).

2. Certain registers used in communicating with Fixed Storage Area routines must be symbolically named (see "Register Use", item 1).

3. The following symbolic displacement values must be defined for those Fixed Storage Area routines which are invoked in the procedure:

```
CAP1      EQU    X'0D4'
CAP2      EQU    X'0D8'
PROLOGFP  EQU    X'0DC'
RETPROG   EQU    X'0E4'
EPILOGP   EQU    X'0E8'
CSWE1     EQU    X'0F4'
VALUCALL  EQU    X'118'
```

See "Parameter Handling" and "Termination".

4. A list of parameter definition constants, identifying the character of the formal parameters, if any, must be defined. See "Entry and Start" and Figure 17.

5. An address constant containing the address of the Program Block Table (item 1 above) and a parameter definition list, must be defined at the load module entry point.

Register Use

The standard IBM linkage conventions are not implemented in any code generated by the compiler involving a transfer of control between an ALGOL load module and a submodule. For this reason, provision must be made in a submodule to insure that externally used registers to be used internally are, at entry, saved in a local save area (and reloaded before exit), and that, where necessary, internally used registers are saved in advance of every parameter call.

All general purpose and floating point registers may be freely used in an Assembler language procedure, subject to the restrictions itemized below.

1. In the code sequences for calling actual parameters (see "Parameter Handling"), registers 8, 10, 11, 13, 14 and 15 are symbolically referenced. Every register so referenced in a calling sequence within the precompiled procedure must be defined as follows:

```
ADR      EQU   8
CDSA     EQU  10
PBT      EQU  11
FSA      EQU  13
STH      EQU  14
BRR      EQU  15
```

2. During every call for an actual parameter and before final exit from the precompiled procedure, registers CDSA (10), PBT (11) and FSA (13) must contain their values at entry to the procedure. At entry, CDSA addresses the Assembler language procedure's data storage area; PBT addresses the Program Block Table (see "Definitions", item 1); and FSA addresses the Fixed Storage Area. If any of these registers are used internally, other than in actual parameter calls, their contents must be saved in a local save area at entry to the procedure, and must be reloaded before all parameter calls and before final exit.

3. Before every call for an actual parameter, the contents of all internally used registers, required after the parameter call, should be saved in a local save area and reloaded on return.

4. All registers except registers 10, 11 and 13 are subject to varying use during a parameter call. The programmer is advised to use register 11 as base register and to specify the Program Block Table ("Definitions", item 1) in the USING statement, as illustrated in Figure 17.1. This insures that the base register is always correctly loaded before return to the procedure.

Parameter Handling

A call for an actual parameter must be implemented by means of an appropriate calling sequence, which depends on the character of the parameter and on whether it is called by name or by value.

In the instructions given below, the notation "displ" represents the displacement of a field reserved for the formal parameter in the precompiled procedure's data storage area. The displacement of the storage field of the nth formal parameter is

$24 + 8\ (n-1)$, except in the case of a type procedure where it is $32 + 8\ (n-1)$.

Important Note: Before every call for an actual parameter, all locally used registers should be saved and registers CDSA, PBT and FSA should contain their original values at entry to the precompiled procedure (see "Register Use"). On return from a parameter call, locally used registers should be reloaded.

Call by Name

1. Formal parameter specified 'ARRAY', 'STRING' or type 'REAL', 'INTEGER' or 'BOOLEAN':

```
BAL    BRR, CAP1 (FSA)
DC     H'8'
DS     H
L      ADR, displ (CDSA)
```

On return, register ADR addresses the actual parameter value or string or the actual array's storage mapping function. The storage mapping function describes the storage layout of the array. Bytes 8 to 11 contain the address of the first element in the array. The array elements are arranged in ascending order, a given subscript being regarded as a unit of the subscript position immediately to the left. For example, if an array is declared A(/1:2, 1:2), the elements are arranged as follows: A(/1,1/), A(/1,2/), A(/2,1/), A(/2,2/).

2. Formal parameter specified 'LABEL':

```
BAL    BRR, CAP1 (FSA)
DC     H'8'
DS     H
L      ADR, displ (CDSA)
B      RETPROG (FSA)
```

The sequence causes an unconditional branch to the labelled statement in the calling ALGOL program.

3. Formal parameter specified 'SWITCH':

```
        BAL    BRR, CAP1 (FSA)
        DC     H '8'
        DS     H
        L      ADR, displ (CDSA)
        LA     BRR, i [i = component
                       number]
        BAL    STH, CSWE1 (FSA)
        B      RETPROG (FSA)
```

   The sequence causes an unconditional branch to the labelled statement in the calling ALGOL program.

