503 TECHNICAL MANUAL
VOLUME 2: PROGRAMMING INFORMATION
PART 1: PROGRAMMING SYSTEMS
SECTION 5: ALGOL MK. 3

The contents of this section are liable to alteration without notice

March, 1968

CONTENTS LIST

Page

Page

Page

Printed in England by
Engineering Unit, English Electric Computers Ltd.

PREFACE

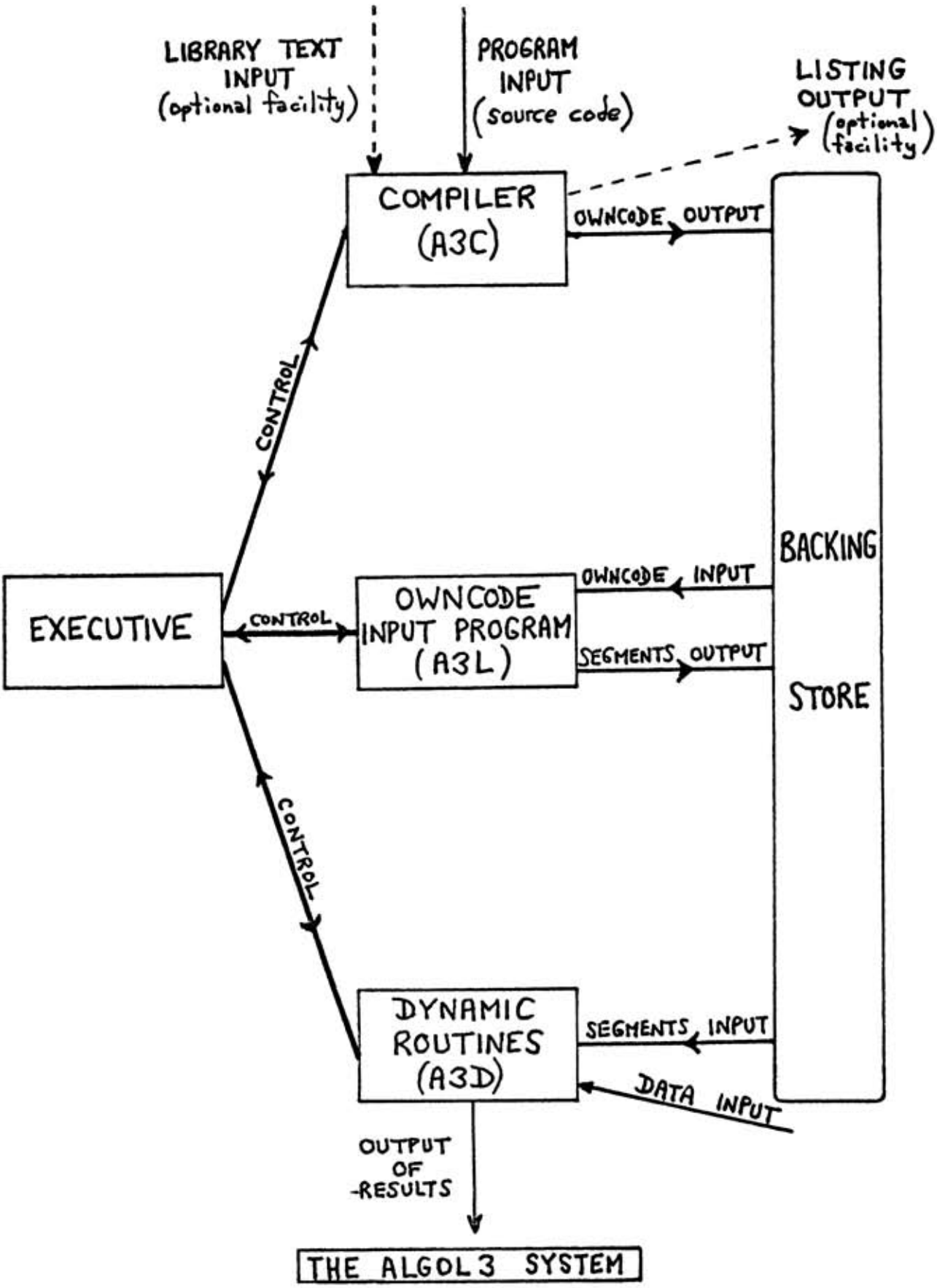
The ALGOL MK. III (Issue 2) compiler system, designed for use with a non-basic 503 equipped with some form of backing store, provides the following extensions to the original 503 ALGOL1 compiler.

1. Program Segmentation.
2. Array storage in both main store and core backing store.
3. Library facility.
4. Code statements, i.e. a combination of machine code and certain features of symbolic assembly code.
5. Improved compile-time error output.
6. Improved run-time error diagnostics.
7. Additional standard procedures.
8. Flexible source-code input and the facility to edit and/or list source-code at compile-time.
9. Adaptable and improved run-time input/output facilities, built-in lineprinter procedures and the output of boolean values.
10. Known ALGOL1 errors corrected.

The following standard procedures have been removed:-

1. dump
2. precompile.

However, a running ALGOL program may be dumped on magnetic tape using DUMP2 (see Chapter 6).


LIBRARY TEXT INPUT (optional facility)
PROGRAM INPUT (source code)
LISTING OUTPUT (optional facility)
COMPILER (A3C)
OWNCODE OUTPUT
CONTROL
BACKING STORE
EXECUTIVE
CONTROL
OWNCODE INPUT PROGRAM (A3L)
OWNCODE INPUT
SEGMENTS OUTPUT
CONTROL
DYNAMIC ROUTINES (A3D)
SEGMENTS INPUT
DATA INPUT
OUTPUT OF RESULTS
THE ALGOL 3 SYSTEM

Chapter 1: SYSTEM SPECIFICATION

1. INTRODUCTION

The ALGOL3 compiler has been designed as an easily modifiable system which gives the maximum space in main store during the compilation and running of an Algol program. The parts of the system not required are held on core backing store or magnetic tape. For this reason the system is only suitable for non-basic configurations.

ALGOL3 consists basically of three systems programs, the compiler (A3C), the owncode input program (A3L) and the dynamic routines (A3D), and an executive program (ALGOLB for CBS configurations and ALGOLM for magnetic tape configurations). At any one particular stage of the compilation and running, only the systems programs required, the executive and any necessary auxiliary SAC common program are in main store. The other systems and auxiliary programs are held on core backing store or magnetic tape.

The other important feature of ALGOL3 is the way in which the executive program controls the system and co-ordinates the use of SAC common programs for various functions. This means that by modifying the executive or any of the SAC common programs, it is not difficult for the user to adapt the system for his own particular requirements and configurations.

The SAC common programs required in the system are used as follows:-

Editing of source-code characters.	(EDIT8)	Required by A3C	
Lineprinter output for error messages and listing	(ALPL)		
Input of library texts from magnetic tape.	(LIBR1)		
Interface between auxiliary programs providing source code.	(INTER)		
Input and Output of own code and segments.	(OCBS or OAST/GMT)	Required by A3L	Required by A3D
Lineprinter output for results.	(ALP)		
Input of characters	(CHARIN)		
Output of characters	(CHAROU)		

2. PROCESS USED - COMPILATION AND RUNNING

2.1 General

As mentioned in the introduction, the ALGOL compiler has been split into 4 major programs:- an executive, A3C, A3L and A3D. A3C converts the ALGOL source code to an intermediate code called owncode, A3L converts the owncode into machine code, and A3D is required by the running translated program to provide the standard procedures and dynamic routines. To increase the ease with which a user may modify the compiler to suit his particular installation and requirements, SAC common programs are used for the input and output of data to and from A3C, A3L and A3D.

A3C uses common programs

(i) to input source code,

(ii) output error messages,

(iii) output owncode.

A3L uses common programs to:-

(i) input owncode,
(ii) output segments of a segmented program,
(iii) output error messages.

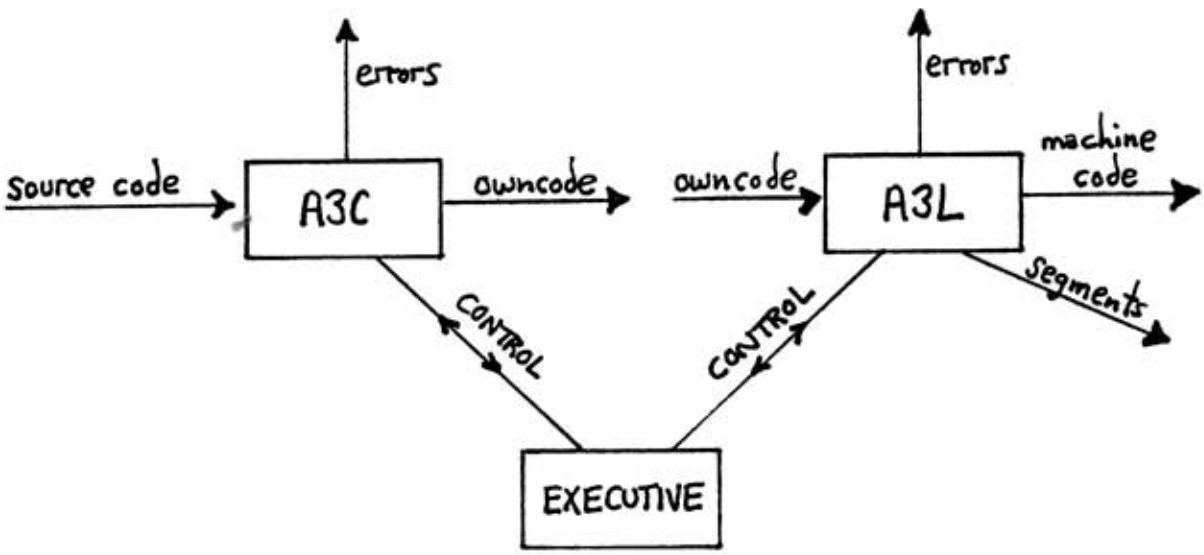
A3D uses common programs to:-

(i) input data required by the running translated program,
(ii) output results produced by the running translated program,
(iii) output error messages,
(iv) input segments of a segmented program.

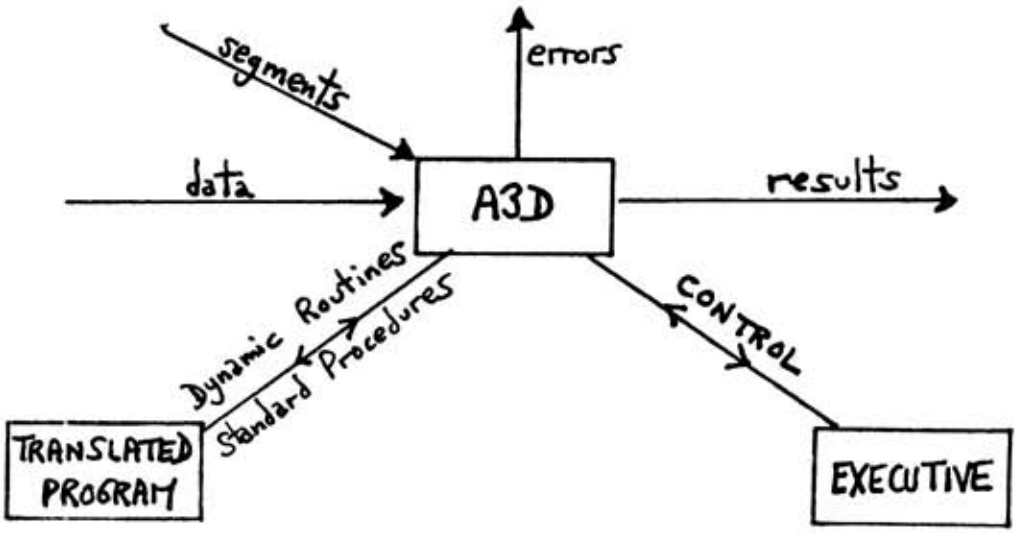
The executive is a SAC program which controls the actions of the compiler. It is the function of the executive to tell A3C, A3L and A3D which common programs are to be used and also to bring A3C, A3L and A3D plus their auxiliary programs into store at the appropriate times.

The process as it affects A3C, A3L and A3D can best be illustrated by the diagram below.

COMPILATION



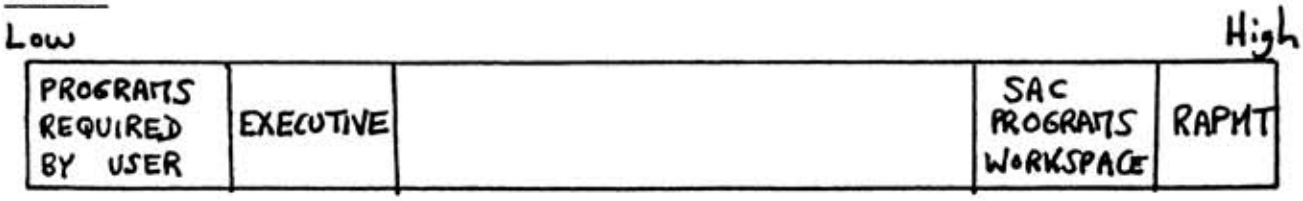
RUN - TIME



2.2 ALGOL3 in an Installation with Magnetic Tape and Core-backing Store

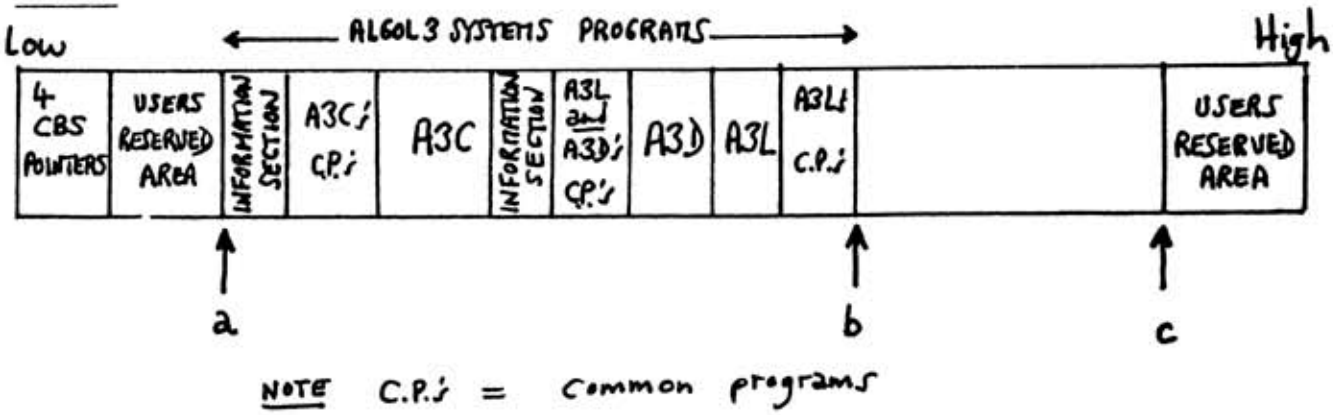
Let us assume that a magnetic tape batch holding the compiler has already been produced. When this batch is loaded into main and core-backing store, the executive (called ALGOLB) and any programs that the user wants in main store during all phases of compilation and running will be in main store. A3C, A3L, A3D and their common programs will be in the bottom of core-backing store. Thus, the main store looks like:-

FIG. 1



and core-backing store looks like:-

FIG. 2



NOTE C.P.'s = common programs

The four C. B. S. pointers are core-backing store pointers. They are:-

BSFF in location 0 — This holds the address of the current First Free location on core-backing store.

BSLF in location 1 — This holds the address of the current Last Free location on core-backing store.

BSMIN in location 2 — This holds the lowest address on core-backing store to which the user is prepared to allow the systems programs and running translated program to extend.

BSMAX in location 3 — This holds the highest address on core-backing store which the user is prepared to allow the ALGOL system to use.

By setting suitable values in BSMIN and BSMAX the user is able to reserve areas at the top and bottom of core-backing store for his own use. These values will be set when the batch is created. BSFF and BSLF are continuously updated throughout the compilation and running of the ALGOL program.

When the system is loaded (see FIG.2).

BSFF points to b

BSLF points to c

BSMIN points to a

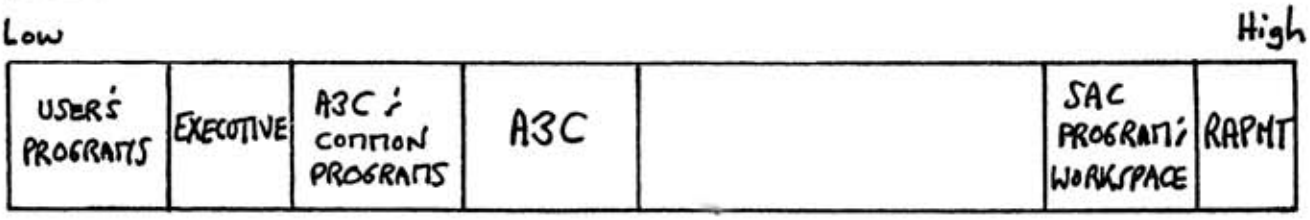
BSMAX points to c

The ALGOL3 systems programs are stored as direct copies of the programs as they existed in main store when the batch was created. Preceding A3C's auxiliary programs on CBS is a small information section holding the values that will be set in the RAP pointers (7920,7925,7926, etc.) when A3C and its auxiliary programs are brought into main store. Another information section precedes the common programs used by A3L and A3D. This will be needed when A3L and A3D with their common programs are brought into main store.

2.2.1 Conversion of Source Code to owncode (1st Pass)

When the executive (ALGOLB) is entered to start compilation, it will load main store with A3C and the common programs it requires, so that the main store looks like:-

FIG.3



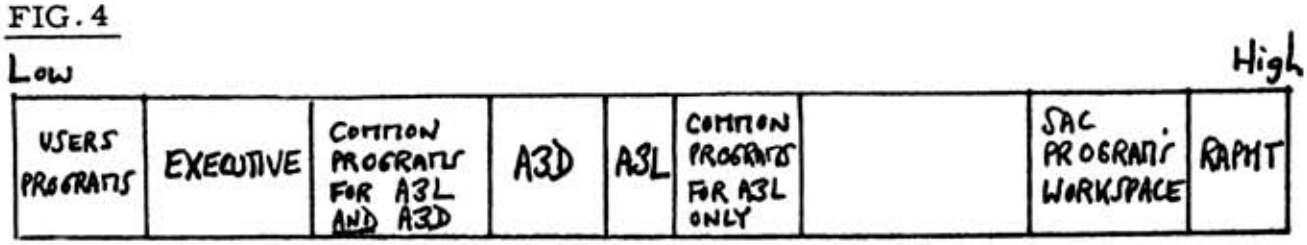
The executive will pass to A3C the names and entry points of the common programs it is to use. A3C is given the source code by one common program and converts it to an intermediate code called owncode.

This is given to a second common program which will place the owncode on core-backing store (for details see Chapter 7.3.11). When this process is complete, the executive will display the title of the program being translated, replace A3C and its common programs, with A3L, A3D and their common programs ready for the second pass.

2.2.2 Conversion of owncode to machine code (2nd Pass)

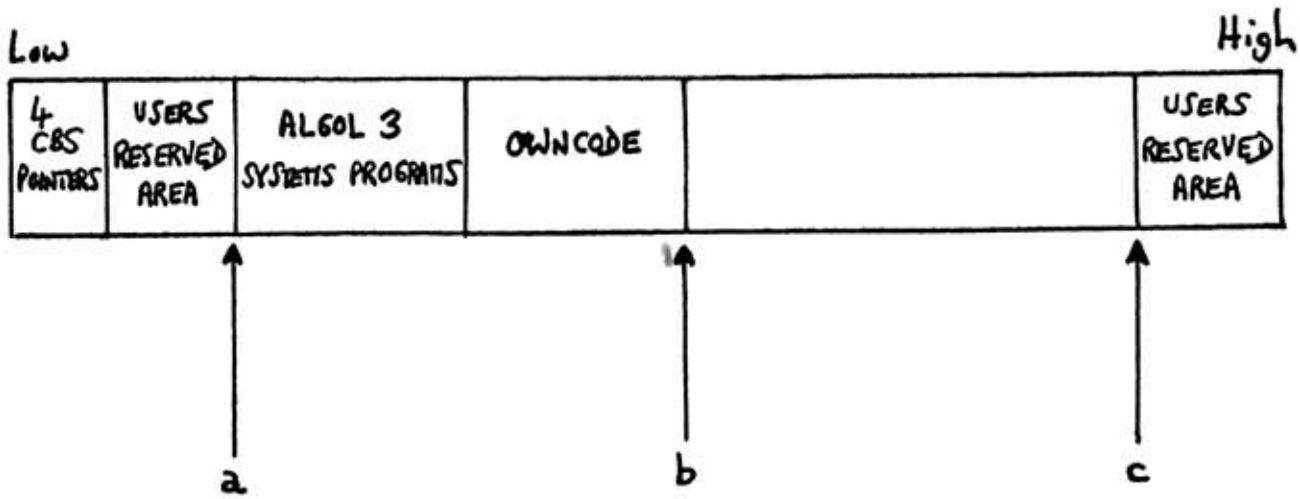
At this stage the main store looks like:-

FIG. 4



and core-backing store looks like:-

FIG. 5



2.1.5.1

The core-backing store pointers hold:-

BSFF = b

BSLF = c

BSMIN = a

BSMAX = c

A3L is the program which converts the owncode into machine code. The executive will pass to A3L the names and entry points of the common programs it is to use. A3L will be supplied with owncode by the common program OCBS and will place the machine code it produces in main store. If there are own arrays which have to be kept on core-backing store (see Chapter 4.10), A3L will suitably reduce the contents of BSLF to allow room for these arrays.

Whilst A3C was translating the source code referring to core-store own arrays, it assumed that the own arrays would be placed on core-store beginning at the address which it found in the core-store pointer BSLF and working towards the low addressed end of free store. However, A3C did not claim the backing-store space itself. The value it found in BSLF was placed in the owncode so that A3L may now check that the value has not been changed.

If the program being translated is segmented (see Chapter 4.9), then when A3L has produced a completed segment it will enter a common program (OCBS) to have the segment copied onto core-backing store. It will receive back a parameter of 19 bits which will be stored with the parameters describing the segment. These 19 bits will later be given to the program which has to bring the segment into main store when it is required by A3D. In the case of an installation at which the segments are stored on core-backing store the 19 bits would give the address of the start of the segment on core store. OCBS will place the segments at the top of backing-store.