4. Formal parameter specified 'PROCEDURE' or '<type>' 'PROCEDURE' with j formal parameters:

```
        BAL    BRR, CAP1 (FSA)
        DC     H '8'
        DS     H
        L      ADR, displ (CDSA)
        BAL    BRR, PROLOGFP (FSA)
        DC     A (CODESEQ. 1)
        DC     XL2 'characteristic 1'
        DC     H 'j'
        DC     A (CODESEQ. 2)
        DC     XL2 'characteristic 2'
        DS     H
               .
               .
               .
        DC     A (CODESEQ. j)
        DC     XL2 'characteristic j'
        DS     H
```

   "Characteristic 1" represents the two-byte constant (Figure 17) which identifies the character of the first actual parameter.

   "CODESEQ. 1" represents the symbolic address of an actual parameter code sequence corresponding to the first parameter, as follows:

```
CODESEQ. 1  LA     ADR, par. add. 1
            B      CAP2 (FSA)
```

   where "par. add. 1" represents the address of the actual parameter. (If the parameter is a string, the first two bytes of the actual parameter should contain the string length +2). A similar code sequence must be included in the procedure for each of the j parameters of the procedure, and each code sequence must be addressed by an address constant, as shown above.

Execution of the calling sequence causes an actual procedure to be called.

Call by value

Formal parameter specified 'ARRAY' or type 'REAL , 'INTEGER' or 'BOOLEAN':

```
        BAL        BRR, CAP1 (FSA)
        DC         H '8'
        DS         H
        L          ADR, displ (CDSA)
        BAL        BRR, VALUCALL (FSA)
        DC         H 'displ'
        DC         CL2 'characteristic'
```

"displ" represents the displacement of the formal parameter's storage field in the data storage area: "characteristic" represents the two-byte characteristic (Figure 17) of the formal parameter.

In the case of a type specification, the calling sequence causes the value of the actual parameter to be moved into the 8-byte field of the formal parameter. In the case of an array, the address of the array's storage mapping function is stored in the first four bytes of the formal parameter's storage field. Bytes 8 to 11 of the storage mapping function contain the address of the first element of the array.

Termination

At the close of a precompiled procedure, the following must be observed.

1. Registers CDSA, PBT and FSA must, where necessary, be reloaded with their original contents at entry to the precompiled procedure.

2. If the precompiled procedure is type-qualified, the value of the procedure must be stored at displacement 24 in the data storage area. The latter is addressed by CDSA.

3. The terminal instruction must be

```
        B        EPILOGP (FSA)
```

   This returns control to the calling ALGOL program.

Type of Parameter	Characteristic Halfword (in hexadecimal form)		Result after call of actual parameter
	When called by name	When called by value	
STRING	CB10		ADR contains address of string
REAL	C212		ADR contains address of real value
REAL		C222	DISPL in CDSA contains real value
INTEGER	C211		ADR contains address of integer value
INTEGER		C221	DISPL in CDSA contains integer value
BOOLEAN	C213		ADR contains address of Boolean value
BOOLEAN		C223	DISPL in CDSA contains Boolean value
ARRAY or REAL ARRAY	CA16		ADR contains address of SMF (see below)
		CA26	DISPL in CDSA contains address of SMF
INTEGER ARRAY	CA15		ADR contains address of SMF
INTEGER ARRAY		CA25	DISPL in CDSA contains address of SMF
BOOLEAN ARRAY	CA17		ADR contains address of SMF
BOOLEAN ARRAY		CA27	DISPL in CDSA contains address of SMF
LABEL	CA18		ADR contains address of label
LABEL		CA28	ADR contains address of label
SWITCH	CA1C		ADR contains address of switch
PROCEDURE	CAD0		If the actual procedure is parameter-less then procedure is called, otherwise ADR contains address of procedure
REAL PROCEDURE	CAD2		If the actual procedure is parameter-less then procedure is called, and ADR contains address of real value, otherwise ADR contains address of procedure
REAL PROCEDURE		C2E2	DISPL in CDSA contains real value
INTEGER PRO-CEDURE	CAD1		If the actual procedure is parameter-less then procedure is called, and ADR contains address of integer value, otherwise ADR contains address of procedure
INTEGER PRO-CEDURE		C2E1	DISPL in CDSA contains integer value
BOOLEAN PRO-CEDURE	CAD3		If the actual procedure is parameter-less then procedure is called, and ADR contains address of Boolean value, otherwise ADR contains address of procedure
BOOLEAN PRO-CEDURE		C2E3	DISPL in CDSA contains Boolean value

Figure 17. Table of parameter characteristics for an Assembler language precompiled procedure. The SMF describes the storage layout of an array. Byte 0 contains a value denoting the number of subscripts in the array. Bytes 8 to 11 contain the address of the first element in the array. Bytes 16 to 19 contain a value denoting the size of the array.

The following is an Assembler language procedure. It is declared under the name COMP (in the ALGOL program below) with the formal parameters V1, V2 and L. V1 and V2 are integers, while L is a label. COMP is called by the ALGOL program and compares V1 to V2. If V1 ⩽ V2, the constant 1 is added to V1, and control is returned to the next instruction in the calling program. If V1 > V2, control is returned to the calling program at the address specified for label L.

```
         START
*
ADR      EQU   8
CDSA     EQU   10                          MANDATORY
PBT      EQU   11                            REGISTER
FSA      EQU   13                              DEFINITICNS
BRR      EQU   15
*
REGV1    EQU   CDSA                        LOCAL
REGADV1  EQU   FSA                           REGISTER
REGV2    EQU   12                              DEFINITICNS (OPTIONAL)
*
CAP1     EQU   X'0D4'                      MANDATCRY
VALUCALL EQU   X'118'                        FIXED
EPILOGP  EQU   X'CE8'                          STCRAGE AREA
RETPROG  EQU   X'0E4'                            DEFINITICNS
*
         USING PBTAB,PBT
PBTAB    DS    F
         DC    CL4'COMP'                   PROGRAM
         DS    F                             BLCCK
         DC    H'48'                           TABLE
         DC    X'C4C3'
*
ENTRY    DC    A(PBTAB,C,PARMDEF)
*
ALSAVE   DS    2F                          SAVE AREA FCR CDSA AND FSA
USSAVE   DS    15F                           AND FOR LOCAL REGISTERS
ONE      DC    H'1'                        CONSTANT
PARMDEF  DS    0H
         DC    XL2'C211'                   CHARACTERISTIC CF V1
         DC    XL2'C221'                                     V2
         DC    XL2'CA18'                                     L
*
         ST    CDSA,ALSAVE                 SAVE CDSA
         ST    FSA,ALSAVE+4                  AND FSA
         BAL   BRR,CAP1(FSA)               CALL
         DC    H'8'                          V1
         DS    H                               BY
         L     ADR,24(CDSA)                      NAME
         LR    REGADV1,ADR
         L     REGV1,0(ADR)                LOAD V1
         STM   12,1C,USSAVE                SAVE LOCAL REGISTERS
         L     CDSA,ALSAVE                 RELCAD CDSA
         L     FSA,ALSAVE+4                  AND FSA
         BAL   BRR,CAP1(FSA)               CALL
         DC    H'8'                          V2
         DS    H                               BY
         L     ADR,32(CDSA)                      VALUE
         BAL   BRR,VALUCALL(FSA)           V2 IS CCNVERTED TO INTEGER ANC
*                                            STORED IN CSA
         DC    H'32'
         DC    XL2'C221'
         MVC   USSAVE(4),32(CDSA)          MOVE V2 TC SAVE AREA
         LM    12,1C,USSAVE                RELOAD LCCAL REGISTERS
*                                          REGV2 CONTAINS V2
         CR    REGV1,REGV2                 CCMPARE V1 TC V2
         BH    LEXIT                       V1 > V2
         AH    REGV1,ONE                   V1 ¬> V2: ADD 1 TO V1
         ST    REGV1,0(REGADV1)            STORE V1
*
         L     CDSA,ALSAVE                 RELCAC CCSA
         L     FSA,ALSAVE+4                  ANC FSA
         B     EPILOGP(FSA)                RETURN TC CALLING PRUGRAM
*
LEXIT    EQU   *
         L     CDSA,ALSAVE                 RELOAD CDSA
         L     FSA,ALSAVE+4                  AND FSA
         BAL   BRR,CAP1(FSA)               CALL
         DC    H'8'                          L
         DS    H                               BY
         L     ADR,4C(CDSA)                      NAME
         B     RETPROG(FSA)                RETURN TC CALLING PROGRAM
*
         END   ENTRY
```

The following ALGOL program reads a number from Data Set Number 0, assigns the number to the variable I, and invokes the Assembler language procedure COMP. The call to COMP includes three actual parameters: the variable I, the constant 200.5 and the label OUT. COMP compares I to 201 (200.5 converted to integer). If I ⩽ 201, COMP adds 1 to I and returns control to the next statement in the ALGOL program. COMP is then called again. The call is repeated until I > 201, at which time COMP passes control to the statement labelled OUT.

```
'BEGIN'
      'INTEGER' I;
      'PROCEDURE' COMP(V1,V2,L); 'VALUE' V2; 'INTEGER' V1,V2; 'LABEL' L;
      'CODE';
      'COMMENT' THIS NOMINALLY DECLARES THE ASSEMBLER PRCCEDURE COMP;
      ININTEGER (0,I);
CONT: COMP(I,2C0.5,OUT);
      'GOTO' CONT;
OUT:
'END'
```

•Figure 17.1. Example of an Assembler language procedure and an invoking ALGOL program.