When all the owncode has been converted to machine code the executive will delete A3L from the main store. At the same time it will delete any programs which were placed in store after A3L when the batch was created. For this reason common programs which are required only by A3L should be input after A3L. The executive will then display the available free space in main store and a "Dwait>'. This marks the end of the second pass, and completes compilation.

The user may now input programs to the main store without destroying the compiled program if he wishes.

The main store now looks like:-

FIG.6

There are three main sections to the translated program.

(i) The constants used in the ALGOL program.

(ii) The body of the compiled code.

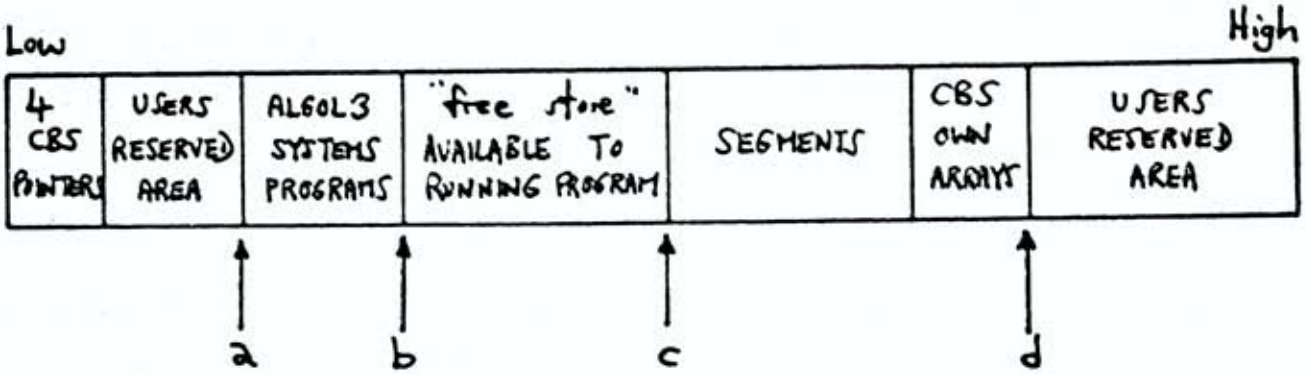
(iii) The workspace of the translated program.

The last includes 16 segment areas in which segments will be placed, own arrays which are to be kept in main store and locations assigned to the variables declared in the ALGOL program.

2.1.5.1

The core-backing store now looks like:-

FIG.7



The core-backing store pointers hold:-

BSFF = b

BSLF = c

BSMIN = a

BSMAX = d

2.2.3 Running the Translated Program

When key 19 of the word generator is changed, the executive will enter A3D to run the translated program. The executive will pass to A3D the names and entry points of the common programs it is to use.

The free store that is available in the main store and core-backing store is used to hold two dynamic stacks on which non-own arrays are placed. The user may specify an array to be kept on core-backing store or main store (see Chapter 4.10). The core-backing store dynamic stack consists solely of arrays, but information in addition to the main store arrays will sometimes be stored in the main store dynamic stack. The space required by arrays is only claimed when the block in which the array is declared is entered, and is given back when the block is left. Both stacks work from the high addressed end to the low addressed end of the free store. A3D will itself adjust the contents of 7926 so that it holds the address of the last free

location in main store, and will adjust BSLF to protect the core-backing store arrays. Should a core-backing store array require more space than is available between BSLF and BSFF, A3D will overwrite the system programs if this will give sufficient room for the array. As soon as part of the systems programs on backing-store have been overwritten, A3D will set the contents of BSMIN in BSFF. Should there still not be enough room for the program to run, a non-continuable error message will be displayed. A3D will not use the reserved area at the low-addressed end of backing-store.

A3D will enter a common program (CHARIN) when it requires data (see Chapter 7.3.4). It enters the common program with the number of the input device to be used in the accumulator. It will expect one character to be in the accumulator when return is made. A3D will allow the input device number to lie between 1 and 10 (see Chapter 3.2.1).

A3D outputs results, one character at a time, to a common program (CHAROUT). It will enter the common program with both the number of the output device to be used and the character in the accumulator (see Chapter 7.3.5) A3D will allow the output device number to lie between 1 and 524,287 (i.e. $2^{19}-1$) inclusive.

When A3D requires a segment of a segmented program to be placed in the main store, it will enter a common program (OCBS) with various parameters specifying the particular segment required. (The parameters include the 19 bits supplied to A3L by the program which wrote the segments to backing store).

Should A3D detect an error in the running program, it will either pass an error number to a common program (CHAROU) which will display the error message, in full, or it will pass the message, one character at a time, to the common program when it is not possible to use an error number, e.g. when an array name has to be displayed.

2.1.5.1

When the program has run to its conclusion, A3D will return control to the executive so that the run may be repeated or a new program compiled. It is never necessary to reload the systems programs into core-backing store from magnetic tape to repeat the run of a translated program. It will not be necessary to reload the systems programs when a new ALGOL program is to be compiled unless the systems programs on core-backing store have been destroyed, by a running ALGOL program, in which case an error message will be displayed by the executive as soon as an attempt is made to compile the new program.

2.3 ALGOL3 in an installation with Magnetic Tape only

This is essentially as described above. The differences lie basically in the executive and in the common programs that A3C, A3L and A3D use for handling owncode and segments.

The ALGOL compiler system on magnetic tape consists of a main store batch, a block containing A3C with its common programs, and a block containing A3D and A3L with their common programs. These last 2 blocks are each preceded by a nine word block holding information necessary to load the system programs into main store (e.g. the values that must be set in locations 7920, 7925 and 7926).

The main store batch needs to hold the executive (ALGOLM) and its common programs GMT and OAST. This batch must be loaded into the main store initially, so that the main store now looks like FIG. 1 except that GMT and OAST lie below the executive.

2.3.1 Conversion of source code to owncode (1st Pass)

When the executive is entered to start compilation it will bring down from magnetic tape the first information block for stage 1 and then the block holding A3C and its common programs. The main store now looks like FIG. 3 except that GMT and OAST lie below the executive.

Compilation will then proceed as described in Section 2.2.1 of this chapter with the difference that the owncode will be written in blocks to magnetic tape by the common program OAST (see Chapter 7.3.10). When this process is complete, the executive will display the title of the program being translated, replace A3C and its common programs with A3L, A3D and their common programs from magnetic tape ready for the second pass.

2.3.2 Conversion of owncode to machine code (2nd Pass)

At this stage the main store is as shown in FIG.4 except that OAST and GMT lie below the executive.

The second pass will proceed as described in Section 2.2.2 of this chapter with the differences that the common program which supplies A3L with the owncode is OAST and this program also writes the segments of a segmented program to magnetic tape. The 19 bit parameter which OAST will pass to A3L when a segment has been written to magnetic tape will be a block number.

When all the owncode has been converted to machine code, A3L will return control to the executive which will delete A3L and all programs placed in store after A3L when the ALGOL3 system batch was created. The executive will then display the extent of the free store available in main store and a "Dwait >". This marks the end of the second pass and completes compilation. The user may now input programs to the main store without destroying the compiled program if he wishes.

The main store now is as shown in FIG.6, except that OAST and GMT lie below the executive.

2.3.3 Running the Translated Program

This is as described in Section 2.2.3 of this chapter, except that there will of course, be no arrays held on backing-store and the common program which A3D enters to have a segment of a segmented program retrieved from magnetic tape is OAST.

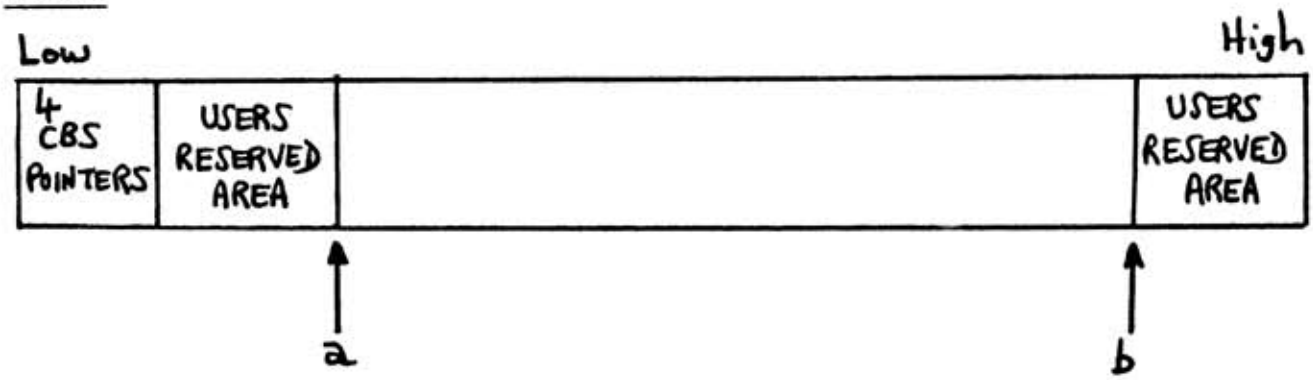
When the program has run to its conclusion, A3D will return control to the executive, so that the run may be repeated or a new program compiled. It will never be necessary to reload the main store with the batch holding the executive to compile a new program unless the user's program has corrupted the executive or any of the programs below it.

2.4 ALGOL3 in installation with Magnetic Tape and a limited amount of core-backing store

Installations of this kind may wish to keep the systems programs, owncode and segments on magnetic tape and use the core-backing store for arrays.

The executive ALGOLM is designed to allow this The values of BSMIN and BSMAX (see Section 2.2 of this chapter) must be specified when the system is created (see Chapter 6.2.1). These values will later be set in the backing store pointers when a program is to be compiled, so that backing store looks like:-

FIG. 8



The core-backing store pointers hold:-

BSFF = a

BSLF = b

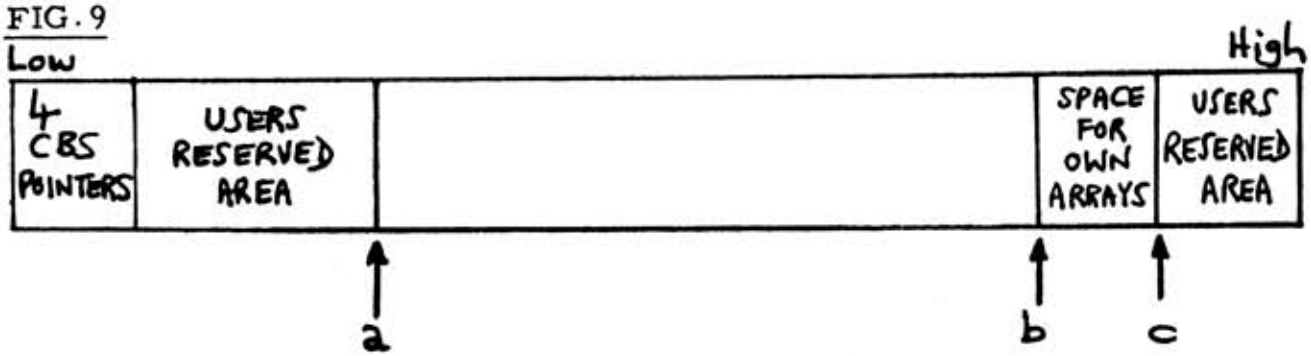
BSMIN = a

BSMAX = b

If core-backing store own arrays have been declared in the program, then during the second pass, A3L will adjust the value of BSLF to allow room for the arrays.

The backing store now looks like:-

FIG.9



The core-backing store pointers now hold:-

BSFF = a

BSLF = b

BSMIN = a

BSMAX = c

Whilst the compiled program is running, A3D will adjust the contents, of the backing store pointers (see Section 2.2.3 of this chapter) when claiming and returning the space required by the non-own core-backing store arrays.

When a new program is to be compiled the executive will restore the backing-store pointers to the values shown in FIG.8.

2.5 ALGOL3 in Installations with core-backing store only

Once the system programs have been set up on core-backing store and the executive is in main store, the description of the compilation and running of an ALGOL program is as described in Section 2.2 of this chapter. When a program is to be compiled and run, the executive must occupy exactly the same position in main store as it held at the time the systems programs were set up on core-backing store.

To reduce the time taken to input the systems programs when the ALGOL system is being created it is suggested that the program CBIT be used to connect all the programs required for the 1st pass of compilation together and all the programs required at run time and during the 2nd pass of compilation together. (See Chapter 7.3.13 for description of CBIT)

3. PROCESS USED - INPUT OF SOURCE CODE (library, edit and listing facilities)

There is a default option in A3C so that should it not be given the name of a common program, which will provide it with the source code, it will read characters directly from reader 1. However, because the source code is normally passed to A3C by a common program it is possible to have various other facilities whilst the program is being compiled. In the programs issued these consist of:-

(i) Editing the source program.

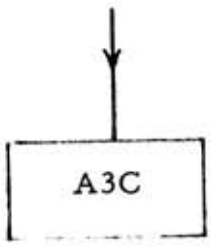
(ii) Listing the source program on the lineprinter.

(iii) A library facility, which enables ALGOL source code text held on magnetic tape to be compiled as part of the ALGOL program.

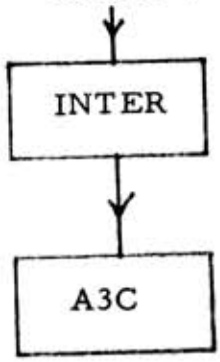
The two programs which will normally supply A3C with characters are INTER and LIBRI. INTER can either get characters directly

from reader 1 or from EDIT 8, and can, if required, pass the character to ALPL which will print the character on the lineprinter. Schematically the following alternatives exist:-

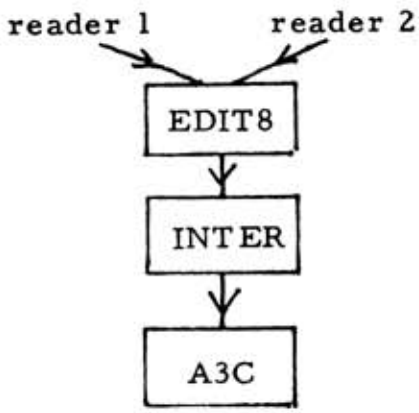
Default option



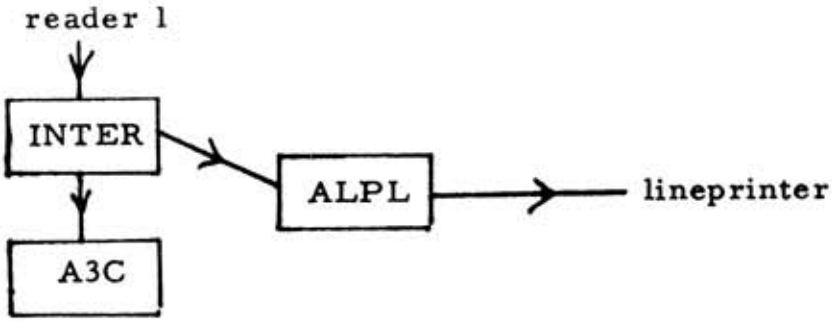
INTER



Editing facility



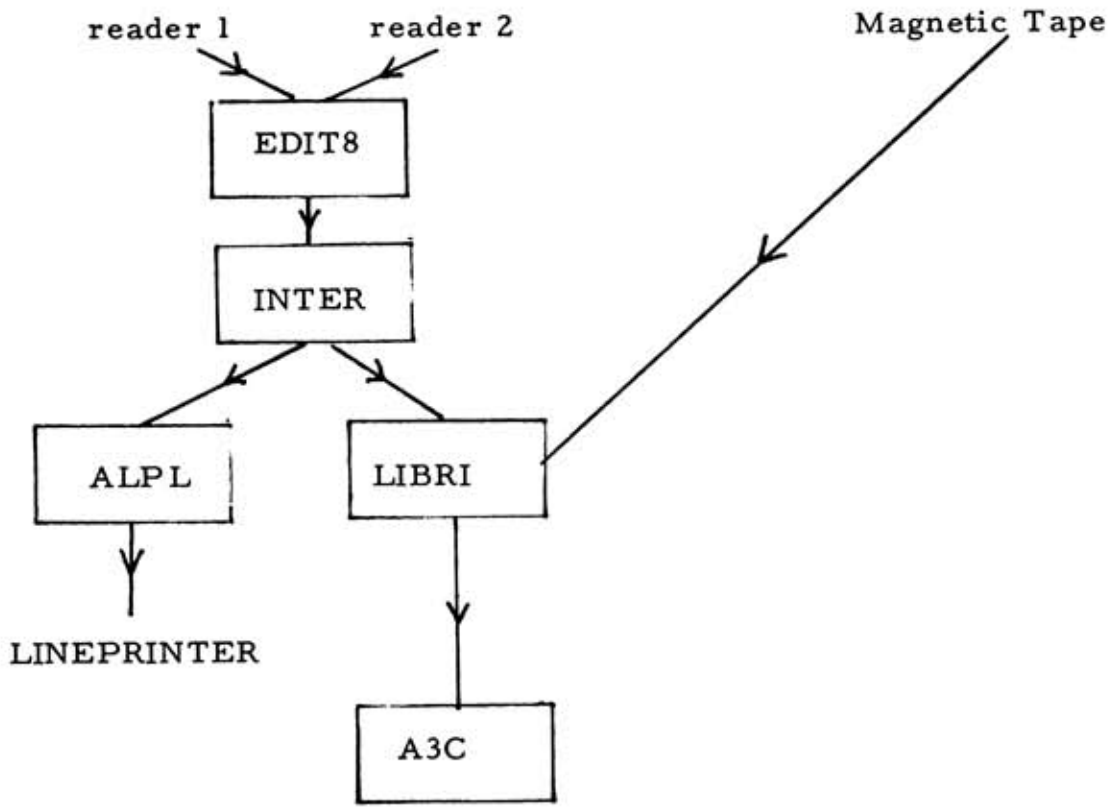
Listing facility



2.1.5.1

LIBRI can get characters from reader 1, from INTER or from magnetic tape.

By suitably combining the programs, any combination of these facilities can be obtained, e.g. library, listing and editing.



Chapter 2: ELLIOTT 503 ALGOL REPRESENTATION

1. ELLIOTT 503 ALGOL

The reference language of ALGOL requires a set of 89 printable characters; 78 of these characters are directly available in the 8-channel Elliott telecode and a further 4 can be reproduced by punching a combination of two characters. This leaves only 7 characters which are not available.

The following section gives the necessary changes which must be made in an ALGOL program before it can be presented for punching in Elliott 8-channel telecode.

1.1 Basic Symbols

503 representation	ALGOL equivalent
and	$\wedge$
or	$\vee$
not	$\neg$
£	‘
?	’
div	$\div$
*	$\times$

To give compatibility with 803 ALGOL, the following alternative representations are acceptable, but not necessary, in 503 ALGOL.

representation	ALGOL equivalent
less	$<$
gr	$>$
lesseq	$\leq$
greq	$\geq$
noteq	$\neq$
equiv	$\equiv$

The synonyms boolean for Boolean and goto for go to are also acceptable in 503 ALGOL, but ** for ↑ and @ for $_{10}$ (subscript ten) are not acceptable. Note the symbol ⊃ is not allowed (see 2.14 of this chapter).

1.1.1 The Characters halt (76) and %

The character H (halt) and % are reserved in Elliott ALGOL for special purposes connected with tape editing. They should not appear in an ALGOL statement or on a data tape, or in a comment, unless the corresponding special effect is required. In a string, % is treated in the same way as any other character, but (H) should not be used (see paragraphs 1.6 and chapter 3, paragraph 1.3).

1.2 Punching instructions

(1) An ALGOL program tape should be punched on an 8-channel Flexowriter. The punched tape should then be printed up and this print up checked against the original program script.

(2) ALGOL program should be written either on a pre-printed form (the Elliott ALGOL Program Sheet - see end of 2.1.3) or on lined paper with some vertical lines added. The heavy vertical lines represent tabulation and indicate where each line of print should commence. The tab. stops should be inserted every six character positions from the left-hand margin, which, in turn, should be set approximately $\frac{3}{4}$" in from the left-hand margin of the paper. This permits eight stops to be inserted when using a standard paper roll.

(3) Punch exactly what is necessary to produce a print-out like the written program, i.e. all blank lines, spaces, etc. Consecutive underlined words should be separated by a space. In general, the exact number of spaces, new line characters

and tabulate characters is not critical, but punching the correct number improves the general appearance of the print-up. However, between the characters £ and ?, the text must be punched exactly as written.

(4) Care must be taken to avoid confusion between the figure '1' and the letter 'ℓ' and also between the figure '0' and the letter 'O'. These must be punched correctly and punch operators should familiarise themselves with the difference in print of these characters.

(5) There should be a run out of blank tape at the beginning and end of every tape punched. Blanks can also be run out at any time.

(6) A wrong character may be cancelled by overpunching with 'Erase'. This may occur anywhere.

(7) Every semicolon should be followed by three or more blanks to simplify future editing of the tape. (The omission of these blanks is not an error.)

(8) A halt code (H) should be punched at the end of every program tape, or at the end of every physical piece of program tape if the program is punched in parts, and also at the end of every data tape.

(9) If it is impossible to punch one line of manuscript on one line of paper, a new line may be commenced between any two consecutive words, numbers or symbols. A single word must not be split.

(10) To produce an underlined word, punch the underline character before each letter of the word. Do not punch

the underline character after the last letter of the word. To produce one of the following symbols, punch the pair of characters shown opposite it:

Symbol	Characters Punched
≠	Vertical bar =
≡	Underline =
≤	Underline <
≥	Underline >

Since 'underline' and 'vertical bar' do not move the carriage, this produces the correct print-up.

(11) If goto is punched as two words then the space must also be underlined.

1.3 Notes to the programmer

(1) ALGOL programs may be written on a pre-printed form (503 ALGOL PROGRAM sheet) or on lined paper (on which a series of vertical tabulation lines have been drawn).

(2) Consecutive words (identifiers) or consecutive basic words must be separated by at least one space and manuscripts must be written with such words clearly separated.

(3) The programmer must clearly differentiate between the figure '1' and the letter 'ℓ' and also between the figure '0' and the letter 'o'. The use of the continental 1 and 7 and a script 'ℓ' is strongly recommended.

1.4 The program

Every ALGOL program written for the 503 must be preceded by a title and followed by a semicolon.

A title consists of any string of characters not including a semicolon or a halt code, and is terminated by a semicolon. This title is displayed when the program is run.