IHIOBA		For OUTBARRAY	70
IHIOBO		For OUTBOOLEAN	400
IHIOIN		For OUTINTEGER	420
IHIOST		For OUTSTRING	300
IHIOSY		For OUTSYMBOL	290
IHIOTA		For OUTTARRAY	120
IHIPTT		For INREAL, OUTREAL, ININTEGER or OUTINTEGER	270
IHISAT	IHCSATAN	For a short precision arctangent operation (ARCTAN)	200
IHISEX	IHCSEXP	For a short precision exponential operation (EXP)	280
IHISLO	IHCSLOG	For a short precision logarithmic operation (LN)	210
IHISOR		For a short precision OUTREAL operation	810
IHISSC	IHCSSCN	For a short precision sine or cosine operation (SIN or COS)	260
IHISSQ	IHCSSQRT	For a short precision square root operation (SQRT)	170
IHISYS		For SYSACT	1890

Figure 18. Table of ALGOL library modules. All are contained in SYS1.ALGLIB except IHIERR which is in SYS1. LINKLIB. For mathematical routines, the corresponding name in the FORTRAN IV library is also given.

APPENDIX B: IBM-SUPPLIED CATALOGED PROCEDURES

The three cataloged procedures for ALGOL that were introduced in Section 2 are contained in the procedure library, SYS1.PROCLIB, of the operating system. They consist of the job control statements listed below.

The procedures may be used with any of the OS/360 job schedulers. When parameters required by a particular scheduler are encountered by another scheduler not requiring those parameters, either they are ignored or alternative parameters are substituted automatically. For example, if these procedures are used with a sequential scheduler the following parameters, which are required for the multiprogramming option with variable number of tasks (MVT), are treated as follows:

REGION=xxxxK is ignored
SYSOUT=B is interpreted as UNIT=SYSCP
DISP=SHR is interpreted as DISP=(OLD,KEEP)

Before use, these procedures should be studied with a view to modifying them for greater efficiency within the particular environment of the installation.

In installations using the MVT option of OS/360, the REGION specifications for the compilation and linkage editing steps must be altered where necessary to suit the available storage. The REGION specification for the compilation step must be at least 4K bytes greater than the storage specified in the compiler SIZE option. In the three procedures in which the Linkage Editor is invoked, a REGION of 96K has been specified for the linkage editing step. If necessary, this REGION specification may be reduced to conserve storage. The minimum REGION specifications for the various design levels of the Linkage Editor are:

Linkage Editor	REGION Specification
E15	24K
E18	26K
E44	54K

Installations using the MVT option must also insert a REGION specification for the execution step in procedure ALGOFCLG, unless the default interpretation is acceptable. The default interpretation is the size required by the system task initiator (i.e., 50K).

Installations not using the MVT option of OS/360 should remove the superfluous parameters.

In addition, the following general recommendations should be considered:

When the MVT option is used, a SPACE parameter may be required for SYSPRINT if the device is other than a printer
the PARM fields for compilation and linkage editing steps should follow installation conventions

the SPACE and UNIT parameters for temporary data sets should be modified according to installation configuration and conventions

blocking factors should be specified for output data sets

For further information on writing installation cataloged procedures, see the publication IBM System/360 Operating System, System Programmer's Guide.

• Compilation, ALGOFC

```
//ALGOL    EXEC  PGM=ALGOL,REGION=48K
//SYSPRINT DD    SYSOUT=A
//SYSPUNCH DD    SYSOUT=B
//SYSLIN   DD    DSNAME=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,                      X
//             SPACE=(360Ø,(1Ø,4)),DISP=(MOD,PASS)
//SYSUT1   DD    UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT2   DD    UNIT=SYSSQ,SEP=SYSUT1,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT3   DD    UNIT=SYSDA,SPACE=(1Ø24,(4Ø,1Ø))
```

• Compilation and Linkage Editing, ALGOFCL

```
//ALGOL     EXEC PGM=ALGOL,REGION=48K
//SYSPRINT  DD   SYSOUT=A
//SYSPUNCH  DD   SYSOUT=B
//SYSLIN    DD   DSNAME=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,                X
//             SPACE=(36ØØ,(1Ø,4)),DISP=(MOD,PASS)
//SYSUT1    DD   UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT2    DD   UNIT=SYSSQ,SEP=SYSUT1,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT3    DD   UNIT=SYSDA,SPACE=(1Ø24,(4Ø,1Ø))
//LKED      EXEC PGM=IEWL,PARM='XREF,LIST,LET',COND=(5,LT,ALGOL),        X
//             REGION=96K
//SYSPRINT  DD   SYSOUT=A
//SYSLIN    DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//          DD   DDNAME=SYSIN
//SYSLIB    DD   DSNAME=SYS1.ALGLIB,DISP=SHR
//SYSLMOD   DD   DSNAME=&GOSET(GO),UNIT=SYSDA,DISP=(MOD,PASS),           X
//             SPACE=(1Ø24,(5Ø,2Ø,1))
//SYSUT1    DD   UNIT=SYSDA,SEP=(SYSLIB,SYSLMOD),SPACE=(1Ø24,(5Ø,2Ø))
```

• Compilation, Linkage Editing and Execution, ALGOFCLG

```
//ALGOL     EXEC PGM=ALGOL,REGION=48K
//SYSPRINT  DD   SYSOUT=A
//SYSPUNCH  DD   SYSOUT=B
//SYSLIN    DD   DSNAME=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,                X
//             SPACE=(36ØØ,(1Ø,4)),DISP=(MOD,PASS)
//SYSUT1    DD   UNIT=SYSSQ,SEP=SYSPRINT,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT2    DD   UNIT=SYSSQ,SEP=SYSUT1,SPACE=(1Ø24,(5Ø,1Ø))
//SYSUT3    DD   UNIT=SYSDA,SPACE=(1Ø24,(4Ø,1Ø))
//LKED      EXEC PGM=IEWL,PARM='XREF,LIST,LET',COND=(5,LT,ALGOL),        X
//             REGION=96K
//SYSPRINT  DD   SYSOUT=A
//SYSLIN    DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//          DD   DDNAME=SYSIN
//SYSLIB    DD   DSNAME=SYS1.ALGLIB,DISP=SHR
//SYSLMOD   DD   DSNAME=&GOSET(GO),UNIT=SYSDA,DISP=(MOD,PASS),           X
//             SPACE=(1Ø24,(5Ø,2Ø,1))
//SYSUT1    DD   UNIT=SYSDA,SEP=(SYSLIB,SYSLMOD),SPACE=(1Ø24,(5Ø,2Ø))
//GO        EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,ALGOL),(5,LT,LKED))
//ALGLDDØ1  DD   SYSOUT=A
//SYSPRINT  DD   SYSOUT=A
//SYSUT1    DD   UNIT=SYSSQ,SPACE=(1Ø24,(2Ø,1Ø))
```

APPENDIX C: CARD CODES

The card deck of the source program is punched line for line from the text written on the coding sheets. The card code used can be either a 53 character set in Extended Binary Coded Decimal Interchange Code (EBCDIC), or a 46 character set in Binary Coded Decimal (BCD). This latter character set has been established as standard for ALGOL by the International Standards Organization (ISO) and Deutsche Industrie Normen (DIN). Figure 19 shows these two codes.

Characters	Card Codes	
	EBCDIC	ISO/DIN
A to Z	12-1 to 0-9	12-1 to 0-9
0 to 9	0 to 9	0 to 9
+	12-8-6	12
-	11	11
*	11-8-4	11-8-4
/	0-1	0-1
=	8-6	8-3
,	0-8-3	0-8-3
.	12-8-3	12-8-3
'	8-5	8-4
(	12-8-5	0-8-4
)	11-8-5	12-8-4
blank	no punch	no punch
<	12-8-4	
>	0-8-6	
|	12-8-7	
&	12	
¬	11-8-7	
:	8-2	
;	11-8-6	

Figure 19. Source program card codes.

AFF=ddname indicates that the data set is to use the same I/O devices as the data set specified in the DD statement named "ddname" in the same job step.

SPACE=subparameter-list
indicates the space required when a direct access device is specified in the UNIT parameter. The "subparameter-list" contains only positional subparameters. The list is:

$$\begin{Bmatrix}\text{TRK}\\ \text{CYL}\\ \text{average-record-length}\end{Bmatrix}\text{primary-quantity}$$

[secondary-quantity][directory-or-index-quantity]

$$\text{[RLSE][}\begin{Bmatrix}\text{MXIG}\\ \text{ALX}\\ \text{CONTIG}\end{Bmatrix}\text{][ROUND]}$$

The first subparameter specifies the units in which the space requirements are expressed, that is, tracks, cylinders or records (with length given in bytes).