The title should be used to give enough information to identify the program and the programmer uniquely. It may include name and department of the programmer, cost code of the work being done, and the date on which the program is presented. Each installation is encouraged to establish its own standard practice for the writing of titles. Since the title is not part of the program, a comment may not come immediately after the semicolon which ends the title.

1.5 The use of Elliott ALGOL program sheets

Elliott ALGOL program sheets have been designed to enable the programmer to indicate to the punch operator the exact layout of his program. For this purpose, one and only one character should be written in each cell; a cell which does not contain a character is treated as a space if it occurs in the middle of a line; after the last occupied cell of a line a change to a new line is punched.

The cells are grouped in units of six by means of a firmer line. This is an aid to punch operators in counting the number of spaces required on a deeply indented line. It is recommended that indentation should be a multiple of six; this allows room for the tabulation of statements just to the right of a begin.

For identification purposes, the name of the program should be written at the head of each sheet; if the sheet contains a recognisable sub-section of the program, a subtitle may be used. These names at the head of the sheets are not reproduced on the punched document. Therefore the first sheet of the program must contain a copy of the title written as part of the text of the program as well as the head of the sheet.

1.6 Correction of ALGOL programs

Any errors in an ALGOL program, whether of a syntactic nature or in the actual formulation of the problem, must be corrected in the source language. There is no means of making corrections to the compiled program in the store of the computer.

To insert a statement or group of statements into a program after a given statement, S say, punch (H) (76) on the tape immediately after the semicolon that terminates S. On reading this the computer comes to a systems wait. Then translate the additional statements (the last one having (H) punched after the terminating semicolon), and finally continue translating the original program. (See also EDIT FACILITY, Chapter 4.12).

To cause the compiler to skip a statement of the program, punch % immediately after the semicolon that terminates the preceding statement. On reading % the computer ignores all subsequent input until the next semicolon.

The halt code facility also permits a program to be punched in sections on separate tapes, each section having (H) at the end.

According to the punching instructions, three or more blanks should be left after every semicolon of the program. If this has been done, it is an easy matter to insert (H) or % by a hand punch wherever necessary.

2. RESTRICTIONS AND PROGRAMMING NOTES

This section describes a number of restrictions imposed on the full generality of ALGOL, and mentions restrictions which are part of ALGOL and whose effects are frequently overlooked.

2.1 The declaration of labels

Any labels used to label a statement in the compound tail of a block must be declared at the head of that block. The method of declaring a label is to include it in the switch list of a switch declaration:

```
switch ss:= label1, label2, ..... ;
```

The switch identifier is obligatory, even if it is not used in a statement of the program. Any identifier different from the other identifiers of the program may be used for a switch; but the use of a sequence of the letter 's' is recommended as standard practice, except where this would cause a clash with other identifiers.

Care must be exercised in declaring labels in the block to which they are local. It is inadmissible to declare labels of an inner block in an embracing block. The following is a scheme of correct label declarations:

```
begin....
   switch s:= ℓ1, ℓ2, ℓ3;
      :
      :
   ℓ1:.............;
      :
      :
      :
   ℓ2;begin switch ss:= ℓ3, ℓ4;
            ........
         ℓ3:........;
         begin comment compound statement;
               ......
            ℓ4:......;
         end compound statement
      end block ℓ2;
      :
      :
ℓ3;....:..........;
      :
      :
end;
```

A label may not prefix a statement in a procedure body unless it is declared in the same procedure body. This may involve turning the statement which forms the procedure body into a block, by attaching a switch declaration, and, if necessary, a begin and end.

2.1.1 Unsigned integers as labels

Unsigned integers may not be used as labels. Where they occur in a program, they should be turned into identifiers. The recommended practice is to precede each number by a sequence of one or more 'ℓ's. For example, '23:' might become 'ℓ23:' or 'ℓℓ23:'.

2.1.2 Switches

The elements of a switch list may only be labels. These labels must be prefixed to statements of the block in the head of which the switch declaration occurs. The occurrence of a label in a switch list serves as a declaration of that label and therefore no label can occur more than once in the switch declarations of any one block.

If, in a published program, these conditions are not satisfied, the switch declaration must be replaced by a portion of program which achieves the desired effect. It should seldom be necessary to do this and no general rules are given.

If a go to statement uses a switch designator, and the subscript of the switch is non-positive, or greater than the number of labels in the switch list, the computer displays SWITCH ERROR and no further statements of the program are obeyed. If this is expected to occur, the go to statement should be turned into a suitable conditional statement. For example,

```
go to ss [i]
```

where the switch list of ss has three elements, would become:

if i > 0 and i < 4 then go to ss [i] ;

this has the same effect as that specified in the Revised ALGOL report.

2.2 Type of Arithmetic Expressions

If the type of an arithmetic expression depends upon the evaluation of an expression or upon the type or value of an actual parameter then it is taken to be real, e.g. the result of exponentiation is always real even when both arguments are integral. Provided that the result is within the range -536 870 912 to +536 870 911, no accuracy is lost as a result of this.

If the result of exponentiation appears on either of div (the integer division sign), the translator will not accept the program and will display Error no. 31 to indicate an inadmissible operation. To correct this, the standard function 'entier' should be used.

e.g. n ÷ 2 ↑ m

should be written as

n div entier(2 ↑ m)

2.3 go to statements

go to statements occurring inside a procedure body may not have as their destination a statement which is outside the procedure body. thereby causing an exit of the procedure; however, a go to statement may lead to a label which is an actual parameter of the procedure. Label parameters may not be called by value.

There are two methods of adapting a program to evade this restriction:

(i) adapt the procedure declaration and procedure calls to specify the label as a parameter

(ii) an 'error exit' may be turned into an 'error procedure' which performs the necessary printing, and then enters the standard procedure 'stop', which stops the program.

2.4 for statements

The implementation of the for list element is not exactly as described in section 4.6.4.2. of the Revised ALGOL Report, in that the address of the controlled variable is computed only once at the start of the statement, and is not re-computed each time round the loop.

2.5 own arrays

The subscript bounds of an own array must be written as integers. There are two courses of action which can be taken for the few programs which violate this rule:

(i) the widest feasible bounds should be chosen and the integer values inserted in the declaration

(ii) the array should be made into a non-own array of a block surrounding the block in which the original own array declaration was made.

The size of the integer bound, x must satisfy $-8589934591 \leqslant x \leqslant 8589934590$.

2.6 Specification of parameters

All formal parameters of a procedure must be specified in the head of the procedure.

2.7 Recursive procedures

A recursive procedure may not have any real, integer or Boolean parameters called by name. However, a procedure may be entered recursively during the evaluation of any of its parameters, whether called by name or called by value.

2.8 switch parameters

A formal parameter of a procedure may not be specified as a switch; consequently, a switch identifier may not appear as the actual parameter of a procedure, though a switch element may do so.

The easiest method of evading this restriction is to use an output name parameter to specify the subscript of the switch.

2.9 Procedures as parameters of procedures

A formal parameter of a procedure may not be specified as a procedure. Consequently no procedure identifier, alone without parameters, may appear as an actual parameter, unless it is a parameterless procedure which stands as an expression.

This restriction may be evaded by use of name parameters.

2.10 Type procedures

A call of a type procedure can only occur in an expression: it cannot stand alone as a procedure statement.

2.11 Sequence of declarations

Any number of declarations may occur in any order in a block head, but other declarations may not occur after a procedure declaration. All procedure declarations of a block must be grouped together at the end of the block head.

2.1.5.2

A procedure body may contain activations of any other procedure declared previously in the same block head. Any set of procedure declarations may be put into sequence in such a way that this rule is satisfied, except in the case of mutual recursion.

2.12 Length of identifiers

Identifiers are distinguished by their first six characters; any subsequent characters are ignored. If an identifier coincides over its first six characters with any other concurrently used identifier, its name must be changed. No general recommendation can be made on a method of systematic alteration; it is advisable to restore uniqueness by making a reasonable abbreviation of the word. For example, the identifiers 'condensation' and 'condenser' appearing in the same program might be abbreviated to 'condsn' and 'condsr'.

2.13 Reserved identifiers

The identifiers 'checkr', 'checki','checkb', 'checkB', 'checks', 'location' and 'elliott' are reserved for special purposes and may not be used in any other way (see Chapter 4).

2.14 The Operator ⊃

The Boolean operator ⊃ (denoting implication) is not allowed in Elliott ALGOL. The Boolean expression

A ⊃ B

may always be replaced by the equivalent expression

B or not A

2.15 Range and accuracy of numbers

In all cases where an operation is defined as having a result of type integer, this result must be in the range

$$-274\,877\,906\,944 \text{ to } +274\,877\,906\,943$$

$$-2^{38} \text{ to } 2^{38} - 1$$

If an integer exceeds these limits, the program stops and the message INTOFLO is displayed. In an arithmetic expression a constant is treated as a positive number with an associated sign. Thus, in the example,

```
begin integer x;
        x := -274877906944 ;
```

ENTIER ERROR is displayed because 274877906944 is stored as a real number, negated and converted to an integer.

In all cases where an operation is defined as having a result of type real, this result must be in the range

$$-5.79 \times 10^{76} \text{ to } +5.79 \times 10^{76} \text{ approx.}$$

$$\text{i.e. } -2^{255} \text{ to } 2^{255} - 2^{226}$$

If a real number exceeds this limit during the running of a program the computer stops and ERRINT 1 is displayed. During translation, a real constant that is too large is replaced by the largest possible real constant ($\simeq 5.79 \times 10^{76}$).

Where any operation (including reading and printing) is defined as having a result of type real, this result may be inaccurate by up to two parts in 10^9. It is not recommended to expect an accuracy greater than eight significant decimal digits in testing convergence of a numerical process.

If a number with exponent greater than 12 is read, a slightly greater inaccuracy may be expected. In particular, if two such numbers a and b are read, and a is greater than b by an amount which is too small to be represented, then the machine representation may be such that b is greater than a.

2.16 The standard function abs(E)

In accordance with the ALGOL 60 Report, this function operates on an argument of type real. If I and J are variables of type integer the statement I:= abs(J) involves a double type conversion, and if J lies outside the range -536 870 912 to +536 870 911 rounding occurs. This effect may be avoided by substituting for 'abs(J)'

if J less 0 then -J else J

2.17 The standard trigonometrical procedures

The arguments of sin(E), cos(E) and tan(E) are expressed in radians. If, within the limits of the representation E is an odd multiple of $\pi/2$, then tan(E) is assigned the value of the greatest representable positive number ($\triangleq 5.79 \times 10^{76}$).

The inverse functions satisfy

$-\pi/2$	arctan(E)	$\pi/2$
$-\pi/2$	arctan(E)	$\pi/2$
0	arccos(E)	π

In extreme cases the results of these operations are indistinguishable from 0, $\pm\pi/2$ and π. As a result of the finite accuracy of the representation both limiting values can occur in each case.

2.18 Sequence of operations

The order in which operations are performed within an arithmetic expression is undefined except insofar as it is determined by the rules of precedence:

(1) exponentiation
(2) multiplication and division
(3) addition and subtraction.

Thus in

a : = b + c + d

the order of evaluation of b, c and d is undefined so that if one of them is a type procedure which affects the value of one of the others (a 'sneaky' procedure), the value of a will be undefined. If it is important that b, c and d are evaluated in that order, they should be converted into expressions by the use of brackets:

a : = (b) + (c) + (d)

Similarly, if it is important that tne two additions are performed in the order shown, the expression should be written:

a : = (b + c) + d

If a, b and c are real, and are of widely differing magnitudes, then (a + b) + c may not be equal to a + (b + c).

2.19 Boolean expressions

In the evaluation of a Boolean expression, every term is evaluated. Thus, in the expression

A : = B or C or D

even if B is found to be true, and thus the value of the expression determined, C and D are nevertheless still evaluated. Hence any side effects from C and D always occur.

Remarks analogous to those of 2.17 apply to the order of operations withina Boolean expression.

2.20 Correspondence between formal and actual parameters

In most cases if the formal and actual parameters of a procedure are of different type, conversion of the actual parameter

automatically takes place. However, in certain cases the correspondence between formal and actual parameters must be exact. These are as follows:

2.20.1 Array parameters

If a formal parameter of a procedure is specified to be an array, the corresponding actual parameter must in all cases be of the same type and have the same number of dimensions.

2.20.2 Parameters called by name

If a formal parameter of a procedure is called by name and within the procedure body has a value assigned to it, then the corresponding actual parameter must be a variable of the same type. It must not be a constant or an expression.

No error indication is given during translation or running. Any attempt to assign a value to a constant or an expression at run time normally results in the loss of the assigned value.

2.21 Array dimensions

An array may only comprise 31 dimensions.

2.22 Arrays in a segment

A maximum of 31 arrays may appear in an array segment.

2.23 Multiple placings of labels

This will not be detected.

2.24 Restrictions in the use of recursive procedures

2.24.1 The following obscure case of recursion will produce incorrect results:

A procedure A which contains a procedure B which calls A must not have any declarations (other than procedure declarations) preceding a call of B.

example

```
procedure A;
    . . . .
    . . . .
    begin procedure B;
          begin
          . . . .
          . . . .
          A;                    body
          . . . .               of
          . . . .               B
          end;
    begin real p, q, r;                    body
          . . . .                          of
          . . . .                          A
          B;
          . . . .
          . . . .
    end
    end;
```

The contents of p, q, r would be lost on entry to B.

Provided B is not a type-procedure, this may be avoided by making all declarations before declaring B.

If B is a type procedure used in a complex arithmetic expression, incorrect results may also be produced even though there are no declarations after the body of B and before its use.

2.24.2 A procedure may not contain a "for" statement having a recursive call of the procedure within the body of the statement if there is more than one "for" list element.

e.g.
```
procedure wrong(n);
. . . .
. . . .
for i := 1,2,3,4 do wrong (i)
```

In this case, this could be overcome by:

```
procedure right(n);
. . . .
. . . .
for i:= 1 step 1 until 4 do right (i)
```

2.25 Alterations required to programs written for the ALGOL1 compiler

2.25.1 Significant space characters

If there is a space between the : and = of an assignment statement an error will occur.

2.25.2 Procedures 'dump' and 'precompile'

As these procedures are not available with ALGOL3, error number 20 will be displayed if programs with these procedures are input.

2.25.3 Arrays in an array segment

The maximum number of arrays permitted in an array segment with ALGOL3 is 31; more than this will give error number 16.

2.25.4 Procedure 'elliott'

There is a restriction in 'elliott' orders with ALGOL3 - see Chapter 4.7.2.1.

An alteration may be required to programs containing references to arrays inside 'elliott' procedures - see Chapter 4.7.2.1.

Chapter 3: INPUT AND OUTPUT FACILITIES

1. INTRODUCTION

ALGOL 60 provides no method of programming a computer to input the data for a calculation or to print out the results. In view of this, each implementor is obliged to adopt a system of his own.

1.1 print and read statements

The system chosen for Elliott ALGOL is based on the introduction of two new types of statement, the print statement and the read statement. The syntax of these statements is very simple, since they consist only of the underlined words print or read, followed by a list of operands, the items of the list being separated by commas, e.g.

```
read x,u,b[j];
print 2↑w, x*y,cos(x/q) - b[j];
```

The effect of a read statement is to cause one or more numbers to be read from some input device, and assigned (in the sequence read) to the variables specified in the list. The effect of a print statement is to cause the values of the arithmetic expressions occurring in the list to be output (in the order given) on an output device of the computer.

1.2 Structure of read and print lists

The elements of a read or print list may be arithmetic variables or procedures. A print list may also contain arithmetic or boolean expressions and strings. These elements are scanned in the sequence in which they are written, with the following effect:

item	effect in a read list	effect in a print list
arithmetic variable	A number is read from the current input device and its value is assigned to the named variable	The value of the variable is printed on the current output device.
number or arithmetic expression	NOT PERMITTED	As for an arithmetic variable.
non-type procedure other elliott.	The procedure body is executed	The procedure body is executed.
arithmetic procedure (inside the body of the procedure)	As for an arithmetic variable	This is taken as a recursive call of the procedure; hence it must have parameters. The value assigned to the procedure by this call is printed on the current output device.
arithmetic procedure (not inside its own body)	NOT PERMITTED	The procedure body is executed and the value of the procedure is printed on the current output device.
Boolean variable or procedure or expression	NOT PERMITTED	As for arithmetic variable expressions and procedures, i.e. prints the value true or false.
string or string parameter	NOT PERMITTED	The string is printed on the current output device.
Elliott procedure	NOT PERMITTED	NOT PERMITTED.

1.3 Input data tape

Numbers to be read by the computer should conform to the ALGOL definition of number, and each number must be followed by some character other than a digit, a decimal point or subscript ten. This terminal character may be followed by any sequence of characters excluding the digits

0 to 9 and the characters $._{10}$ £ + - . The sequence may be of any length; its only function is to separate the numbers on the data tape, and it is otherwise completely ignored. Note that spaces may not normally be used in the middle of numbers, since they are classified as terminating characters (but see 2.5.3 of this chapter).

Blank tape is ignored under all circumstances. The character 'halt' in any position on the input tape causes the computer to wait.

The character £ is not permitted on a data tape, except in connection with the 'instring' facility (see 2.6 of this chapter).

The example data tape	is read as a succession of numbers:
item 176329	176329
197 ft 7 ins	197
	7
$g = 1.276_{10} -11$	$1.276_{10} -11$
$7.392_{10} -4$	$7.392_{10} -4$
710623 1.49700	710623
	1.49700

1.4 Presumed settings

The print and read statements of the simple type explained above are executed under the control of the presumed settings; methods of changing the settings and obtaining varied and more sophisticated effects of format control is described in 2.3 of this chapter.

All numbers read under control of the presumed settings are read from tape reader number one, and all numbers output on tape punch number one.

Numbers printed under presumed settings appear on separate lines with up to eight digits. The values of integer expressions less

than 100 000 000 are printed with leading zeros replaced by spaces and, in the case of a negative number, the sign floated, i.e. the sign is moved along so that it immediately precedes the most significant digit.

E.g. - 00000252 is printed as
-252

Larger integers are printed in a form similar to that of large real numbers.

The values of real expressions, if less in absolute magnitude than 100 000 000, are printed as decimal numbers containing eight digits and preceded if necessary by a minus sign. Larger numbers are printed with an exponent part and only four significant digits.

Examples of printing under presumed settings:

Integers	Real numbers
9	-9.0000000
-127	127.30614
239610	-23961.000
-14306971	.00000000
$-2.75_{10} + 11$	$-1.364_{10} + 22$

1.5 Output of text

In order to output headings and other messages, the characters which comprise the message should be enclosed within the string quotes £ and ?, and included as an item of the list of a print statement; these items being output in the sequence in which they appear.

Examples:

```
print £ height                weight      speed?;
print F, £ ft ?,i,£ ins ?;
```

To output lines, spaces, blanks, or the symbols £ and ?, an inner string should be used which consists of certain special interpreted characters enclosed by two pairs of string quotes (£ and ?). These characters

may appear inside single string quotes amongst text which is itself enclosed in string quotes. The meanings of the special characters are given in the following table:

Character	Interpretation
ℓ	new line
s	space
r	blank (i.e. runout)
q	£ (i.e. quote)
u	? (i.e. unquote)
t	tabulate
h	stop

All the above characters will be accepted on upper case as well as lower case.

If any of these characters is followed by an unsigned integer, the effect is the same as writing the character the specified number of times.

Spaces, tabulation symbols and changes to a new line may also occur within a single pair of string quotes, and they are reproduced in the same way as other characters; care must be taken to ensure that only those characters required are quoted.

Examples:

```
print ££ℓ4?chapter?,n,££ℓ2s5??;
print ££ℓ?height £s12? weight £s12? speed?;
```

Text is output on the same device as a number in the same list. Under presumed settings, therefore, output will be on punch one. Text is not preceded by a change to a new line, and therefore is printed on the same line as the last number. A change to a new line may be specified as part of the text.

2. SETTING PROCEDURES

The presumed settings are adequate for inexperienced programmers, for program testing and low-volume output. In many applications, however, the programmer may wish to use all available input and output devices, to tabulate his results, and to exercise control of output format. For these purposes a number of procedures (setting procedures) are provided.

A setting procedure statement normally occurs in the list of a print or read statement, separated in the normal way by commas from the other operands. All numbers and text in a list subsequent to a setting procedure statement will come under the control of the settings specified. Any number occurring previously in the list or in the lists of other print statements, is totally unaffected. Thus operations in a list previous to any setting procedure statement are always executed under the control of the presumed settings (see 1.4 of this chapter).

A setting procedure statement may also occur as a separate statement of the program. In this case, once it is obeyed it will be effective until it is cancelled by a contradictory setting procedure statement. Such a cancellation may occur locally inside a print statement, or globally as a separate statement of the program.

Where a parameter of a setting procedure is of type integer, the actual parameter may be an expression, allowing the format of results to be determined dynamically.

When a procedure body is executed as a result of a procedure call inside a read or print statement, any setting procedures called globally (i.e. not inside a print or read statement) will remain effective for the remainder of the read or print statement containing the procedure call. When this particular read or print statement is completed, however, the values of

setting procedures are restored to the values they had upon entry to it.

Particular care should be taken when the procedure called contains a label parameter through which exit may be made, since the settings which were current inside the read or print statement will now be effective outside the statement until changed by further calls of the setting procedures.

e.g. EXAMPLE 1

```
begin  real     a, b, c, d;
       integer  procedure F(x); value x; integer x;
       begin  F := x*2;
              punch (3);
       end ;
       a := 1;
       b := 2;
       c := 3;
       d := 4;
       print       a, punch (2), b, F(c), d;
       comment  a is output to punch 1, b is output to punch(2), and the
       result of the procedure call F(c) is to cause 6 and the value of d to
       be output to punch(3) since the procedure F contains a call of
       punch(3) ;
       print       a;
       comment  a is output to punch 1;
end;
```

EXAMPLE 2

```
begin   integer          b;
        switch           ss := L1, L2;
        integer procedure F(L, a) ; value a ;
                         integer a ; label L ;
        begin
            if a := 2 then goto L ;
            F := a*2;
        end ;
            b:=1 ;
        L1 : print b;
            if b = 1 then goto L2;
            stop ;
        L2 : b := 2;
            print punch(3), b, F(L1, b) ;
        end ;
```

In the second example, the first time the print statement at L1 is obeyed, the value of b will be output to punch 1. However, the second time this statement is obeyed, b will be output to punch(3), since this was the value of the setting procedure "punch" current inside the print statement which contained the call of the procedure F.