The next subparameter specifies the space required. It has three parts (of which the second and third are optional) and is enclosed in parentheses if more than one part is specified. If the second part is omitted, then it must be substituted by a comma if the third part is included. The initial space to be allocated is given by "primary-quantity". Each time this initial space is filled, additional space is to be provided as specified by "secondary-quantity". The number of 256 byte records to be allocated for the directory of a new partitioned data set, or the number of cylinders, taken from the initial space reserved, to be allocated for the index of an indexed sequential data set, is given by "directory-or-index-quantity".

RLSE indicates that any unused space assigned to the data set is to be released.

MXIG requests that the largest single block of storage available is to be allocated to the data set.

ALX requests that extra blocks of storage (in track units) are to be allocated to the data set. As many available blocks that are equal to or greater than "primary-quantity", up to a maximum of five, will be allocated.

CONTIG specifies that the space specified by "primary-quantity" is to be in a single block.

ROUND requests that when records are used to express the space required on the direct access device, the space is to begin and end on cylinder boundaries.

DISP=subparameter-list
indicates the status of the data set and specifies its disposition at the end of the job step. The "subparameter-list" consists of the following positional subparameters:

$$\begin{Bmatrix}\underline{\text{NEW}}\\ \text{OLD}\\ \text{MOD}\\ \text{SHR}\end{Bmatrix}\left[\begin{Bmatrix}\text{DELETE}\\ \text{KEEP}\\ \text{PASS}\\ \text{CATLG}\\ \text{UNCATLG}\end{Bmatrix}\right]$$

NEW specifies that the data set is to be generated in this job step, and would be deleted at the end of the job step unless KEEP, PASS or CATLG is specified.

OLD specifies that the data set already exists, and would be kept at the end of the job step unless PASS or DELETE is specified.

MOD specifies that the data set already exists and is to be modified in this job step. If the data set cannot be found by the operating system then this parameter is equivalent to NEW.

SHR specifies that, in a multiprogramming environment, an existing data set may be used simultaneously by more than one job.

DELETE specifies that the space used by the data set (including that in the data set catalog, etc.) is to be released at the end of the job step.

KEEP specifies that the data set is to be kept at the end of the job step.

PASS specifies that the data set is to be referred to in a later step of this job, at which

time its final disposition, or a further pass, will be specified.

CATLG specifies that the data set is to be cataloged at the end of the job step. Thus KEEP is implied. The catalog structure must already exist.

UNCATLG specifies that the data set is to be deleted from the catalog at the end of the job step. KEEP is implied.

SYSOUT=subparameter-list
specifies the printing or punching operation to be used for the data set. The "subparameter-list" is:

classname [progname][number]

"classname specifies the system output class to be used. Up to 36 different classes (A to Z, 0 to 9) may be fixed by the user for his installation, according to device type, priority, destination, etc. The standard classname is A. Classes 0-9 should only be used when the other classes are insufficient.

"number" can be used to specify an installation form number to be assigned to the output.

"progname" can be used to specify the name of a user-written output routine.

For sequential scheduling, the "subparameter-list" consists of only the standard class-names A and B. SYSOUT=B is interpreted as UNIT=SYSCP.

VOLUME=subparameter-list
indicates the volume or volumes assigned to the data set. If the data set is cataloged this parameter is not necessary. The "subparameter-list" is:

Positional subparameters	[PRIVATE][RETAIN][number][value]

Keyword subparameters	SER=list-of-serial-numbers
	REF= { dsname / *. ddname / *. stepname. ddname / *. stepname. procstep. ddname }

PRIVATE specifies that the volume is to be dismounted after the job step and that other data sets will not be assigned to the volume unless a specific request is made.

RETAIN specifies that, if possible, the volume is to remain mounted until referred to in a later DD statement, or until the end of the job, whichever is first.

"number" is any number between 2 and 9999, and is used if an input or output operation, on a cataloged data set residing on more than one volume, does not start on the first volume of the data set. The number specifies the volume on which input or output is to start (for example, 3 indicates the third volume of the data set).

"value" specifies the number of volumes required by an output data set. It is not required if SER or REF is used.

SER=list-of-serial-numbers, specifies the serial numbers allocated by the user to the volumes required by the data set. These serial numbers can consist of between one and six alphameric characters.

```
     { dsname                      }
REF= { *. ddname                   }
     { *. stepname. ddname         }
     { *. stepname. procstep. ddname }
```

specifies that this data set is to use the same volume or volumes as the data set specified by one of the alternative sub-subparameter forms. If the latter data set resides on more than one tape volume, then only the last volume (as specified in the SER subparameter) can be used.