2.1 Device setting procedures

In order to use input and output devices other than the first reader and punch, it is only necessary to mention the name of the device prior to the operands concerned.

reader (n) This standard procedure enables a particular device to be used for the input of data. n is an integer parameter and is the input device number. The value of n must satisfy

$$1 \leqslant n \leqslant 10.$$

If n lies outside this range, a return to the presumed setting occurs, i.e. reader (1).

punch (n) This standard procedure enables a particular device to be used for the output of results. n is an integer parameter and is the output device number. The value of n must satisfy

$$1 < n < (2^{19} - 1), \text{ i.e. } 524,287.$$

If n lies outside this range, a return to the presumed setting occurs, i.e. punch (1).

lineprinter This is exactly equivalent to the statement

punch (4);

2.1.1 Device used

When A3D wishes a character input as a result of a read statement, "instring" or "advance", it will enter a common program (CHARIN) with the current input device number in the accumulator.

The particular device number which is associated with a particular input device is determined by CHARIN.

In CHARIN ISSUE 1:-

reader (1) will set input from paper tape reader 1

reader (2) will set input from paper tape reader 2

reader (3) will set input from the typewriter.

reader (n) where n satisfies $3 < n \leq 10$ will set input from paper tape reader 1.

When A3D wishes a character output as a result of a print statement or "outstring", it will enter a common program (CHAROUT) with the current output device number and the character to be output in the

accumulator. The particular device number which is associated with a particular output device is determined by CHAROUT.

In CHAROUT ISSUE 1:-

punch (1) will set output to paper tape punch 1.

punch (2) will set output to paper tape punch 2.

punch (3) will set output to the typewriter.

punch (4) will set output to the lineprinter.

punch (n) where n satisfies $4 < n \leq 524,287$ will set output to paper tape punch 1.

2.1.2 Method of Use

The presumed settings for the input and output devices, are, reader (1) and punch (1). If a call of a device setting procedure is made inside a read or print statement, the device specified is used only for the remainder of the statement in which it appeared. If the procedure is called outside a print or read statement it will cause the device specified to be used until changed by another call of the device setting procedures.

Example

```
        EXAMPLE;
    begin read  v,w,x,y,z,a;
          read  v; comment v is input from the presumed device,
          tape reader 1;
          reader (2) ;
          read w; comment w is input from tape reader 2;
          read x; reader (3), y, z; comment x is input from tape reader 2,
          y and z are input from the typewriter;
          read z; comment z is input from tape reader 2;
    end;
```

2.2 Prefix setting procedures

The procedure 'sameline' should be used to print numbers

on one line; this will suppress output of the prefix (new line) normally output before each number.

The statement

print t, £tons?, sameline, c, £cwt?, l, £lbs?;

causes the value of t to be printed on a new line (the presumed setting), followed by the rest of the numbers and text on the same line, e.g.

17tons 12 cwt 19lbs

If all the numbers to be printed on one line are to be preceded by the same character or characters, the procedure 'prefix (£....?)' should be used. This causes the text displayed between £ and ? to be printed (in the place of new line) before every number. If the procedure 'prefix' is used, the procedure 'sameline' is unnecessary. In fact, 'sameline' will cancel the effect of 'prefix' if it occurs after it in the list, and vice versa.

For example, to print five numbers on a line separated by a comma and two spaces, write the statement

print a, prefix(£, £s2??), b, c, d, e;

2.3 Format setting procedures

The presumed format settings are satisfactory for numbers up to a hundred million in absolute size, with an accuracy of eight significant figures. If the range of numbers to be printed could be greater or less than this, or if a greater or lesser accuracy is required, it may be desirable to change the number of digits printed. To specify the number of digits in an integer expression, write 'digits(n)' in the list just before the items to be printed with n digits, where $1 \leqslant n \leqslant 12$.

For example, if i, j and k are integer variables, the statement

```
print i, digits(4), j, k;
```

could produce the output:

```
  -213
-1024
  362
```

The first integer is printed under control of the presumed setting (8 digits), while the other two have up to four digits, and therefore take less room in the column.

The procedure 'digits' has no effect on the printing of the values of real numbers or real variables. To achieve the same effect with real numbers, use the procedure 'freepoint(n)', where $1 \leqslant n \leqslant 9$; this has no effect on the printing of integers. For example, the statement

```
print x, freepoint(4), y, z;
```

could produce the output:

```
1.9632450
176.1
.0032
```

2.4 Scaled and aligned formats

When a column of numbers is to be printed, it is often desirable to position the numbers in such a way that the decimal points of all the numbers in the column are aligned one beneath the other. Elliott 503 ALGOL provides two methods of doing this. The scaled format always places the decimal point after the first digit, and uses an exponent part to indicate the scale of the number. The aligned format allows the programmer to specify the number of digits to be printed before and after the point.

To obtain the scaled format with n digits, write 'scaled(n)' as a setting procedure, where $1 \leqslant n \leqslant 9$; to obtain aligned format with m digits before the point and n digits after, write 'aligned(m, n)' as a setting procedure, $(m + n \leqslant 15)$; to revert to the freepoint format with n digits, write 'freepoint(n)' as a setting procedure. Note that more than one setting procedure may appear in a print statement; if the setting procedures contradict each other, the later ones cancel the effect of the earlier. Thus the statement

print x, prefix(£, ?), scaled(4), x, aligned(4, 3), x, freepoint(4), x

would cause the following sample output if it were obeyed repeatedly:

-1.2345678,	$-1.235_{10} + 00$,	-1.235,	-1.235
123,45678,	$1.234_{10} + 02$,	123.457,	123.5
.001234567,	$1.235_{10} + 03$,	0.001,	.0012

2.5 Additional format setting procedures

2.5.1 Grouping of digits

The special format setting procedure 'grouping(n)' should be used to obtain a spacing of digits in printed numbers. The digits of numbers output under control of this setting procedure are grouped in sets of n, the sets being separated by a single space. The origin of the grouping is the decimal point for real numbers, and the least significant digit for integers.

This procedure may be used to improve the legibility of numbers which consist of a long series of digits. It is not recommended for use with the freepoint format. Tapes produced under the control of 'grouping(n)' can be reinput only under the control of special(4) (see 2.5.3 of this chapter).

2.1.5.3

The following examples illustrate the effect of the procedure statement 'grouping(3)' on the output of integers, and of real numbers in the aligned and scaled formats.

1 076 253	12 073.193 6	$1.207\ 319_{10} + 04$
-92 765	- 120.965 0	$1.209\ 650_{10} + 02$
- 142	-24 306.701 4	$2.430\ 670_{10} + 04$
12 630 441	12.000 0	$1.200\ 000_{10} + 01$

2.5.3 Leading zeros

Integers are normally printed with leading zeros replaced by spaces and the sign floated. The setting procedure statement 'leadzero', allows the programmer to specify any character to replace these leading zeros. It also suppresses the floating of the sign.

E.g.

normal print format	setting procedure statement	resulting print format
-9	leadzero (£0?)	-00000009
176	leadzero (£*?)	*****176
-9	leadzero (£?)	-9
-12345	leadzero (£?)	-12345
-12345	leadzero (£ ?)	- 12345

Note that if the actual parameter is a null string, the significant digits and the sign if negative, are the only characters printed. Also the characters £ ? blank and erase may not appear within the string (i.e. an inner string is not allowed)

2.5.3 Specialities

The design of output documents sometimes requires certain special effects which may be obtained by the use of the procedure 'special(n)', ($1 \leqslant n \leqslant 4$).

special(1) causes the space normally output before positive numbers to be suppressed. It should not be used when printing negative numbers.

special(2) causes a plus sign to replace the space normally output before a positive number. It should be used if the numbers are separated by spaces and are to be reinput under special(4).

special(3) causes any sign output not to be floated see 2.5.2 of this chapter.

special(4) affects reading only, and causes spaces, normally treated as terminating characters, to be ignored. This facility enables tapes with grouped numbers to be reinput.

special(1) cancels special(2) and special(3), and vice versa. special(2) and special(3) can coexist.

2.5.4 presume

The standard procedure "presume", which has no parameters, may be used to restore all the setting procedures to their presumed values.

i.e. reader (1)
punch (1)
digits (8)
freepoint (8)
prefix (£ £ 1 ? ?)
leadzero (£ ?)

"grouping (n)" and "special (n)" are cancelled.

It may be used with local effect, when placed in a read or print statement and with global effect when placed outside a read or print statement in exactly the same way as is described for the procedures

2.1.5.3

"reader(n)" and "punch(n)" (see this chapter 2.1, method of use).

2.5.5 Character handling procedures

advance (n)

This has the effect of entering a common program (CHARIN) to be supplied with 1 character from input device n.
The characters erase and blank are ignored. n is an integer parameter and must specify

$$1 \leqslant n \leqslant 10.$$

If n does not lie within this range, the statement has the effect of "advance(1)". The character received will be stored in a 1 location buffer (see 2.9) associated with input device n overwriting the character previously stored there. This character can be accessed using the standard procedures "buffer" or "character".

buffer (n, £ a?)

This is a boolean procedure which takes the value true, if and only if the current character in the buffer for input device n is the character "a". "a" must be a single character but not £ or ? or blank or erase. No character is actually input from a read device. The use of an inner string is not allowed in this case. n is subject to the same conditions as are described under "advance".

character (n)

This is an integer procedure with takes the 503 paper tape code value of the character currently in the buffer for input device n. No character is actually input from a read device. n is subject to the same conditions as are described under "advance"

2.5.6 Errors

If an attempt is made to print a number that is too large for the style of printing called for, e.g. to print 289.2 in the aligned(2, 4) format, or to print 12347 in the digits(4) format, then 'alarm printing' occurs in such a way that the layout of the printed page is undisturbed. Provided that the total number, n, of characters called for is greater than six, the number is printed in scaled(n-6) format; if $n \leqslant 6$, the letter H and a halt code are printed, preceded by (n-1) spaces.

An attempt to print a real number that is not in standard floating point form causes the message PRINT ERROR to be displayed.

2.5.7 Parameters out of range

If the parameters of a format procedure are out of the permitted range, a return to presumed settings occurs as follows:

Procedure called	Permitted range of parameters	Presumed setting occurring
digits(E)	$1 \leqslant E \leqslant 12$	digits (8)
freepoint(E)	$1 \leqslant E \leqslant 9$	freepoint(8)
scaled(E)	$1 \leqslant E \leqslant 9$	freepoint(8)
aligned(E, F)	$1 \leqslant E + F \leqslant 15$	freepoint(8)
grouping(E)	$1 \leqslant E \leqslant 12$	prefix(£ £ ℓ ? ?)
special(E)	$1 \leqslant E \leqslant 4$	Cancels grouping, special(1), special(2) and special(3) as for grouping but special(4) is also cancelled.
reader(E)	$1 \leqslant E \leqslant 10$	reader (1)
punch(E)	$1 \leqslant E \leqslant 524,287$	punch (1)
advance(E)	$1 \leqslant E \leqslant 10$	advances one reader one
buffer(E, £ ?)	$1 \leqslant E \leqslant 10$	tests buffer of reader one.

2.6 Input and output of strings

[see also 'output of text' (1.5)].

Strings may be read in as data, and printed out again by using the procedures

instring (A,m)
outstring (A,m)

A is a single dimensional integer array
m is an integer variable (it may be subscripted but it must not be a constant).

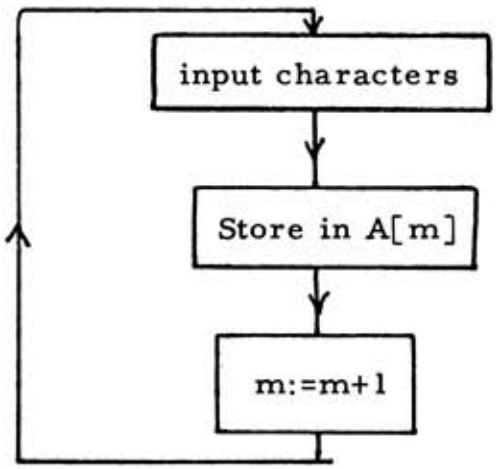
2.6.1 Instring (A,m);

This procedure causes a search on the current input device to find the opening string quote "£". If, during this search, a numeric character is found the message

READ ERROR

is displayed.

The string which follows the "£" is input as follows:



see 'storage of strings' (2.7)

until the close string quote "?", which brackets the original "£", is found.

The string is therefore stored in successive elements of the array A starting at A[n] (where n is the value of m on entry to the procedure) and m is set to the index of the next available element of A.

Thus, if the procedure is now re-entered, the next string will be automatically stored in the array without overwriting the previous string. If it is required to output the strings in an order which differs from the input sequence, the value of m on exit from the procedure should be recorded (see outstring).

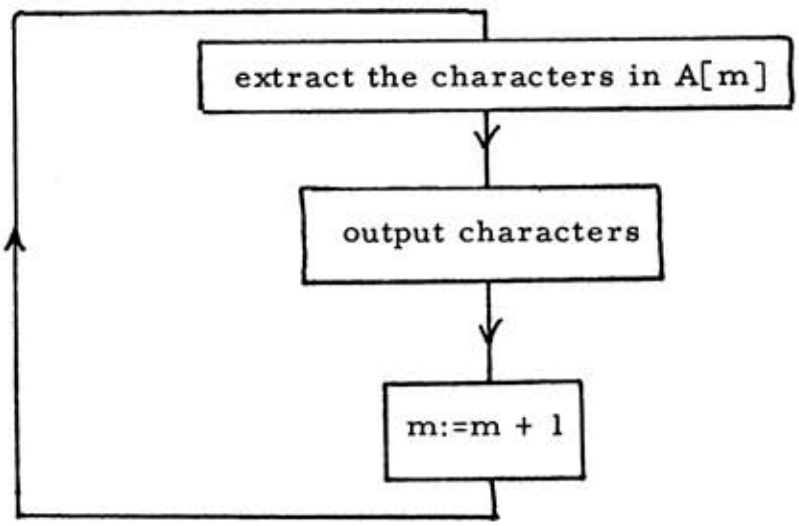
No check is made on the content of inner-strings by this procedure (see outstring).

The array A should be large enough to contain all the strings read. A string of n characters will occupy

(n + 4) ÷ 5 locations.
(see 'storage of strings' 2.7).

2.6.2 outstring (A,m);

This procedure will output on the current output device a string, previously stored in the array A by the instring procedure, as follows:



until the end of the string is found.

Thus, on exit from the procedure, m is set to the index of the element containing the first characters of the next string (if any), so that on repeated entry to the procedure successive strings stored in the array A will be output in succession. Alternatively, strings may be selected by setting m to the appropriate value (see instring).

Inner-strings are interpreted in the same way as that specified for strings occurring in print statements. (See 1.5 of this chapter).

2.7 String parameters and storage of strings in 503 ALGOL

(a) String parameter in a call of "elliott" or a code statement

The characters of a string are packed into a group of consecutive locations with the leading characters in the lowest-addressed location (see below). The address of this location is assigned to the string parameter; thus if S is a string parameter the order

elliott (0,0,S,1,3,0,0); or code 00S/30 0;

will place the contents of the first location of the string in the accumulator.

(b) Method of packing strings

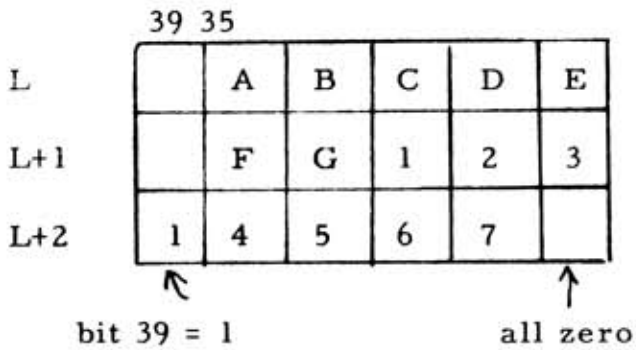
The 7-bit characters are packed five to a location to occupy the least significant 35 bits. Bits 36, 37 and 38 are always zero, but bit 39 is set to 1 in the last location of the string. The order of the characters in the location is such that the character in bits 35-29 precedes, in the string, the character in bits 28-22 which precedes the character in bits 21-15 and so on. If there are less than 5 characters in the last location the characters are stored such that any spare bits are at the least significant end.

When a location is full the adjacent higher-addressed location is filled and so on until the whole string has been stored. The string thus occupies a group of consecutive locations such that the leading characters are in the lowest-addressed location.

e.g. the string

£ABCDEFG1234567?

is packed as



(c) Inner Strings

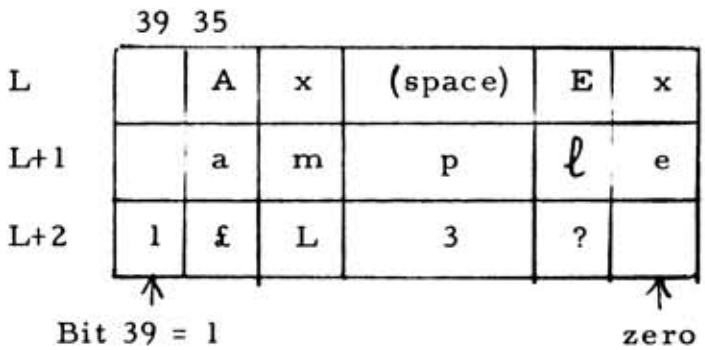
The initial £ and the final ? are not packed. However, inner £'s and inner ?'s are stored.

Characters within an inner-string are stored in the normal 7-bit form.

e.g. the string

£An ExampleL3??

is stored as



2.8 Procedures in read and print lists

The body of a procedure occurring in a read or print list may contain read and print lists and setting procedures. On entry to each read or print list the current read format or print format settings respectively are preserved. They are restored on exit. Thus the input and output of numbers during the execution of a procedure can be controlled externally by putting the procedure call into an appropriate read or print list.

2.9 Read device buffer

Because it is possible to have a data tape in which the terminator of one unit of data is the first character of a second unit of data, caution must be exercised when data is input using elliott orders or code statements. A one location buffer inside A3D, is associated with each input device, in which A3D will store the last character read from that device by the ALGOL routines, read, "instring" or "advance".

This buffer is examined by A3D whenever it is possible that it could hold a significant character.

e.g. consider a tape of the form

2.4£ABC?

The number, if input as a result of a read statement would be terminated by the £ of the string. Unless this character is preserved in the character buffer it would be lost, resulting in errors when the string was read by the "instring" procedure.

2.10 Lineprinter procedures

The lineprinter is made the current output device, by using either of the device setting procedures "punch(4)" or "lineprinter". The procedures "top of form", "find(M)", "lines(M)", and "overprint" are available in ALGOL3.

2.10.1 top of form

This procedure causes the lineprinter to search for a hole in the top of form channel of the lineprinter vertical format control loop and print on that line. It is the responsibility of the programmer or operator in charge to see that the control loop is correctly placed before the program is run.

2.10.2 find(M)

This causes the lineprinter to search for a character of value M on the control loop and print on the corresponding line. M must take values from 0 to 30. If M is negative or greater than 30, then the message "ERRCALL LP" is displayed on the output typewriter, followed by a "Dwait". If the left most F2 digit (Key 19) of the word generator is depressed, the procedure "top of form" is entered and the program is allowed to continue.

2.10.3 lines(M)

This causes the lineprinter to throw M lines. The call "lines(1)" has no effect. The same range and error action occur as with "find(M)"

2.10.4 overprint

This will cause the lineprinter to overprint as soon as a newline character is encountered or the buffer is full. In this case the effect of a newline character is to print on the same line all the characters formed since the last newline character. It is the responsibility of the programmer to determine proper page layout.

NOTE: The printing and other operations are not performed on the lineprinter until a newline character is encountered or until the lineprinter buffer is full.

Chapter 4: ADDITIONS TO THE LANGUAGE

1. INTRODUCTION

In addition to the standard functions described in the ALGOL60 report, certain other functions and procedures are available to users of Elliott ALGOL without explicit declaration.

2. ADDITIONAL STANDARD FUNCTIONS

arccos(E)	:	result in range 0 to π
arcsin(E)	:	result in range -π/2 to +π/2
tan(E)	:	the tangent of the value of E
oflo noflo		used to suppress certain error messages and affect the continuation values after the call of certain standard procedures (see chapter 5.4.3.1).

3. CHECKING FUNCTIONS

The standard checking functions provide the programmer with a means of optional printout of intermediate results during the course of a calculation.

There are four such functions:

checkr	for real argument
checki	for integer argument
checkB	for Boolean argument (the spelling checkb is also accepted)
checkS	outputs a string trace

and they are written in the program as functions with a single parameter, e.g.

```
A := checkr(B + 2*C) + 1.5;
```

If the B-digit on the keyboard is not depressed during translation, the checking functions are ignored - that is to say, the statement (1) is treated

exactly as if it were

```
A := (B + 2 * C) + 1.5;
```

If the B-digit is depressed during translation, then extra orders are compiled so that, if the B-digit is depressed also during the running of the program, the value of the argument is output on the current device. Thus (1) would have the effect of assigning to A the value of B+2*C+1.5, and of printing out the value of B+2*C.

In the case of Boolean checking, the output is one of the words true or false. In all cases check output is preceded by a change to a new line and an asterisk.

The first three checking functions may be nested, so that it is possible to write

```
A := B + checkr(M[checki(i + 3)]);
```

in which case the value of (i + 3) would be output before the value of M[i + 3].

There is a checking procedure

```
checks
```

whose parameter is a string. It can conveniently be used to provide a form of 'trace' of the progress of a calculation, or to identify the output of the checking functions. A typical call of this procedure would be

```
checks(£ stage 1 complete?).
```

Groups of checked output can be spaced by using the procedure call checks(£?), which outputs a newline character and an asterisk only.

4. ARRAY HANDLING PROCEDURES

In certain applications of ALGOL it is considered highly important to produce an efficient program. While the ALGOL translator in general produces machine code programs of reasonable efficiency, there are

certain occasions where ALGOL prevents efficient use of the computer. For this reason, some facilities, similar to those available in machine code, have been introduced in Elliott ALGOL.

4.1 Addressing facilities

The addressing of subscripted variables is one of the less efficient aspects of ALGOL. Elliott 503 ALGOL therefore provides facilities which enable the programmer to write his own address calculation routines in a notation very close to that of ALGOL, without necessitating a knowledge of the machine code of the 503.