LABEL=subparameter-list
indicates the type of label or labels associated with the data set. If the data set is cataloged this parameter is not necessary. The "subparameter-list" is:

```
Positional                  { NL  }
subparameters   [number ]   { SL  }
                            { NSL }
                            { SUL }
                            { BLP }

Keyword         { EXPDT=yyddd }
subparameters   { RETPD=dddd  }
```

"number" is any number between 2 and 9999, and specifies the position of the data set on the volume (for example, 3 would indicate the third data set on the volume).

NL, SL, NSL, and SUL specify the type of label or labels to be used, that is, no labels, standard labels, non-standard labels, and standard

```
//WEATHRP JOB MSGLEVEL=Ø
//JOBLIB DD DSNAME=WTHRPR,DISP=(OLD,PASS)
//CREATE EXEC PGM=FILECR
//ALGLDDØ2 DD DSNAME=WEATHER(+1),UNIT=(HYPERT,,DEFER),                 X
//               VOLUME=(RETAIN,SER=ØØ12),DISP=(NEW,CATLG),             X
//               LABEL=(,SL,RETPD=ØØ3Ø),                                X
//               DCB=(RECFM=F,BLKSIZE=4ØØ,LRECL=8Ø)
//ALGLDDØ1 DD UNIT=PRINTER,SEP=ALGLDDØ2
//SYSPRINT DD UNIT=PRINTER,SEP=ALGLDDØ2
//SYSIN DD *      WEATHER DATA FOLLOWS
  WEATHER DATA
/*                INDICATES END OF DATA
//FORECAST EXEC PGM=FORCST
//ALGLDDØ4 DD DSNAME=WEATHER(+1),DISP=OLD
//ALGLDDØ7 DD DSNAME=WEATHER(Ø),SEP=ALGLDDØ4,DISP=OLD
//ALGLDDØ8 DD DSNAME=WEATHER(-1),DISP=OLD
//ALGLDDØ9 DD DSNAME=WEATHER(-2),DISP=OLD
//ALGLDDØ1 DD UNIT=PRINTER,SEP=(ALGLDDØ4,ALGLDDØ7)
//SYSPRINT DD UNIT=PRINTER,SEP=(ALGLDDØ4,ALGLDDØ7)
```

Figure 31. Job control statements for Example 3.

grams FILECR and FORCST are contained in a partitioned data set WTHRPR.

Explanation of coding: The job control statements used in Figure 31 specify that:

1. The job is to have control statement messages plus the relevant control statement printed on the normal system output device only if an error occurs
2. The partitioned data set WTHRPR is concatenated to the operating system library, SYS1. LINKLIB
3. The first job step executes the program FILECR
4. The output data set is
   - a new generation of the data set WEATHER
   - to use the device class HYPERT
   - to be written on volume 0012 which need not be mounted until the data set is opened, and is then to remain mounted for later use
   - to be cataloged and have standard labels
   - to be retained for 30 days
   - to have fixed length records, 80 bytes long, and a maximum block size of 400 bytes
5. The printed output is
   - to use the device class PRINTER
   - to use a separate channel to the output data set
6. The input data is included in the input stream
7. The second job step executes the program FORCST
8. The input data sets are the last four generations of WEATHER, all of which are to be kept at the end of the job step
9. The output data set is
   - to use the device class PRINTER
   - to use a separate channel to the last two generations of WEATHER

Example 4: Compiling and Linkage Editing an ALGOL Precompiled Procedure

Statement of problem: The ALGOL language procedure ADD is to be compiled, linkage edited and stored in load module form as a member on the partitioned data set PREPROC, for use in subsequent programs. An illustration of a program in which ADD is invoked is provided in Example 5.

1. The job is to have all control statements (plus control statement diagnostic messages if an error occurs) printed on the normal system output device
2. The job step is to invoke the ALGOFCL cataloged procedure to compile and linkage edit the source module, which is identified as an ALGOL precompiled procedure
3. A new partitioned data set named PREPROC is to be allocated and cataloged; the procedure ADD is to be stored on the data set as a member; and a primary allocation of 30 tracks (plus a secondary allocation of 10 tracks, if needed) and a directory of ten 256-byte records is to be assigned to the data set.

Example 5: Compiling, Linkage Editing and Executing an ALGOL Program which Invokes a Precompiled Procedure

Statement of problem: An ALGOL program in which the precompiled procedure ADD (Example 4) is invoked, is to be compiled, linkage edited and executed.