In order to do this effectively, it is necessary to know the method of storing arrays in the computer. A variation of one in the last (rightmost) subscript of an array variable corresponds to a variation of one in the address of the variable. A variation of one in the other subscripts corresponds to a variation in address equal to the product of the ranges of all subsequent subscripts. Thus, in the case of a matrix, storage may be said to be rows.

E.g. Given the array A[1:8, 1:9, 1:10]
and two elements A[3,4,5] and A[3,5,5]
the difference in the addresses of the locations in which they are held = 10.

In order to compute the address of a subscripted element of an array, use may be made of the following standard functions, which have integer values:

address(A)	the address of the first element of the array A;
lowbound(A,I)	the declared lower subscript bound for the Ith subscript position from the left in the array A;
range(A,I)	the number of values in the range of the Ith subscript position from the left in the array A;
size(A)	the number of elements in the array A.

The first element of A is the subscripted variable all of whose subscripts take values equal to their lower bounds. The range of a subscript is computed by subtracting the lower bound from the upper bound and adding one. The size of an array is equal to the product of the ranges of its subscripts.

E.g. given the array A[1:8, 3:12, 5:15];

address (A) would take the value of the address of the subscripted variable A[1,3,5];

lowbound (A, 2) would take the value 3

range (A, 2) would give the value 10

size (A) would give the value 8 x 10 x 11

i.e. 880

All these standard procedures may be used with main store or with core-backing store arrays (see Section 10). However, when using the standard function "address" the user must realise that he will be given the address of a location on core-backing store, if the parameter is the name of an array held on core-backing store.

For a description of the error message displayed if in a call of "lowbound" or "range" the value of I does not satisfy the condition

$1 \leqslant I \leqslant$ number of subscripts of A, see chapter 5.

The following table gives the formulae which may be used to calculate the addresses of subscripted variables:

variable	address
A [I]	I - lowbound(A,1) + address(A)
A [I,J]	(I - lowbound(A, 1))*range(A, 2) + J - lowbound(A, 2) + address(A)
A [I,J,K]	((I - lowbound(A,1))*range(A, 2) + J - lowbound(A, 2)) *range(A, 3) + K - lowbound(A, 3) + address(A)

location [I]

is a subscripted variable of type real which is used to access the main store location whose address is the value of the unsubscripted integer variable I. [In effect "location" is a single-dimensional array whose subscript takes values corresponding to addresses of main store locations].

Like any arithmetic variable, it may occur on either side of an assignment statement.

A particular use of this variable is to access elements of main-store arrays where the addresses have been calculated by the above standard functions, e.g. calculate the address of the element I of array A:-

```
i := I-lowbound (A,I) + address (A) ;
```

and now access this element

```
x := location[i] +2.0; (assigns the value of A[I] +2.0 to x)
location[i]:= x; (assigns the value x to A[I].
```

NOTE: The corresponding facility for core-backing store may be obtained by using the single-word transfer orders for core-backing store in a code statement.

The facility for explicit address calculation will be most profitable in the following cases:

(i) the same subscripted variable occurs several times in close proximity

(ii) array variables with the same subscript bounds appear with the same subscripts several times in close proximity

(iii) one or more variables used as subscripts are controlled by nested for clauses, and none of the relevant subscripts is altered by assignment within the loop. In this case the necessary increment to the address may be computed outside the loop and there is no need to resort to a multiplication inside the loop.

5. STOREMAX PROCEDURE

The standard integer procedure 'storemax' takes as its value the number of locations of free store currently available in the main store for use by an array. The declaration for this array would be:

```
begin array A [I : storemax];
```

If any other arrays are declared in the same block head or in an inner block, or as workspace of a procedure called in the block, the combined size of these arrays must be subtracted from storemax; otherwise space overflow may result.

The activation of recursive procedures or procedures in which a local array is declared, is also likely to cause space overflow in a block whose arrays already occupy most of the available space.

The procedure storemax may not be used to obtain the number of free locations on core-backing store. However, if this facility is required, the information given in Chapter 1.2.2 should enable a user to write his own procedure.

6. CONTROL PROCEDURES

Elliott ALGOL provides a number of standard procedures to assist in the operating of a program; most of them do not have any effect on the actual calculations.

6.1 wait

This procedure causes the operation of the program to be held up until a signal is given by the operator to continue. It should be used in cases where some action is required from the operator (e.g. changing paper or magnetic tapes). The operator must be informed what action is required by an entry under Special Instructions on the Operating Sheet.

6.2 restart

This procedure causes a transfer of control back to the beginning of the program. It should be used whenever it is expected that the same program will be run several times with different sets of data.

If the 'restart' instruction is omitted from a program, and it is required to run the program several times, the operator may be asked to make a manual restart at the end of the program.

6.3 stop

This procedure causes an immediate end of the program, thereby freeing the computer for the next program to be run. However, it is still possible to make a manual restart.

7. MACHINE CODE

There are certain operations which may be performed very efficiently when expressed in the machine code of the 503, which cannot be performed efficiently when they are specified in ALGOL. This is because the ALGOL translator must cater for all possibilities, but the programmer writing in machine code, need only consider his special case. Therefore a facility is included for writing machine code instructions in an Elliott ALGOL program.

Machine code is particularly effective in cases where the following operations occur in inner loops:

(1) multiplication or division of integers known to be positive

(2) multiplication and division of positive integers by a power of 2.

The use of machine code imposes on the programmer the responsibility for testing and clearing overflow where necessary, and of ensuring that the program itself is not interfered with by a B-lined instruction. It is strongly recommended that all programs should be written exclusively in ALGOL during the testing stages, and that when some of the operations are replaced by machine code, all the tests should be run again to ensure that no coding error has occurred.

ALGOL3 provides for ALGOL programs containing machine code instructions to be compiled provided that the instructions are written in the form of code statements (see 7.2 below). Code statements have largely superseded the standard procedure "elliott", but programs including "elliott" procedures are accepted by ALGOL3.

7.1 Code statements

7.1.1 General

Code statements provide a means of writing in machine code within an ALGOL program. The format is based on 803 TI code (T102) with the main addition of 'diamond bracket' (DB) constants. All references to other words within the code statement are made using relative addressing.

E.g. 30 1,

loads the accumulator with the contents of the second line of the code statement.

7.1.2 Syntax

A new basic word code is defined which begins the code statement. A code statement is an unlabelled basic statement (see 4.1.1 revised ALGOL report Computer Journal, January, 1963). It can hence appear anywhere that an assignment statement can appear. It is terminated in the usual way by end or else or semicolon.

7.1.3 Description

7.1.3.1 Every line of code specifies a machine word. A line of code is terminated either by a newline or by the end or else or semicolon at the end of the code statement.

7.1.3.2 Each line of code can be either

(a) a wholeword

(b) an instruction pair.

7.1.3.3 Wholewords

There are five kinds of wholeword:-

(a) Numbers

These consist of a sign followed by an unsigned number in the ALGOL format, e.g.

+3
-.47
+3.14159
$-_{10}5$
$+1.0_{10}-2$

NOTE: spaces and tabs are ignored.

(b) Octal Groups

These consist of an 8 followed by 13 octal digits. e.g. 87700775770077

NOTE: spaces and tabs are ignored.

(c) Alphanumeric Groups

These consist of a £ sign followed by 5 characters. The 5 characters following the £ are packed in reverse order. e.g. £ ? A, $/_{10}$ would be packed into a location as $_{10}/$, A ? and the top four bits would be left clear.

NOTE: spaces, tabs and newlines are packed in the same way as other characters.

(d) RAP replacements - see advanced facilities, ref. 4.2.

(e) RAP words - see advanced facilities, ref. 4.3.

7.1.3.4 Instruction Pairs

An instruction pair consists of two instructions separated by the appropriate B-digit (/ or :). If only one instruction is supplied the separator must still be specified.

An instruction is either null or consists of two octal digits (specifying the function) and an address.

An address is either null, absolute, relative, or a DB constant.

(a) Relative Addresses

(i) An identifier

The identifier is replaced by an address at time of translation as indicated in the table below:-

Identifier	Address
simple variable or formal parameter called by value, i.e. real, integer or Boolean. Simple name variables must not be used.	the address of the location in which is stored the value of the variable, e.g. 26 a sets the variable a to zero.
array or formal parameter specified as an array whether called by name or value	the address of the representative location of the array, i.e. the address of a location which contains the address of the first element of the array, e.g. 00A/260 sets the first element of the array A to zero whatever ranges the subscript(s) of A have. (For the method of storing arrays, see section 10 of this chapter).
label within the current block which is NOT a formal parameter of a procedure	the address of a location which contains a jump to the statement which the label prefixed. Only the orders 40,41,42,43 should be used.
label within the current block specified as a formal name parameter (a label cannot be specified as a value parameter)	the address of the representative location, i.e. a location which contains a jump to a routine which determines the ultimate destination of the label. Note: If the procedure has an array parameter called by value then it is not possible to refer to the label parameter even though the body of the procedure is not a block.

Identifier	Address
type procedure within own body	address of the location which holds the value of the procedure.
string specified as a formal name parameter	the address of a location which contains the address of the first location occupied by the string, e.g. 00S/300 picks up the first location of the string.

(ii) Relative to the first location occupied by the code statement. A relative address consists of an integer followed by a comma, e.g. 0, or 15, The first location of the code statement has relative address 0. Hence 0, refers to the first location and 15, refers to the sixteenth location or word of code in the statement in which it appears.

(iii) Reference to another program - see advanced facilities, ref. 4.4.

(iv) References to the dynamic routines, public block - see advanced facilities, ref. 4.4.4.

(b) Absolute Addresses

(i) An integer

The integer may be signed or unsigned, e.g. 3 or +4 or -1.

NOTE: In an instruction with a negative absolute address, the address is made positive by decreasing the function digits, e.g. 20-1 is turned into 178191. If the instruction is 00 then no carry is made, e.g. 00-1: 00-1 is translated as 77 8191: 778191.

(c) Diamond bracket (DB) constants

These consist of < followed by a wholeword or an instruction pair and terminated by >.

e.g. <+3> or <-3.142> or <40255/00255>
or <20 a: 776,> or </>

It specifies the contents of a location whose address is allocated by the compiler. The DB constant is replaced by the address of the contents of the DB.

e.g. 30 <+7>

loads the accumulator with +7.

The space for the DB constant is allocated by the compiler. All constants of the same value are allocated the same address and are commoned with the constants in the rest of the program.

e.g. a := 1; code 30<+1>: 20b
30<:001>: 20c;

will only use the single constant +1.

NOTE 1. DB constants cannot be nested.

2. Apart from the restriction imposed by 1. above, any whole word or instruction pair can appear inside the diamond brackets.

7.1.4 Advanced facilities

7.1.4.1 Compound Addresses

The absolute part of an address can be split.

e.g. In the instruction 744129 the 4129 may be represented as 4096+33 or even 4096 + 32+1 denoting that A is output to the typewriter; or 764869 could be written as 76 4096+512+256+4+1, i.e. prepare to write to magnetic tape using controller 2 format 2, handler 1 even parity.

(a) Relative compound addresses, e.g. a+2 or 6,-3 are allowed but they must be used with care. Two relative addresses may not be combined to form a single address, e.g. a+2, will be rejected.

If the resultant relative address lies below the program or data area, then the address will be rejected causing error no.103 to be displayed.

(b) Each part of the address, and the final result must not be greater than 8191, i.e. both 4096+4096 and 8192-15 are not allowed.

7.1.4.2 RAP replacements

The following references to RAP subroutines are allowed as wholewords:-

RAP1print, RAPprint, RAPiprint, RAPsearch, RAPsneak, RAPread.

RAPiprint (737932:448069) displays the accumulator on the typewriter as a four digit integer with no suppression of leading zeros and no sign. The accumulator is taken to be positive.

RAPsneak (737932:448147) obeys the order pair in the accumulator on entry.

The order pair in the accumulator is obeyed and the resultant content of the accumulator is preserved so that on the next RAPsneak entry, this result will be restored to the accumulator before obeying the order pair. The auxiliary register is not preserved.

RAPsneak is mainly used for altering the reserved area without producing an error interrupt.

For a specification of the other words, see the description of RAP (2.2.1.7).

The following references to the RAP workspace are allowed:-

RAPFF, RAPLF, RAPpointer, RAPword, RAPsep, IMASK.

They are all treated as relative addresses when in their mnemonic form.

For a specification of these addresses see the description of RAP (Section 2.2.1.7 of the MANUAL).

7.1.4.3 RAP words

These are wholewords and consist of a 9 followed by letters, digits and spaces. These characters, up to a maximum of six are packed in RAP reduced form, i.e. for output by the RAP sub-routines RAPlprint and RAPprint.

NOTE 1. a tab is treated as a space.

2. any characters after the sixth are ignored.

3. if the RAP word is terminated before the sixth character, spaces are added to bring the total number of characters packed up to six.

7.1.4.4 References to other Programs

(a) Each reference consists of a relative address followed by the named program.
e.g. 5,PML or 0,PML.

0, is taken to refer to the first location after the RAP head. For SAP programs this would be the LINKCP.

(b) This facility is particularly useful when using another program as a common program.
e.g. 73 0, OWNPROG : 40 1, OWNPROG.

NOTE: the reference is treated as a relative address, insofar as it cannot be combined with another relative address.

(c) The program referred to must be in store at the time of translation. If owncode is produced, no check is made that the program referred to is in store when the owncode is read in. If it is not in exactly the same place in store then the subsequent action is undefined.

(d) References to the Dynamic Routines
If the program referred to has the name DRS then the reference is taken to be to the public block of the ALGOL Dynamic Routines.
i.e. 731,DRS sets the main link.

7.1.4.5 Title Facility

This is a compile time facility.

If an equal sign (=) occurs at the beginning of a line, then subsequent characters up to, but not including, the end of the line are output, preceded by a newline to the typewriter. The line is otherwise ignored.

7.1.4.6 Wait Facility

This is a compile time facility.

If a close round bracket or a double minus i.e.) or --, occurs at the beginning of a line by themselves they cause an Swait and are otherwise ignored.

7.1.5 Differences between code statements and other input routines

7.1.5.1 Elliott

The standard procedure elliott only inserts a wholeword each time it is used whereas a code statement can insert more. code statements can insert a single whole word and in this way could be used in the same manner as elliott. All the facilities of elliott are available in code statements.

7.1.5.2 T102

The following facilities of T102 are not allowed:-

1.	permanent directories	i.e. @
2.	temporary directories	e.g. +3,7 $
3.	block addressing	e.g. 4,6
4.	increasing block count	i.e. *
5.	preset parameters	e.g. 2015,8'4
6.	error cancellation	i.e. ?
7.	skip	i.e. %
8.	relative constants	e.g. +3,

The following facilities have been altered:-

1. stop i.e. -- now causes an Swait
2. stop and increase block count i.e.) now causes an Swait
3. title facility = ... bl output the title - WHICH IS TERMINATED BY A NEWLINE to the typewriter.
4. Alphanumeric facility, i.e. £ . All non-printable character, e.g. blank and erase, are ignored and only 5 characters are packed in the word.

All permitted contractions are allowed, e.g. 30,:means 30 0, : Identifiers in the address portion are also allowed in code statements.

7.1.5.3 S.A.C.

The following facilities of S.A.C. are not available:-

1. Instructions cannot be labelled.
2. A single order cannot appear by itself unless accompanied by a B-digit specifier.
3. No declarations or implied declarations are allowed, e.g. in S.A.C. block A; declares the block A and its LINK
4. Relative constants are not allowed.
5. Replacements other than those referring to RAP are not allowed. This includes standard identifiers like COMP and EXIT.

7.1.6 Error messages

See Chapter 5, paragraph 5.1.

7.2 elliott procedures

For description see section 2.1.3.3.4.2.1 of the Manual but note the following:-

7.2.1 Modification required to programs containing elliott procedures

In ALGOL1, the location (called the "representative location") assigned to an identifier which had been declared as an array contained only the address of the first element of the array in the least significant 13 bits of the word.

However, in the case of ALGOL3, this representative location will also contain other information required by the dynamic routines. Thus it might be necessary to insert a collate instruction in "elliott" procedures which contain references to an array identifier.

In ALGOL3, the sign bit is set in the representative location if and only if the array is stored on core-backing store, bits 38 to 20 contain information required by A3D for handling the array, and the address of the 1st element of the array is held in the least significant 19 bits, (more bits are required than in ALGOL1 since the address may now refer to a core-backing store location).

8. USE OF SAC COMMON PROGRAMS WITHIN AN ALGOL PROGRAM

To enable SAC common programs to be used as subroutines of an ALGOL program, a standard procedure "enter cp" has been introduced.

8.1 entercp (m,n,o)

This standard procedure is used to enter SAC common programs at a specified entry point. The parameters m,n and o must all be of type integer.

(i) m. This must hold the name of the SAC program to be entered. The name should be packed in the RAP form (see section 2.2.1.9 of the Manual).

(ii) n. This must hold the entry point number at which the program is to be entered.

(iii) o. When entry is made to the common program, the contents of o will be in the accumulator. This provides a method of passing a 39 bit parameter over to the common program.

E.g. To enter the SAC program LPRINT, the following could be written in the ALGOL program:-

```
code      30 <9LPRINT>
          20      m     ;
comment   This will place the name, LPRINT, packed in RAP form,
          in the location assigned to the integer variable m;
entercp   (m, 2, j);
comment   This will cause entry to LPRINT at entry point 2. The
          contents of the variable j will be in the accumulator when
          entry is made;
```

The address of the location holding the call of the standard procedure "entercp" will be planted in the LINKCP of the common program, so that it is possible to pick up several parameters once entry has been made to the common program.

e.g. The ALGOL program might contain the following:-

```
code      30 <9SEVPAR>
          20      m     ;

entercp   (m, 4, j);

code      40 a : 01 b
          20 c / 00 d
          01 f : 00 7930 ;
```

and at entry point 4 of the common program SEVPAR, the code might be:-

```
          20  WSI     (WSI:= contents of the variable j)
          67 LINKCP  (LINKCP holds the address of the call
          30    1     of the procedure entercp)
          20  WS2     (WS2:= 40 a : 01 b)
          67 LINKCP
          30    2
          20  WS2     (WS3:= 20 c /00 d)
          67 LINKCP
          30    3
          20  WS4     (WS4:= 01 f : 00 7930)

          EXITCP,   4.
```

The exit instruction in the common program must be suitably modified so that return is not made into the parameters following the entercp instruction in the ALGOL program.

Repeated entries to the same program at the same entry point

The procedure "entercp" will leave the instruction

```
73 LINKCP of
   SAC program  :  40 required entry point
```

in location "127, DRS" (see code statement description 7.1), so that when

entercp has been used once, the contents of this location could be accessed and stored in the ALGOL program.

This instruction could then be obeyed whenever entry has to be made to this common program at the same entry point, with a consequent increase in speed of execution. The instruction will remain in 127 DR's until overwritten by a further call of "entercp". It is only necessary to have the common program in store when the ALGOL program is running, so that it could be input when the "Dwait" is displayed at the end of compilation of the ALGOL program (see Chapter 6 for operating instructions).

9. PROGRAM SEGMENTATION

Programs which are too large for the available main store may be run if some parts are made segments. [See also 10, core-backing store arrays]. These segments are held on backing store (see this Chapter, 9.6) and are automatically brought into main store when required. Space is saved because the segments share the same main store area, known as a segment area (see 9.2 of this Chapter).

Except for the condition that a segment must be a block which is itself not part of a segment, the programmer has complete control over the segmentation of a program. In general no programming restrictions are imposed; for example, a programmer may include in a segment a call of a procedure whose body is also a segment.

9.1 Method of Use

To specify that a block is a segment, write a "significant comment" (see 9.1.1) immediately after the begin of the block.

9.1.1 Definition of Significant comment for segmentation

<comment list>: :=<any sequence not containing ;>

<segment-area number>: :=<unsigned integer>|+<unsigned integer>

<significant comment>: :=comment segment [<segment-area number>]
<comment list>; |
comment segment:<comment list>;

The segment-area number must lie in the range 1 to 16 inclusive (see 9.2).

comment segment: ; is equivalent to
comment segment[1] ;

The part of the comment which follows the] or the : is not significant and may be used for the ordinary purposes of comment. Thus,

comment segment[2] This is an example ;

significant part — segment area number — non-significant

E.g.
```
begin comment segment [2] ;
      real a, b ;
         etc
```

NOTE: (i) Only blocks may be segmented (a declaration must follow the comment statement)

(ii) Segments must not be nested (a segment may not contain another block specified as a segment).

9.2 Segment Areas and Segment Area Numbers

At compile time up to 16 areas of main store, known as segment areas, are set aside.

A segment is allocated to an area by means of its segment area number, and each area is as big as the largest segment associated with it.

The main program (all the code not contained in segments) always remains in main store, and a copy of each segment is always held on backing store (see Chapter 9.6). When a segment block is entered, if it is not already in main store, it is copied into the segment area corresponding to its segment area number. Thus, segments with the same number may not be in store together, but if a segment is re-entered several times before another segment associated with the same area is entered, it is not re-copied from backing store.

9.3 How to segment efficiently

9.3.1 Space

A segment area is as big as the largest segment associated with it. Thus, in order to reduce the size of the program in main store more than one segment must be associated with each segment area used.

The greatest amount of space is saved when all segments are placed in one segment area, and this is the simplest method of segmentation, but see TIME (this Chapter, 9.3.2).

9.3.2 Time

Because segments must be transferred from backing store to main-store, segmentation tends to increase the running time of a program.

Therefore to keep this increase to a minimum,

(i) segment as little of the program as possible,

(ii) segment those blocks which are infrequently entered,

(iii) place segments which are entered repeatedly one after the other, in different segment areas so as to reduce the frequency with which they are re-copied from backing store. But note that the program may have to be segmented further to allow for the space occupied by the extra segment areas.

(iv) try not to segment procedures with parameters that are called by name, if the procedure is called from other segments in the same segment area.

9.4 Segment Sizes

To assist the programmer in the decisions taken as to where the program may best be segmented a facility is provided whereby the size of each segment created is displayed on the typewriter at compile time. This facility is invoked by depressing key 38 on the word generator. As each segment is compiled the size of the segment is printed as a 4-digit integer preceded by the number of the associated segment area, also as a 4-digit integer (no suppression of leading zeros).

9.5 Error message interpretation (see Chapter 5)

*error no.55

(a) Nesting of segments has been attempted.

(b) The significant comment has been placed illegally, e.g. an attempt has been made to segment a compound statement.

(c) an impermissible segment area number has been used, e.g. greater than 16 or the number was not terminated by a close bracket.

9.6 Backing store

The storage of the segments depends on the compiler system and may be on core-backing store or magnetic tape. The initial writing-up of the segments to backing store and the subsequent re-copying into main store is carried out automatically and requires no action by the programmer.

10. ARRAY STORAGE

The arrays in a program compiled by ALGOL3 may be stored either in main store or in core-backing store. An array will be allocated space in core-backing store (provided it is fitted) if and only if the declaration in which the array appears is immediately preceded by a significant comment. In all other cases the array will be stored in main store. This applied to both own arrays and non-own arrays.

The syntax of the significant comment is:-

<comment list>: :=<any sequence of basic symbols not containing ;>

<significant comment>: := comment CBS:<comment list>;

If the first word following comment is CBS and the next character (ignoring the typographical features such as blank, space or change to a new-line, and the special characters & and halt-code) is not : then error No.55 will be displayed.

If no core-backing store is present then the significant comment will be checked syntactically but the array will be allocated space in main store.

The significant comment applies to the arrays in the declaration immediately following it and to no others.

Thus,

```
comment  CBS : any comment list;
integer  array  A[1:100],B,C,D[1:50,2:4];
integer  array  E[1:20],F[1:3];
```

will cause arrays A, B, C and D to be stored in core-backing store and arrays E and F to be stored in main store.

10.1 Arrays as parameters of procedures

A procedure which has a value array parameter will store the copy of the array in the same medium, (i.e. main store or core-backing store) as that containing the corresponding actual parameter. It is not possible to specify the medium on which the formal array parameter is to be stored.

11. LIBRARY FACILITY

ALGOL3 provides a facility for accessing a library of ALGOL routines held in character form on a magnetic tape. A new basic word library is introduced which serves to call the texts specified after it. A text on the library tape could be a procedure, a series of declarations or any other piece of code. The program LIBR1 acts merely as a macro-generator, taking the macro texts plus any associated texts from a tape previously prepared by the program LIBR2.

11.1 Writing texts to magnetic tape

The names and texts to be written to magnetic tape should be on paper tape in the following form. Firstly, a list of names of texts each separated by a comma the last name being followed by a semi-colon then the corresponding texts each separated by a HALT CODE, the last text being terminated by a HALT CODE. If, however, no more names and texts are to follow two consecutive HALT CODES must be provided.

In a library statement (read by LIBR1) or a list of names and texts (read by LIBR2) the paper tape characters with the following values are ignored:-

0, 2, 64, 123-127

and the following are treated as significant:-

20, 16-25, 32, 33-58, 76, 97-122

If any other character is encountered,the programs display an error message and do not continue. In addition, if any significant character appears in the wrong position, e.g. a halt code in a name, an error message will be displayed, (see LIBR1 and LIBR2 program description, Chapter 7).

Example 1

a, b, A; text of a Halt Code text of b Halt Code
text of A Halt Code
b, a; text of a Halt Code
X, y, Z; text of X Halt Code text of y Halt Code
text of Z 2 Halt Codes

The letters a, b, A, X, y, Z act merely as names to identify each text on the magnetic tape and in the source code when the text is called ; the name need not appear in the text itself. Only the first 6 characters of the name are significant ; the first character must not be an integer. If more than one name of a text is listed before a semi-colon all but the last name are treated as auxiliary names and texts. In example 1 a, b, X and y are auxiliary tests. As such, each time text A is called, the auxiliary texts a and b are also read from the library tape and similarly with Z, texts X and y will also be read. The number of auxiliary texts used by 1 routine must not exceed 6.

NOTE: The single halt codes should always be followed on the tape by a non-significant character (e.g. new line, space).

11.1.1 Texts already on the library tape

Note that if auxiliary texts are already on the library tape, as is the case with b in the second line of example 1, the names must be listed prior to any others although the texts should not be given. However, in the case of a text calling other auxiliary texts, see a in the second line of example 1, the text must always be given.

11.2 Reading texts from magnetic tape

To call a routine from the library tape, a statement using the basic word library and the name(s) of the texts is included in the ALGOL source program. The statement is terminated by a semi-colon in the usual way.

Example 2

```
a := 1 ;
library A, name, X ;
if b = 2 then go to label ;
```

The program LIBR1 reads characters from the tape in reader 1 until the basic word library is encountered. It then reads the specified library texts from magnetic tape and passes this code to the compiler. When the semi-colon /at the end of the library statement is reached the program LIBR1 returns to reader 1 to obtain its characters to pass to the compiler. Note that the texts specified after library will be compiled into store in their order on magnetic tape and not in the order as listed in the statement. A maximum of 30 texts may be referred to either directly or indirectly in any one library statement.

Example 3

```
library A, B, C, D, E;
```

where texts A, B, C, D and E each have 6 auxiliary texts.

Example 4

library A;

where A has 6 auxiliary texts, each auxiliary text calls 3 other texts one of which calls 5 more texts.

In example 2, the texts specified by A, name and X will be read from magnetic tape and the code passed to the compiler, although not necessarily in the order as listed in the library statement. If, as in example 1, auxiliary texts were written up with text A and X they will also be read from the library tape and passed to the compiler each time the main texts, in the example A and X, are called. However, if one of the auxiliary texts is called, only this one will be compiled.

NOTE: A call of library is not permitted inside a library text

12. EDIT FACILITY

ALGOL3 makes use of the program EDIT8 to give an editing facility so that source programs may be edited whilst being compiled. The edit commands tape must be prepared according to the EDIT8 description (see Manual, Section 2.2.3.27). No edited tape is produced but it would be possible to provide such a facility by modifying the common program INTER.

12.1 Edit and compile

The program tape to be edited is loaded in reader 1 and the edit tape in reader 2. The tapes are read by the program EDIT8 and the edited characters presented one at a time to the compiler.

12.2 Edit, list and compile

The program tape to be edited is loaded in reader 1 and the edit tape in reader 2. The tapes are read by the program EDIT8 and the edited characters presented one at a time to the compiler and the program ALPL (for output on the lineprinter)

12.3 Edit, input library text and compile

The program tape to be edited is loaded in reader 1 and the edit tape in reader 2. The tapes are read by the program EDIT8 and the edited characters presented one at a time to the program LIBR1. This program in turn passes the characters to the compiler.

NOTE: The editing facility applies only to the program tape; texts input by the program LIBRI from magnetic tape may not be edited.

12.4 Edit, list, input library text and compile

The program tape to be edited is loaded in reader 1 and the edit tape in reader 2. The tapes are read by the program EDIT8 and the edited characters presented one at a time to the program LIBRI, (this program in turn passes the characters to the compiler), and the program ALPL (for output on the lineprinter).

NOTE: The editing and listing facility applies only to the program tape; texts input by the program LIBRI from magnetic tape may not be edited.

For method of operation of these facilities see Chapter 6.

13. LISTING FACILITY

ALGOL3 makes use of the common program ALPL to give a listing facility on the lineprinter so that source programs may be listed as they are compiled into store.

13.1 List and compile

The program tape is loaded in reader 1 and the characters are presented one at a time to the compiler and the program ALPL.

13.2 Edit, list and compile

See 12.2 of this chapter.

13.3 Edit, list, input library text and compile

See 12.4 of this chapter.

13.4 List, input library text and compile

The program tape is loaded in reader 1 and the characters are presented one at a time to the programs LIBR1 and ALPL. The program LIBR1 passes the characters on to the compiler after having read any necessary library texts from magnetic tape and the program ALPL outputs the characters from reader 1 on the lineprinter.

NOTE: Library texts input from magnetic tape are not listed on the lineprinter although this is a facility that could be provided by modifying the program LIBRI.

For method of operation of the listing facilities see Chapter

14. SUMMARY OF STANDARD PROCEDURES

abs, sin, cos, arctan, entier, sign, exp, ln, sqrt	see ALGOL 60 report
tan, arcsin, arccos	see Section 2 of this Chapter
digits	see Chapter 3.2.3
freepoint, scaled, aligned	see Chapter 3.2.4
sameline, prefix	see Chapter 3.2.2
reader, punch	see Chapter 3.2.1
grouping	see Chapter 3.2.5.1
leadzero	see Chapter 3.2.5.2
special	see Chapter 3.2.5.3
presume	see Chapter 3.2.5.4
instring, outstring	see Chapter 3.2.6
advance, buffer, character	see Chapter 3.2.5.5
entercp	see Section 8 of this Chapter
oflo	see Chapter 5.4.3.2
noflo	see Chapter 5.4.3.1
address, size, range, lowbound	see Section 4.1 of this Chapter
storemax	see Section 5 of this Chapter
elliott	see Section 2.1.3.3.4.2.1 of the Manual

stop	see Section 6.3 of this Chapter
wait	see Section 6.1 of this Chapter
restart	see Section 6.2 of this Chapter
checki checkr checkb Check B checks	see Section 3 of this Chapter
top of form find lines overprint	see Chapter 3.2.10

Chapter 5: ERROR INDICATIONS

1. INTRODUCTION

Error indications given by the common programs which are used by A3C, A3L, A3D and the executive will be preceded by the common program name.
e.g. "OCBS error BSfull" would be displayed by OCBS if it discovered that there was not sufficient room on core-backing store for the segments.

For the meaning of an error message which is displayed by a common program and the possible action to be taken, the relevant common program description should be consulted.

The error indications described below (Sections 2, 3 and 4) will originate in A3C, A3L or A3D.

2. ERRORS DURING THE TRANSLATION OF A PROGRAM - 1st PASS

When a syntax error is detected by the compiler, the following action is taken:-

(1) The program is now invalid, so, if this is the first error, output of owncode is stopped.

(2) The error is classified: A3C outputs to the error-message program the message

*error no. n

(preceded and followed by a newline character) where n is an integer which gives a reference in the ERROR TABLE (2.4). If the error occurred in a library text held on magnetic tape, this message is followed on a new line by

*in mt text

(3) (a) Sufficient of the text following the error is also output in order to help pin-point the position of the error, (see COPIED TEXT).

(b) That part of the text which is immediately affected by the error is not checked for further errors. This means all characters between the error and the end or semicolon which terminates the statement, declaration, or procedure, (when the error occurs in the head of the procedure), are ignored. In the case of an error in a code statement the text is ignored only as far as the next newline, end, else or semicolon.

(c) The rest of the program is checked for further errors.

2.1 Copied text

The executive specifies the name and entry point of an auxiliary program to which syntactic error messages (see Chapter 1.2.1) are sent. This auxiliary program determines the amount of the text following the error which is to be displayed. (See INTER description, Chapter 7.3.7). The number of characters displayed using INTER issue 1 is 31.

Should a further error be detected in the text which is being copied, A3C will output the appropriate error message and copying will start anew.

If the ALGOL program is being both printed on the lineprinter and compiled, then if the error messages are also printed on the lineprinter no code is copied since the error messages will be inserted immediately after the error (see INTER description).

2.1.1 Note on default option

If the A3C default option is used, A3C will itself, display on the typewriter the syntactic error message followed by 31 characters of the source code.

2.2 Notes on error no.20 and error no.49

2.2.1 Error no.20 (undeclared identifier) is treated specially

The name of the offending identifier is output after the error number and the identifier is treated as though it had been declared to be of type integer in the head of the innermost current block or, if the symbol immediately following it is an open bracket, of type integer array. If this declaration is incompatible with the use of the identifier - for example, if it is used as a label or switch - then further, spurious, error indications will occur.

2.2.2 Error no.49

(Program too complex to be compiled), occurs if A3C requires more room in main store than is available. For methods of modifying the ALGOL program to avoid this situation see OVERFLOW CONDITIONS (Section 5 of this chapter).

2.3 Spurious errors

The explanations given by the error table will normally be correct. There are, however, some cases when an error will be detected but attributed to a cause unrelated to what has been written. When this happens, the actual error will usually be found in a statement near the apparent error.

2.1.5.5

It is possible for one actual error to lead to several spurious error indications; for example, errors in declarations are bound to cause spurious errors later since part of the declaration will be ignored.

2.4 Error Table

n	Error
1 (a)	Number of impermissible form.
(b)	Error in an underlined word.
2 (a)	Impermissible beginning to a statement.
(b)	Symbol following own not real, integer, Boolean or array.
(c)	Procedure declaration not terminated by semicolon.
(d)	Name in declaration not followed by comma or semicolon.
(e)	Declaration following procedure declaration not a procedure declaration.
3	Name declared twice in same block head.
4	Item after a comma in a declaration not a name.
5, 6	First identifier in switch declaration not followed by ':='.
7	In a procedure declaration:
(a)	Item following procedure name not semicolon or open bracket.
(b)	No semicolon or close bracket after formal parameter part. N. B. If the item following the close bracket is not a semicolon, the compiler will ignore a letter string until ':'.
(c)	List in value or specification part has impermissible form.

n	Error
7 (d)	Specification part occurs before value part.
8 (a)	Parameter of non-allowable type.
(b)	Too many parameters (> 1023).
(c)	Parameter not specified.
9	Recursive procedure with real, integer or Boolean name parameter.
10	Name in value part not a formal parameter.
11 (a)	Name in specification part not a formal parameter
(b)	Parameter specified twice.
12	go to statement leading out of a procedure body.
13	In array declaration:
(a)	No comma or open bracket after identifier.
(b)	No colon between upper and lower bounds.
(c)	No comma or close bracket after bound pair.
14	Negative own array size.
15	own array with variable or real bounds.
16	array with more than 31 dimensions, or array segment with more than 31 arrays.
17 (a)	No end or semicolon after statement in compound tail.
(b)	No colon after a label.
(c)	Impermissible beginning to a statement.
18	Left part variable in assignment statement not followed by ':='.
19	Value assigned to a procedure identifier outside procedure body.
20	Identifier not declared, or used outside scope of declaration.
21	'location' not followed by '[integer identifier]'.
22	Inadmissible complex primary in arithmetic expression.

n		Error
23		Empty arithmetic expression.
24		Missing operand in arithmetic expression.
25		No close bracket after subscript list.
26		Missing bracket.
27		Wrong number of subscripts in subscripted variable.
28		Missing else.
29		Missing then.
30		Conditional statement or expression after then.
31	(a)	Missing or inadmissible operator in arithmetic expression.
	(b)	Boolean operand in arithmetic expression.
	(c)	Missing open bracket after name of procedure with parameters.
32		Non-identifier or Boolean variable in a read list.
33	(a)	Inadmissible identifier as left part element.
	(b)	Inadmissible identifier as controlled variable in for statement.
34		Empty Boolean expression.
35		Missing relational operator.
36		Missing operand in Boolean expression.
37		Inadmissible complex Boolean primary.
38		Inadmissible operator in Boolean expression.
39		Type procedure name in read list, but not within its own body.
40		Inadmissible symbol at start of an expression.
41		During procedure call:
	(a)	No open bracket following name of procedure with parameters.
	(b)	Actual parameter not followed by comma or close bracket (cf. error 7(b)).

n	Error
42	Error in parameter delimiter of form ') letter string:('.
43	No actual array or string parameter where one expected in a procedure call.
44	Non-allowable parameter in a procedure call.
45 (a)	Controlled variable in for statement not followed by ':='.
(b)	For list element not followed by a comma or do.
46	Incorrect designational expression.
47	Arithmetic expression in for list element not followed by step, while, do or a comma.
48	Missing until.
49	Program too large or compex to be compiled.
50	Label not declared in innnermost possible block.
54 (a)	Occurrence of a £ within an inner string (possibly caused by omission of ? at end of a previous string).
(b)	comment occurs otherwise than after a semicolon or a begin.
55 (a)	Nested segment.
(b)	Segment is not a block.
(c)	Segment area number is less than 1 or greater than 16.
(d)	In a significant comment, "segment" or "CBS" is followed by an incorrect terminator.
56	Wrong number of parameters in a call of a procedure.

N.B. The following errors will cause an ALGOL program tape to shoot through the reader, instead of stopping at the end:-

(i) No HALTCODE at the end of a tape which is not the last tape of the program.

(ii) Insufficient end's to match all the begin's in the program.

(iii) No semicolon after the final end.

(iv) Missing ? at the end of a string, causing the program statements that follow to be treated as part of the string. (This error will be detected (Error 54) if an inner string occurs in any subsequent statement.)

2.4.1 Code statement errors

Error No.	Cause
101	more than one word on a line, e.g. newline omitted.
102	impermissible beginning to word, e.g. function digits omitted.
103	(a) named program not found (b) negative relative address.
104	error in function digit.
105	impermissible address, e.g. address or part of address > 8191.
106	error in DB constant, e.g. close diamond bracket missing.
107	(a) more than one B specifier in a word (b) more than one sign in a word (c) incorrect number, e.g. decimal exponent part $_{10}$.4
108	(a) error in octal group (b) impermissible word.

Error No.	Cause
109	impermissible identifier in address part.
110	(a) block address used, e.g. 3, 17
	(b) identifier used as wholeword is not a standard replacement.

2.5 Errors not detected

(1) Labels placed more than once.

(2) Labels not placed at all.

3. ERRORS DURING CONVERSION OF OWNCODE TO MACHINE CODE - 2nd PASS

The errors described below are detected by A3L, which will enter the executive for the following error messages to be displayed.

Message	Meaning
A3Lerr1	An error has been detected in the owncode supplied to A3L. No continuation is possible. This means an error has occurred in the system programs rather than the ALGOL source program.
A3Lerr2	There is not sufficient room in main store to hold the translated program. No continuation is possible. (See segmentation Chapter 4.9 and also storage of own arrays on core-backing store, Chapter 4.10.)
A3Lerr3	A3C allocates space to the core-backing own arrays on the basis of the value it finds in the pointer BSLF. A3L has now discovered that BSLF no longer holds this value. No continuation is possible, (see Chapter 1. 2.2.2). This means an error has occurred in the sytems programs rather than the ALGOL source programs.

Message	Meaning
A3Lerr4	There is not sufficient room available in core-backing store to hold core-backing store own arrays. No continuation is possible, (see Section 5 of this chapter).

4. ERRORS DURING RUNNING OF PROGRAMS

In some of the error messages described below the medium on which an array is stored will be displayed. The convention is:-

MS means main store

BS means core-backing store.

4.1 Non-continuable errors

Message	Meaning
INT OFLO	The result of some integer operation (e.g. division by zero) is outside the range -2^{38} to $(2^{38}-1)$.
SUBSCR OFLO	A subscript of a subscripted variable is outside the range declared for the corresponding array. The medium on which the array is held and the name of the array will be displayed. This will be followed by further diagnostics in the order listed: (i) The subscript position at which the error occurred. (ii) The value of the erroneous subscript. (iii) The current lower and upper bounds of each subscript position of the array, subscript position 1 first.
MS NOROOM	The program requires more space than is available in main store.

Message	Meaning
MS NOROOM FOR MS ARRAY A.	There is not sufficient room in main store for array A. This is displayed when the array has to be allocated space.
BS NOROOM FOR BS ARRAY A.	There is not sufficient room in backing store for array A. This is displayed when the array has to be allocated space.
MS NOROOM FOR BS ARRAY A.	There is not sufficient room in main store for the information which describes the core-backing store array A. This is displayed when the array has to be allocated space.
VAC NOROOM	There is not sufficient room to hold the copy of a value array parameter. The medium on which the actual array parameter is held and the name of the actual array parameter will be displayed, (see Chapter 4.10).
BOUND ERROR	The upper bound is less than the lower bound in one of the subscript positions of an array. This will be displayed when space is being allocated to the array. The medium on which the array was to be held and the name of the array will be displayed. This will be followed by the subscript position at which the error occurred and the current lower and upper bounds for this subscript position.
LOWBOUND ERROR	An error has been detected in a call of the procedure "lowbound". The medium on which the array parameter is held and the name of the array will be displayed.
RANGE ERROR	An error has been detected in a call of the procedure "range". The medium on which the array parameter is held and the name of the array will be displayed.

Message	Meaning
IOSTRING ERROR	The value of the subscript m (see Chapter 3.2.6) during the execution of the procedures "instring" or "outstring" lies outside the range declared for that subscript position of the array. This message will be followed by the diagnostic information that follows SUBSCR OFLO.
DIV ERROR.	The result of the operation div is outside the range -2^{38} to $(2^{38}-1)$.
ENTIER ERROR.	The result produced by the standard procedure "entier" or a necessary conversion of a number from real to integer (e.g. assigning a real number to an integer variable) lies outside the range -2^{38} to $(2^{38}-1)$.
SWITCH ERROR.	The value of a subscript expression in a switch designator lies outside the range 1 to n where n is the number of labels in the switch list of the corresponding switch declaration.

These errors will all be followed by the name of the last array (in the dynamic sense) successfully allocated. The name of the array is preceded by the word

ALLOC.

The copying of an array which occurs when a procedure which has a value array parameter is entered does not affect the name displayed for the last array allocated.

4.2 Continuable errors

After the following error messages, a "Dwait" will be displayed. The program can be continued by changing the leftmost F2 digit (key 19) on the word generator. In the descriptions below, x stands for the

argument of the function which caused the error and cv the value which the function will assume if the program is continued.

Message	Meaning
EXP ERROR	$x > 255 \log_e 2$ ($\triangleq 176.75$) $cv = (1 - 2^{-39}) \times 2^{255}$ (i.e. largest positive number the computer can hold).
SINE ERROR	This is displayed for an error in both the functions sine and cosine. $x \geqslant 2^{30}$ ($\triangleq 10^9$). $cv = 0$.
LOG ZERO	$x = 0$ $cv = -2^{255}$ (largest negative number the computer can hold).
LOG ERROR	This is caused by:- (i) ln (x), where $x < 0$. The continuation value is $\ln(\lvert x \rvert)$ (ii) attempted evaluation of $p \uparrow q$ with $p \leqslant 0$ and q real and $\leqslant 0$. The continuation value of the function is zero.
TAN ERROR	$x \geqslant 2^{30}$ ($\triangleq 10^9$). $cv = 0$
ARCSIN ERROR	This is displayed for an error in both the functions arcsine and arccosine. $x > 1$ $cv = 0$
SQRT ERROR	$x < 0$. $cv = \text{sqrt}(\lvert x \rvert)$.
STRING ERROR	The string being printed contains an inadmissible character. On continuation, the rest of the string is ignored.