The job control statements in Figure 33 specify:

1. The job is to have all control statements (plus control statement diagnostic messages if an error occurs) printed on the normal system output device
2. The partitioned data set PREPROC, containing the precompiled procedure ADD, is to be concatenated to the operating system library, SYS1. LINKLIB
3. The job step is to invoke the ALGOFCLG cataloged procedure to compile, linkage edit and execute the ALGOL source program

```
//CODEPC  JOB   MSGLEVEL=1
//STEP    EXEC  ALGOFCL,PARM.ALGOL=PROCEDURE
//SYSIN   DD    *
                'PROCEDURE' ADD(A,B,C).,
                'REAL' A,B,C.,
                C=A+B.,
/*
//LKED.SYSLMOD DD DSNAME=PREPROC(ADD),                                  X
//              DISP=(NEW,CATLG),SPACE=(TRK,(30,10,10)),UNIT=SYSDA,      X
//              VOLUME=SER=222222
```

• Figure 32. Job control statements and source module for Example 4.

```
//MAINPG  JOB   MSGLEVEL=1
//JOBLIB  DD    DSNAME=PREPROC,DISP=OLD
//STP1    EXEC  ALGOFCLG
//SYSIN   DD    *
                'BEGIN'
                'REAL' E,F,G.,
                'PROCEDURE' ADD(A,B,C).,
                'REAL' A,B,C.,'CODE'.,
                E:=5.6.,F:=-7.8.,
                ADD(E,F,G).,
                OUTREAL(1,G)
                'END'
/*
```

• Figure 33. Job control statements and source module for Example 5.

APPENDIX F: DIAGNOSTIC MESSAGES

Each of the three operations-compilation, linkage editing and execution - may produce diagnostic messages.

COMPILER MESSAGES

The diagnostic messages that may be produced by the ALGOL compiler are given below. Each diagnostic message occupies one or more printed lines and contains:

- The message key, consisting of the letters IEX, a three digit decimal number identifying the message, and the letter I to indicate an informative message requiring no action from the operator.
- The severity code W, S or T (see below)
- The semicolon number (see Section 3). This number is sometimes omitted if the error cannot be directly related to a point in the program. The semicolon number is indicated in the list below by the sequence NNNNN
- The message text describing the error and, in the case of some W or S type errors, the modification performed on the program by the compiler. In the message text listed below the words in parentheses, together with the parentheses themselves, will be replaced in the actual message that is printed, by specific information taken from the program. The word "operator" usually refers to all delimiters defined in IBM System/360 Operating System: ALGOL Language, but an internal compiler operator may sometimes be listed. The word "operand" refers to an identifier or an expression.

The three severity codes for errors and their corresponding compiler action are as follows:

W (Warning): The program is modified internally and the compilation is continued. The modification may not make the program correct but it allows object module generation to continue. A diagnostic message is produced.

S (Serious): An attempt is made to modify the program internally, including skipping or changing parts of it. Generation of the object module is stopped, but syntax checking continues. A diagnostic message is produced.

T (Terminating): A diagnostic message is produced and the compilation is terminated.

IEX001I W NNNNN INVALID CHARACTER DELETED.

Explanation: A character not recognized by the compiler has been deleted from the program.

IEX002I W NNNNN ILLEGAL PERIOD. PERIOD DELETED.

Explanation: The character period has been used wrongly and deleted from the program. It can be used only as a decimal point, or as part of a colon or semicolon.

IEX003I W NNNNN INVALID COLON AFTER (six characters). COLON DELETED.

Explanation: The character colon has been used wrongly and has been deleted from the program. It can be used only after a label, between subscript bounds, within a parameter delimiter or as part of an assign symbol.

IEX004I T NNNNN LETTER STRING TOO LONG.

Explanation: A letter string used to supply explanatory information exceeds capacity limitations (see Section 4).

IEX005I S NNNNN IDENTIFIER BEGINS WITH INVALID CHARACTER. IDENTIFIER DELETED.

Explanation: An identifier has been deleted because it does not begin with an alphabetic character.

IEX006I T NNNNN LABEL CONTAINS TOO MANY CHARACTERS.

Explanation: A label identifier has been used whose length exceeds capacity limitations (see Section 4).

READER'S COMMENT FORM

IBM System/360 Operating System
ALGOL Programmer's Guide

Form: C33-4000-0

- How did you use this publication?

As a reference source	☐
As a classroom text	☐
As a self-study text	☐

- Based on your own experience, rate this publication . . .

As a reference source:	Very Good	Good	Fair	Poor	Very Poor
As a text:	Very Good	Good	Fair	Poor	Very Poor

- What is your occupation? ....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

C33-4000-0

YOUR COMMENTS PLEASE . . .

This SRL bulletin is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Department 813

Fold Fold

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

C33-4000-0

Printed in U.S.A. C33-4000-0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601