Message		Meaning
READ ERROR	(a)	error found in the number being input (in particular, if a number expected to be an integer contains a decimal point or a subscript ten or is too large), or £ read at the start of a number. On continuation, the read subroutine is re-entered.
	(b)	'instring' has read either a numeric character before reading a £, or a £ within an inner string. On continuation, the program continues with the statement following the call of 'instring'.
BUFFER ERROR		The string parameter in a call of 'buffer' is an empty string. On continuation, the function is assigned the value false.
PRINT ERROR		The real number to be printed is not in standardised floating-point form. On continuation, the printing of the number is omitted.
ERRCALL LP		The parameter m in a call of the procedures "lines" or "find" is outside the range $0 < m < 30$. On continuation a call of the procedure "top of form" is executed.

After these continuable error messages it is possible to have displayed the name of the last array allocated by suitably modifying the common program CHAROUT. The run may then be stopped or allowed to continue (see CHAROUT description, Chapter 7.3.5).

4.3 Suppression of certain error messages

Two standard procedures, "noflo" and "oflo" have been introduced, which can be used to suppress some of the error messages and modify the values given to certain of the standard functions with particular arguments. Used in conjunction with the SAC program ERINT, "noflo" can

also be used to suppress the error message ERRINT1.

4.3.1 noflo

This is a procedure without parameters which, when called, affects the standard functions "ln", "sqrt", and "exp" in the manner described below.

(i) ln(0) The message "LOG ZERO" and the "Dwait" are not displayed. The continuation value is as before, i.e. -2^{255}.

(ii) exp(x) where $x > 255 \log_e 2$ ($\doteq 176.73$)

The message "EXP ERROR" and the "Dwait" are not displayed. The continuation value is as before, i.e. $(1-2^{-39}) \times 2^{255}$.

(iii) sqrt (x) where $x = (1-2^{-39}) \times 2^{255}$

This expression takes the value $(1-2^{-39}) \times 2^{255}$. Since sqrt (x) when $x<0$ is evaluated as sqrt ($|x|$) then, when $x = -(1-2^{-39}) \times 2^{255}$, this expression will also be given the value $(1-2^{-39}) \times 2^{255}$. The error message SQRT ERROR will still be displayed, however, when x is negative.

(iv) ln (x) where $x = (1-2^{-39}) \times 2^{255}$

This expression takes the value $(1-2^{-39}) \times 2^{255}$. Since ln(x) where $x<0$ is evaluated as ln ($|x|$) then, when $x = -(1-2^{-39}) \times 2^{255}$ this expression will also be given the value $(1-2^{-39}) \times 2^{255}$. The error message LOG ERROR will still be displayed, however, when x is negative.

4.3.2 oflo

This is a procedure without parameters and is used to cancel the procedure "noflo". The action taken in the cases listed in 4.3.1 of this chapter then reverts to that described in 4.2 of this chapter.

2.1.5.5

The procedure which is called last in the dynamic sense is the one that is operative. The presumed state corresponds to a call of "oflo".

4.3.3 Continuation after floating point overflow, i.e. ERRINT 1.

A SAC program ERINT has been provided to enable the run of an ALGOL program to be continued after the detection of a floating point overflow.

If ERINT is in store when the error interrupt occurs, RAP will enter ERINT which, unless the procedure "noflo" is currently active in the ALGOL program, will display

" ERINT 1
Dwait "

When the leftmost F2 digit (key 19) of the word generator is changed, the ALGOL program continues with the value $(1-2^{-39}) \times 2^{255}$ taken as the result of the operation which generated the floating point overflow.

If the procedure "noflo" is active in the ALGOL program then the run will be continued without an error message and with the continuation value $(1-2^{-39}) \times 2^{255}$, (see ERINT description, Chapter 7. 3.1.2.)

5. NOTES ON SPACE OVERFLOW CONDITIONS

[all solutions require the program to be re-compiled]

5.1 Compile time [i.e. during the first pass]

(a) Error No.49 - program too complex to be compiled at all.

cause: the compile time list and dictionary occupy all the available main store space.

solution: (i) if the error occurs at the start of a block reduce the number of variables declared in that block (possibly by splitting the block into 2 un-nested blocks)

(ii) if the error occurs during the evaluation of an expression (arithmetic or boolean) split it into a number of smaller expressions if possible.

(b) OCBS error BSfull

cause: the owncode has filled the core-backing store.

solution: recompile and place the owncode on magnetic tape. This involves modifying the executive so that it will cause A3C to use OAST instead of OCBS for the output of owncode (see OAST, ALGOLB and GMT (for ALGOL3) descriptions, Chapter 7).

5.2 Conversion of owncode to machine code - 2nd pass

(a) A3Lerr2

cause: not enough room in main store for the compiled program and its workspace.

solution: segment the program or put main store own arrays on core-backing store.

(b) A3Lerr4

cause: not enough room on core-backing store between BSLF and BSMIN (when second pass begins) for core-backing store own arrays.

solution: If core-backing store is used as described in chapter 1.2.2 the size of the reserved areas, the number or the size of core-backing store own arrays must be reduced.

(c) OCBS error BSfull

cause: not enough room on core-backing store for segments, i.e. first free location is equal to last free location.

solution: reduce the size of the reserved area.

5.3 Runtime

(a) MS NOROOM
MS NOROOM FOR MS ARRAY.
MS NOROOM FOR BS ARRAY.
VAC NOROOM MS ARRAY.

cause: the dynamic stack in main store requires more space than is available.

solution: (i) Segment the program
(ii) Place some of the arrays on core-backing store.

(b) BS NOROOM FOR BS ARRAY.
VAC NOROOM BS ARRAY

cause: the dynamic stack in core-backing store requires more room for core-backing store arrays than is available.

solution: (i) if possible, reduce number or size of core-backing store arrays,

or (ii) if segments are in core-backing store, place them on magnetic tape instead. This will involve modifying the executive so that it will cause A3L to pass segments to OAST instead of OCBS, and A3D to enter OAST to be supplied with segments. (See OAST, ALGOLB and GMT (for ALGOL3) descriptions, Chapter 7).

Chapter 6: OPERATING INSTRUCTIONS

1. OPERATING INSTRUCTIONS WITH THE CORE-BACKING STORE EXECUTIVE PROGRAM (ALGOLB)

1.1 Creating the system

1.1.1 Introduction

It is suggested that if magnetic tapes are available a batch holding the system is produced. When this batch is loaded into main store and core-backing store, the executive (called ALGOLB) and any programs that the user requires in main store during all phases of compilation and running will be in main store. A3C, A3L and A3D and their common programs will be in the bottom of core-backing store. For installations with no magnetic tape handlers, the program CBIT (see Chapter 7), has been produced to reduce the time taken to input the systems programs from papertape.

1.1.2 Operating instructions

Step	Typed Instruction	Message output by Typewriter	Tape in reader 1	Remarks
0	IN.	OCBS	OCBS	Input OCBS and any other programs required during all phases of compilation and running.
1	IN.	ALGOLB	ALGOLB	
2	IN.	ALPL	ALPL)	Common programs required by A3C
3	IN.	EDIT8	EDIT8)	
4	IN.	INTER	INTER)	
5	IN.	GMT	GMT)	Common programs required by A3C if programs with library calls are to be compiled and run.
6	IN.	LIBR 1	LIBR 1)	

2.1.5.6

Step	Typed Instruction	Message output by Typewriter	Tape in reader 1	Remarks
*7	IN.	PROGRAM NAME		Any other programs required during the first pass.
8	IN.	A3C	A3C	A3C may be input anywhere between steps 1 and 8.
9	ALGOLB;2.	BSMIN=	)	$0 \leqslant m \leqslant a-1$
10	m.n.	BSMAX=	)	$0 \leqslant n \leqslant 8191$
11	m.n.		)	where a = number of $\frac{1}{2}$ units of CBS. BSMIN must be >3 but see Section 4 note 2.
				If XRANGE is displayed see ALGOLB description 'ERROR MESSAGES'.
				ALGOLB now copies the section of main store above ALGOLB occupied by A3C and its common programs to core-backing store. All programs input after ALGOLB are now deleted from main store.
		END		
12	IN.	ALP	ALP)	Common programs required by A3D and A3L.
13	IN.	CHARIN	CHARIN)	
14	IN.	CHAROU	CHAROU)	
*15	IN.	PROGRAM NAME		Any programs wanted in store during run-time.
16	IN.	A3D	A3D	
17	IN.	A3L	A3L	
18	IN.	PROGRAM NAME		Any programs wanted in store during the second pass. If there are any auxiliary programs required only by A3L they should be input here.

Step	Typed Instruction	Message output by Typewriter	Tape in reader 1	Remarks
19	ALGOLB;2.	END		ALGOLB copies to CBS the section of main store above ALGOLB occupied by A3D, A3L and their auxiliary programs. All programs input after ALGOLB are now deleted from main store.
20	IN.	DUMP2	DUMP2	
21	DUMP2.NAME.	END		The created system should now be dumped on magnetic tape (previously prepared by DUMP2) as a main store and backing store batch called NAME.

NOTE: If ALGOLB is corrupted at any stage it is necessary to restart at step 1. This is because ALGOLB contains information about the positions and sizes on CBS of the main store copies and the current values of the backing store pointers.

* Step 7 may be made at any time between steps 2 and 7 inclusive.
Step 15 may be made at any time between steps 12 and 15 inclusive.

1.2 Compiling and running an ALGOL program

Typed Instruction	Message output by Typewriter	Tape in reader 1	Remarks
IN; NAME	END		Input the ALGOL3 main store and backing store batch from magnetic tape - prepared as described in Section 1.1.2 of this chapter. Handler 7 should be loaded with a library tape of texts (written up by LIBR2 - see Chapter 7.3.9). If there are library statements in the ALGOL program.
ALGOLB. n.	PROGRAM NAME FREE STORE a-b Dwait	SOURCE PROGRAM	See Section 3 of this chapter for key to n a = main store first free location. b = main store last free location.
	>	DATA TAPE	Change key 19 and the compiled program is entered. Alternatively, to enter the program interrupt and type ALGOLB. R.
ALGOLB. R.		DATA TAPE	ALGOLB enters the compiled ALGOL program.

Note It is not necessary to read down the batch containing ALGOLB and the systems programs each time an ALGOL program is run. It is only necessary initially and if ever the systems programs on core-backing store are overwritten in which case an error message will be displayed when an attempt is made to compile a new program.

2. OPERATING INSTRUCTIONS WITH THE MAGNETIC TAPE EXECUTIVE PROGRAM (ALGOLM)

2.1 Creating the system

2.1.1 Introduction

The ALGOL3 systems programs and the SAC common programs they require, are held as blocks on magnetic tape. They are written and read using the programs OAST and GMT (for ALGOL3). At the beginning of the tape are three dummy blocks (5 words long). Next on the tape are an information block and systems block for stage 1 followed by two similar blocks for stage 2. If OAST (for two handlers) is in store, the three dummy blocks will be preceded on the tape on handler 1 by a DUMP2 or DUMP2M batch which contains ALGOLM, OAST and GMT (for ALGOL3). This batch may be either preceded or followed by any number of DUMP2 or DUMP2M batches. Once the batches have been written up it is not recommended that selective overwriting takes place as this can lead to the ALGOL3 systems blocks being corrupted. Segments are also written to handler 1; these will be placed after the systems blocks.

If OAST (for 3 handlers) is in store the systems blocks and segments will be written to the tape on handler 5.

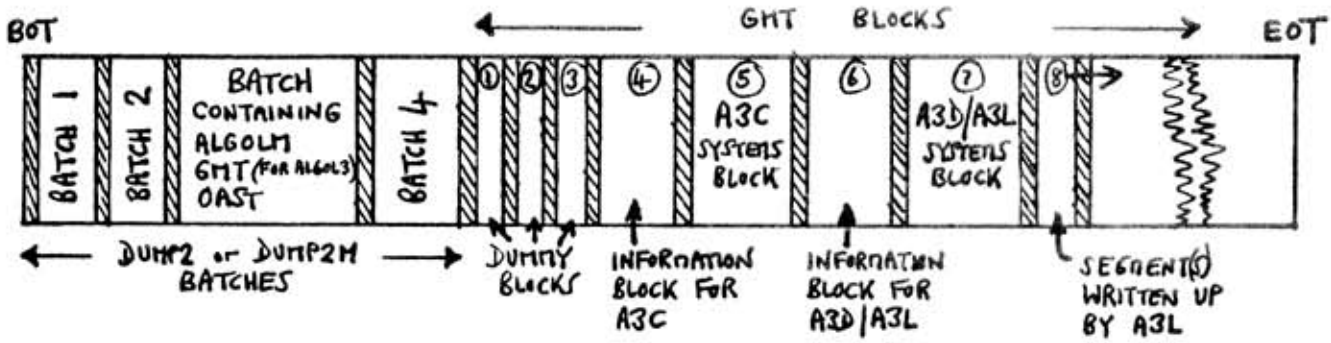


DIAGRAM OF MAGNETIC TAPE FORMAT - HANDLER 1
(with OAST (FOR 2 HANDLERS) IN STORE)

NOTE When OAST (for 3 handlers) is used, the format for the tape on handler 5 (which holds the systems blocks and segments) is essentially as above. The difference being that there are no DUMP2M or DUMP2 batches on the tape so that dummy block 1 is at the beginning of the tape.

Block numbering is as follows:-

Blocks 1-3		Dummy blocks
Block	4	Information block for the A3C block.
Block	5	A3C + auxiliary programs block.
Block	6	Information block for the A3L/A3D block.
Block	7	A3L and A3D and auxiliary programs block.

Blocks 8 and upwards are segments formed by A3L. The block numbering for segments starts at 8 each time a new program is compiled. Users with core-backing store may reserve part of the store (from location 4 upwards) for their own requirements (see Chapter 1.2.2) by typing in limits at steps 13 and 14 (see below) when creating the system.

2.1.2 Operating Instructions

Step	Typed Instruction	Message output by Typewriter	Tape in reader 1	Remarks
1				See Section 4, note 1
2	RESET.		-	RAPMT in store
3	IN	GMT	GMT (for ALGOL3)	
4	IN.	OAST	OAST	
5	IN.	PROGRAM NAME		Any programs required during all phases of compilation and running. (See Chapter 1.2.2).

Step	Typed Instruction	Message output by Typewriter	Tape in reader 1	Remarks
6	IN.	ALGOLM	ALGOLM	
7	ALGOLM;2.	END		First free location increased by one. This is to allow one spare location for GMT to write in the block number when stages 1 and 2 are written to magnetic tape.
8	IN.	DUMP2M	DUMP2M	
9	DUMP2M;2.NAME			Dump the main store as a batch called NAME on magnetic tape (handler 1) see note 4 in Section 4.
10	IN.	ALPL	ALPL)	Common programs required by A3C
11	IN.	EDIT8	EDIT8)	
12	IN.	INTER	INTER)	
13	IN.	LIBR1	LIBR1	Common program required by A3C if programs with library calls are to be compiled and run.
* 14	IN.	PROGRAM NAME		Any other programs wanted during the first pass.
15	IN.	A3C	A3C	
16	ALGOLM;3.			ALGOLM writes away three dummy blocks.
17		CBS OK		If core-backing store is available.
		NO CBS		If core-backing store is not available. In this case steps 18 and 19 are omitted.

Step	Typed Instruction	Message output by Typewriter	Tape in reader 1	Remarks
		BSMIN=		
18	m.n.			) $0 \leqslant m \leqslant a-1$
19	m.n.	BSMAX=		) $0 \leqslant n \leqslant 8191$) where a = number of $\frac{1}{2}$ units of CBS. BSMIN must be >3 but see Section 4 note 2. If XRANGE is displayed see ALGOLM description 'ERROR MESSAGES'.
20		END		ALGOLM now writes away:- a) an information block for stage 1. b) a block containing A3C, and the common programs input after the Executive.
21	IN.	ALP	ALP)	Common programs required by A3L and A3D.
22	IN.	CHARIN	CHARIN)	
23	IN.	CHAROU	CHAROU)	
24	IN.	PROGRAM NAME		Any programs wanted in store during run-time
25	IN.	A3D	A3D	
26	IN.	A3L	A3L	
27	IN.	PROGRAM NAME		Any programs wanted in store during the second pass only or by A3L only.
28	ALGOLM;4.			ALGOLM writes away:- a) an information block for stage 2. b) a block containing A3L, A3D and their common programs.
		END		

* Step 14 may be made at any time between steps 10 and 14 inclusive.

2.2 Compiling and running an ALGOL program

Typed Instruction	Message output by Typewriter	Tape in reader 1	Remarks
IN;NAME	END		Input an ALGOL3 batch containing the program GMT (for ALGOL3), OAST and ALGOLM - prepared as described in Section 2.1.2 of this chapter.
			Handler 7 should be loaded with a library tape of texts (written up by LIBR2 - see Chapter 7.3.9) if there are library statements in the ALGOL program. See also Section 4 note 1 for other tapes required.
ALGOLM.n.		SOURCE PROGRAM	See Section 3 of this chapter for key to n
	PROGRAM NAME FREE STORE a-b		
	Dwait		a is first free location b is last free location
	>	data tape	change key 19 and the compiled program is entered.
ALGOLM.R.		data tape	the run of the program last compiled is repeated.

3. CODE FOR COMPILING ENTRY TO THE EXECUTIVE

C. compile from reader 1

CE. compile from reader 1 and edit from reader 2.

CP. compile from reader 1 and print the ALGOL text on the lineprinter.

CEP. compile from reader 1, edit from reader 2 and print the program text on the lineprinter.

CL. compile from reader 1 and input library texts from magnetic tape.

CEL. compile from reader 1, edit from reader 2 and input library texts from magnetic tape.

CLP. compile from reader 1, input library texts and print the text of the program (in reader 1) on the lineprinter.

CELP. compile from reader 1, edit from reader 2, input library texts and print the text of the edited program (not including the library texts) on the lineprinter.

4. GENERAL OPERATING NOTES

NOTE 1 If OAST (for 2 handlers) is in store the following handlers are required:

HANDLER 1 Remote - write permit ring - tape with DUMP2 or DUMP2M batches. Used to hold segments and systems blocks.

HANDLER 4 Remote - write permit ring - scratch tape used to store owncode.

If OAST (for 3 handlers) is in store the following handlers are required:

HANDLER 1 Remote - batch tape (required initially if a batch containing GMT (for ALGOL3) OAST and ALGOLM is to be read into store).

HANDLER 4 Remote - write permit ring - scratch tape. Used to hold owncode.

HANDLER 5 Remote - write permit ring - scratch tape. Used to hold segments and systems blocks.

With both versions of OAST the tape on handler 4 is not needed once the ALGOL program is compiled.

NOTE 2 The values to be set in the pointers BSMIN and BSMAX are read as two integers m and n. The actual address placed in BSMIN is given by
(m x 8192) + n
i.e. m represents the integer number of $\frac{1}{2}$ units of core store and n is the address of a location inside a $\frac{1}{2}$ unit so that n satisfies $0 \leqslant n \leqslant 8191$.
e.g. to reserve the first two units of core-backing store type 4.0 for BSMIN.

NOTE 3 It is not necessary to read down the batch containing OAST, GMT (for ALGOL3) and ALGOLM each time an ALGOL program is run. The batch need only be brought down initially.

NOTE 4 The program DUMP2M would normally be used to dump the batch but DUMP2 should be used if information in the reserved areas of core-backing store is to be preserved. If DUMP2 is used it may be necessary to retrieve the dumped batch before it is possible to move onto the next stage of creating

the system (DUMP2 destroys the main store when certain entry points are used).
By dumping the main store as a batch, input of GMT (for ALGOL3), OAST and ALGOLM from papertape each compile time is avoided. In any case, these programs have to occupy the same position in main store as they did when the ALGOL3 system was created. Inputting from magnetic tape is the best way of ensuring this.

5. DUMPING AN ALGOL PROGRAM ON MAGNETIC TAPE

Because A3D continuously updates locations 7925 and 7926, it is possible to interrupt a running ALGOL program, input DUMP2 or DUMP2M and dump the main store (and core-backing store if it is being used to hold arrays) onto a magnetic tape batch.

To continue the run later, the batch should be read into store and

CONT.

typed.

However, DUMP2 ISSUE 1 and DUMP2M ISSUE 1 do not dump location 8176 with the batch written to magnetic tape, so that should a 76 instruction but not the corresponding 77 instruction have been obeyed when the program is interrupted, an error interrupt could occur when an attempt was made to continue the run of the dumped program.

Chapter 7: DESCRIPTIONS OF PROGRAMS USED IN ALGOL3

1. SYSTEMS PROGRAMS

1.1 A3C

Code RAP - binary

Function To convert syntactically correct ALGOL text to owncode.

Store used 4672 locations.

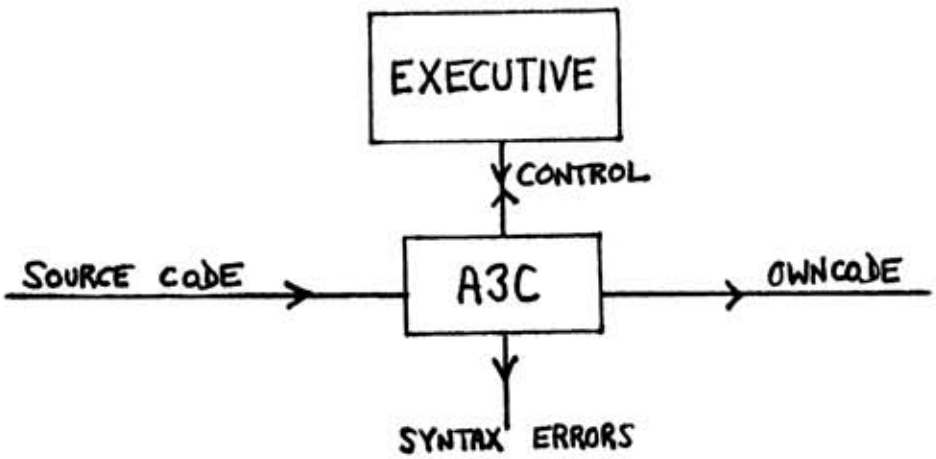
Method of use

A3C is used as a common program by the Executive.

To start the first pass of compilation, the Executive enters A3C to hand over the names and entry points of the three auxil iary programs which A3C will use for:-

(1) Input of ALGOL text or source code.
(2) Output of syntax error information.
(3) Output of owncode.

A3C then translates the source-code to owncode and finally exits to the Executive with an indication as to whether the translation was successfully completed.



2.1.5.7

Interface between the Executive and A3C

The three auxiliary programs are specified by three consecutive parameter words within the Executive. So that A3C can access these parameters, the Executive, on entry to A3C, plants the address of the location preceding the parameters in the LINKCP of A3C [i.e. the location immediately following the program head]. A3C will eventually exit to the 4th location after the address in LINKCP.

Parameters are of the form

$$00\ A_1 : 00\ A_2$$

where A_1 and A_2 are the addresses of the locations containing the name and entry point respectively.

e.g. 00 <9OCBS>:00<+2>

Interfaces between A3C and its auxiliary programs

ENTRIES

1. Source-code program.

   A3C enters this program to obtain one character. Exit should be made with the character in the accumulator.

2. Error-messages output.

   A3C enters with the character to be displayed in the accumulator.

3. Output of owncode.

   A3C enters with 39 bits of owncode in the accumulator.

EXIT

All exits from these programs should be made using the instruction

EXITCP

Entry to A3C

There is one entry point, and the instructions in the Executive are of the form

SUBR, A3C*ENTERCP
Source code parameter
error message parameter
owncode parameter

SUBR, A3C*ENTERCP is an entry instruction to a routine which finds A3C in store, plants the address of the instruction in its LINKCP, and then enters it. (See description of the Executive, 2.1).

Default Options

1. Paper tape reader 1.
   A3C will automatically read source-code from paper tape reader 1 if the A_1 address of the source code parameter contains zero.

2. Error-messages on typewriter.
   A3C will automatically print any error messages on the typewriter if the A_1 address of the error message parameter contains zero. The error message will be followed by 31 characters of the source code to enable the position at which the error occurred to be pin-pointed.

2.1.5.7

Exit from A3C

Immediately prior to the exit, A3C displays the name of the ALGOL program on the typewriter.

The exit instruction within A3C is effectively EXITCP,4 and the accumulator is set as follows:-

Accumulator zero: The source code was syntactically correct and translation to owncode has been successfully completed.

Accumulator non-zero:

Syntax errors were found or translation to owncode was not completed successfully (see error No.49).

Tapes

A tape coded in RAP binary form is supplied.

Error messages

A3C classifies syntax errors and outputs this information to the error message program (see 3.7 and 3.8 of this Chapter).

ERRerr

This message is displayed if A3C discovers that it is corrupt. The absence of this message does not mean that A3C is not corrupt.

1.2 A3L

Code RAP binary.

Function To convert owncode produced by A3C into machine code. The segments of a segmented ALGOL program are supplied to a common program for storage. The remainder of the machine code is placed in the main store.

Store used - 383 locations.

Method of use

For operating instructions see Chapter 6. A3L is used as a common program. It has one entry point which is used by the ALGOL3 executive. To start the second pass of compilation the Executive will enter A3L to hand over the names and entry points of the SAC common program A3L is to use. The common programs are required to perform two functions:-

(1) supply owncode to A3L

(2) if the ALGOL program is segmented, to store the segments formed by A3L.

Programs must be specified by the Executive to perform both functions as there are no default options in A3L.

A3L translates the owncode to machine code, and finally exits to the Executive; the value in the accumulator indicates whether or not the translation was successful.

2.1.5.7

PROCESS USED

Interface between Executive and A3L

The common programs to be used by A3L are specified by two consecutive parameter words within the Executive in the same way as is described in the description of A3C. The first parameter gives the name and entry point of the program to be used to supply owncode and the second gives the name and entry point of the program to be used to store segments.

Interfaces between A3L and its auxiliary programs

ENTRY

(1) Input of owncode.

A3L will enter this program to receive 39 bits of owncode. Exit should be made with the owncode in the accumulator.

(2) Output of segments.

When a segment has been built up in the main store, A3L will enter the auxiliary program with a parameter in the accumulator. The parameter is of the form:

00 a : 00 b

a is the segment length

b is the address of the start of the segment in main store.

The auxiliary program copies the segment onto backing store and will exit back to A3L with a parameter in the least significant 19 bits of the accumulator. This should enable the segment to be identified and will be one of the parameters which A3D passes to the auxiliary program which retrieves the segments at run time. (In the case of OCBS, which

sends segments to core-backing store, the 19 bit parameter is the address of the start of the segment on core store, and in the case of OAST, the parameter is the block number of the segment on magnetic tape).

EXIT

A standard exit should be made from the auxiliary programs, i.e. by the order

EXITCP.

ENTRY TO A3L

There is one entry point, and the instructions in the executive causing entry are of the form described in the A3C description (1.1 of this Chapter).

i.e. SUBR, A3L*ENTERCP
owncode input parameter
segment output parameter.

EXIT FROM A3L

The exit instruction within A3L is effectively

EXITCP, 3

The value of the accumulator upon exit from A3L is significant. The code is as follows:-

Accumulator	Meaning
0	The second pass has been successfully completed.
+1	The owncode supplied by A3L is incorrect (i.e. an error has been found in the syntax of the owncode). This implies an error in the systems programs, not in the ALGOL source program.

Accumulator	Meaning
+2	There is not sufficient room in the main store for the compiled program. (This may be corrected by introducing or increasing the program segmentation.)
+3	The value found by A3L in the core store pointer BSLF is not that found by A3C when translating the source code referring to core store own arrays (see Chapter 1. 2.2.2). This implies an error in the systems programs.
+4	There is not sufficient room between BSLF and BSMIN for the core store own arrays. (See Chapter 1.2.2).

TAPES

A tape coded in RAP-binary form is supplied.

ERROR MESSAGES

If a common program specified by the executive for use by A3L cannot be found in main store, the name of the program will be displayed, followed by "NOPROG". This will also occur if A3D is not in main store when A3L begins the second pass. A3L will then return control to the Executive with +1 in the accumulator.

1.3 A3D

Code RAP-binary.

Function

A3D consists of the dynamic routines and standard procedures required by the running ALGOL program.

Store used - 1760 locations.

Method of use

For Operating Instructions, see Chapter 6.

A3D is entered by the Executive when a translated ALGOL program is to be run. A3D is supplied by the Executive with the names and entry points of the auxiliary programs it is to use. It requires the common programs to perform six functions.

(1) To retrieve segments of a segmented program from backing store and place them in main store.

(2) To input characters from a specified input device. This is required by read statements and the procedures "instring" and "advance".

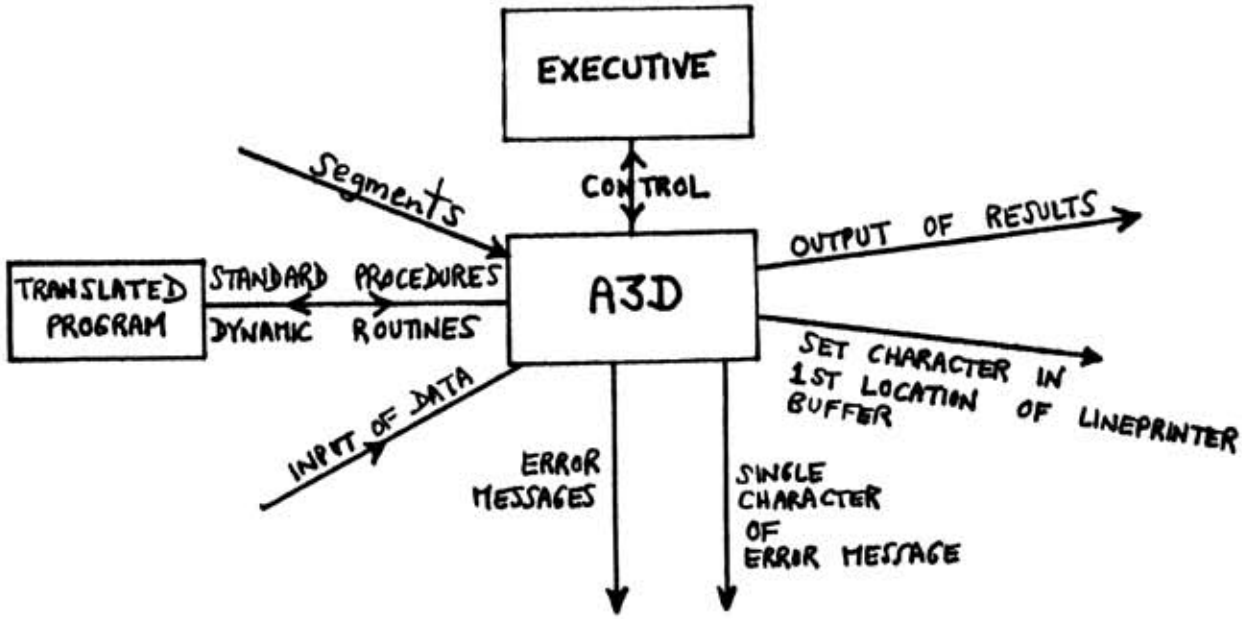
(3) To output characters to a specified output device. This is required by print statements and the procedure "outstring".

(4) To set a character in the first location of the lineprinter buffer. This is required by the procedures "top of form", "lines(m)", "find(m)" and "overprint".

(5) To output a single character of an error message, when the message cannot be output using (6) below.

(6) To output an error message when an error is detected in the running ALGOL program. The message is specified by a number passed to the program by A3D.

Programs must be specified by the Executive to perform all six functions as there are no default options in A3D.



PROCESS USED

Interface between A3D and Executive

The common programs to be used by A3D are specified by six consecutive parameter words within the Executive in the same way as is described in the description of A3C. The programs names and entry points appear in the order in which their functions are described in Method of Use above.

Interfaces between A3D and its auxiliary programs

ENTRY

(1) Retrieval of segments.

This program is entered when a segment has to be retrieved from backing store into main store. The segment and the position on main store in which it is to be placed are specified by two consecutive parameter words within A3D. So that the common

program may access these parameters, A3D on entry, plants the address of the location preceding the parameters in the LINKCP of the common program.

The two parameter words are of the form:-

00 L : N

00 0 : 00 M

where L is the length of the segment, N is the 19 bit parameter supplied to A3L when the segment was copied onto backing store (see A3L description, 1.2) and M is the starting address of the area in main store in which the segment is to be placed. Thus, the segment will occupy locations M to M+L-1 in main store.

Exit must be made from the common program by the order EXITCP,3.

See descriptions of OAST (3.10) and OCBS (3.11).

(2) Data input.

A3D enters this program to obtain one character from a specified input device. The input device number will be in the accumulator when entry is made.

The format of this parameter is the accumulator is:

00 n : 00 0

where n is the input device number and will satisfy $1 \leq n \leq 10$.

The character input must be in the least significant 7 bits of the accumulator upon exit. (See description of CHARIN 3.4).

(3) Output of results.

A3D enters this program to output one character to a specified output device. The character and the device number will be in the accumulator when entry is made. The format will be:-

n : 00 c

where n is the output device number which will satisfy $1 \leqslant n \leqslant 524$, 287 and c is the character to be output, (see description of CHAROUT, 3.5).

(4) Setting a character in the lineprinter buffer.

A3D uses a common program to set a character in the first location of the lineprinter buffer. This is required by the standard procedures "top of form", "lines(m)", "find(m)" and "overprint". The character will be in the least significant seven bits of the accumulator upon entry, (see description of CHAROUT, 3.5).

(5) Error message output, one character at a time.

Certain messages following the detection of an error in a running ALGOL program will be followed by diagnostic information that cannot be output using the method described in (6) below, e.g. the numbers and array names following the message "SUBSCR OFLO". This information is output, one character at a time, to an auxiliary program. The character will be in the least significant seven bits of the accumulator upon entry to the common program (see the description of CHAROUT, 3.5).

(6) Output of an error message specified by a number.

When A3D has detected an error in a running program, a common program will be entered with a number in the accumulator. This number specifies the message to be output. The correspondence between number and message to be displayed is listed in the description of CHAROUT, (3.5). PROCESS USED, entry point 5.

The action that will be taken by A3D after a continuable error message has been displayed is determined by the value of the accumulator upon return from the common program.

For details see CHAROUT (3.5), PROCESS USED, entry point 5.

EXIT

Exit from the common programs is made by the order EXITCP except in the case of the program which retrieves segments which exits by the order

EXITCP, 3.

EXIT FROM A3D

The exit instruction within A3D is effectively

EXITCP, 7.

The value of the accumulator upon exit is significant.

Value in Accumulator	Meaning
zero	The program has run to its conclusion or a non-continuable error has been detected.
positive	The program is to be re-run. This occurs when the procedure "restart" is called in an ALGOL program.

TAPES

A tape coded in RAP-binary form is supplied.

ERROR messages

If a common program specified by the Executive for use by A3D cannot be found in mainstore, the name of the program will be displayed followed by "NOPROG". This also occurs if the program which is to be entered using the procedure "entercp" cannot be found. A3D will then return to the Executive with +0 in the accumulator.

2. EXECUTIVE PROGRAMS

2.1 ALGOLB

Code SAC

Function

ALGOLB is the Executive program for the ALGOL system in an installation with core-backing store (CBS).

Facilities

ALGOLB contains facilities for

(1) Initial storage of the components of the system (A3C, A3L, A3D and their auxiliary programs) on CBS.

(2) Supervision of A3C, A3L, A3D: allocation of auxiliary programs and the detection of error conditions.

(3) Distribution of space on CBS, loading of the appropriate components of the system into main store during the two stages of compilation, and, where necessary, the initialisation and close-down of auxiliary programs.

(4) Communication between the computer operator and the system.

Store used - 327 locations, including workspace.

Method of use

There are three phases in the life of an ALGOL program: Compilation (pass 1), Compilation (pass 2) and Run-time. The system is arranged so that during any phase only those programs necessary for that phase are in main store. Before any programs can be compiled the system has to be created; it has to be decided which facilities are to be included, (library, edit, listing etc.) and the programs loaded into CBS. Eventually the system will be overwritten either by the space demands on CBS of a compiled program or because the computer will be used for purposes other than the running of ALGOL programs. In order that it should be unnecessary to re-make the system, it is envisaged that the main store and the appropriate part of CBS will be written as a batch to magnetic tape.

To run a series of programs, the previously created batch is read down from magnetic tape and ALGOLB entered (see below).

Core-backing store (CBS) pointers

The first four locations of CBS are used by the system to indicate the extent of CBS available to the ALGOL system and the current

first-free and last-free locations.

Location	Name	Content
0	BSFF	address of first-free location.
1	BSLF	address of last-free location.
2	BSMIN	address of first location available to system.
3	BSMAX	address of last location available to system.

These pointers are available for use by any auxiliary programs (e.g. OCBS).

The settings of BSMIN and BSMAX define the limits of CBS for the ALGOL system. It is thus possible to reserve space both at the bottom and the top of CBS for any use whatsoever outside the ALGOL system, e.g. by a SAP-program which is used by the compiled ALGOL program via the entercp procedure (see Chapter 4.8).

Creation of system

Part 1 (i) Read into main store any programs which are required to be in main store during both compilation and running of ALGOL programs.

(ii) Read ALGOLB into main store.

The main store now looks like:-

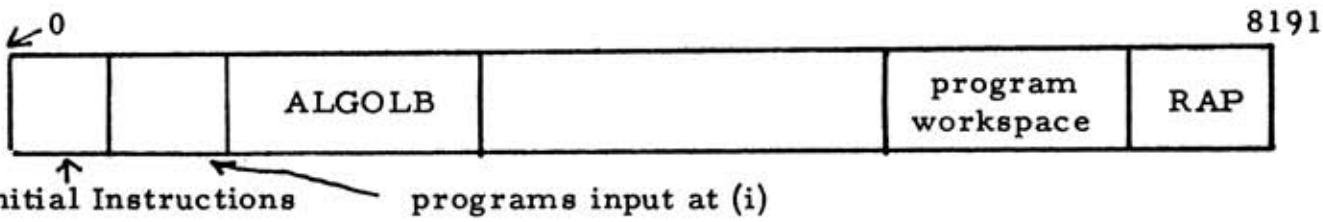


Fig.1

Note In general there are no programs in the category described in (i), thus ALGOLB will usually be the first program in store.

ALGOLB and any programs input before it, remain in main store and are not copied to CBS.

Part 2 This is split into two stages corresponding to the two passes of compilation.

(a) Stage 1

(i) Read into main store the programs needed for the first pass of compilation, i.e. A3C and its auxiliary programs [A3C may be input at any point].

(ii) Enter ALGOLB by typing

ALGOLB;2.

ALGOLB detects that A3C is in store and checks that A3L and A3D are not present. It then reads from the typewriter the values of BSMIN and BSMAX as follows:-

Message from ALGOLB	Action
BSMIN=	type the address (see below) of the first location of CBS available to the ALGOL system.
BSMAX=	type the address (see below) of the last location of CBS available to the system.

[The CBS is supplied in units of 16384 locations, thus each unit is twice the size of the main store. It is convenient to consider CBS addresses in terms of mair store sizes. The address must be given in the form

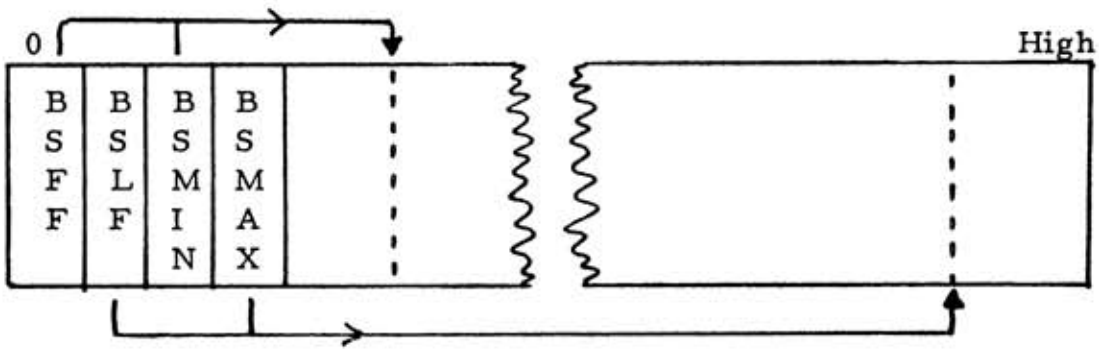
x. y.

where x represents the number of times 8192 can be divided into the address and y is the remainder.
e.g. if there are four units of CBS attached to the computer and all of the CBS is available to the ALGOL system then BSMIN = 0.4.
and BSMAX = 7.8191.]

ALGOLB checks BSMIN $\geqslant$ 4

BSMAX $\geqslant$ BSMIN

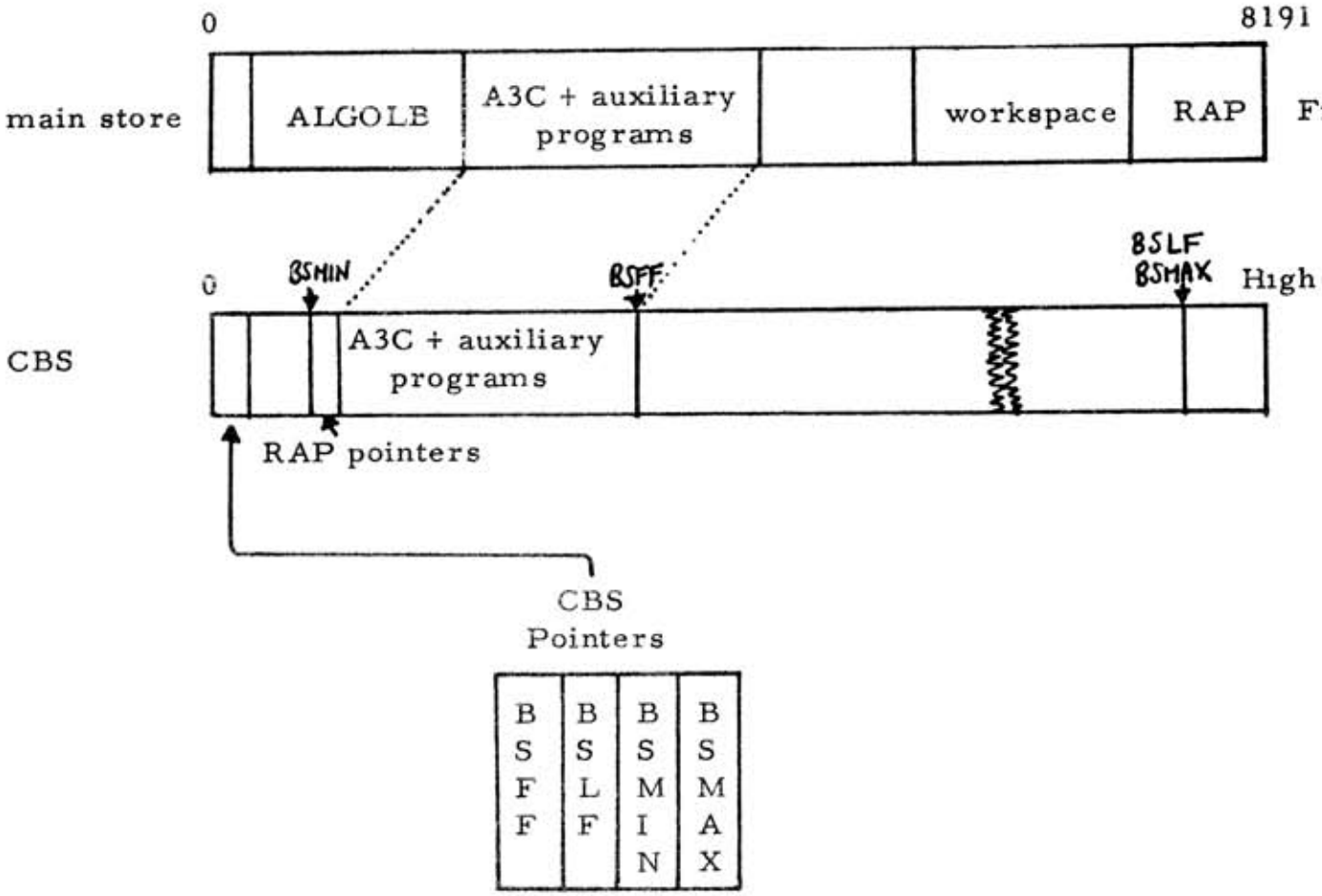
and that the address specified for BSMAX exists. If these conditions are satisfied the backing store pointers are set up (see Fig.2).



CBS

Fig.2.

ALGOLB now copies the section of main store occupied by A3C and the auxiliary programs to CBS together with RAP pointers (see below), specifying the current state of the main store, and updates BSFF. It also notes the position and length of the copy of main store on CBS.



ALGOLB now deletes from main store all the programs input after it (i.e. A3C, etc.), so that the main store is now as in Fig.1.

Stage 2 of the system is now prepared.

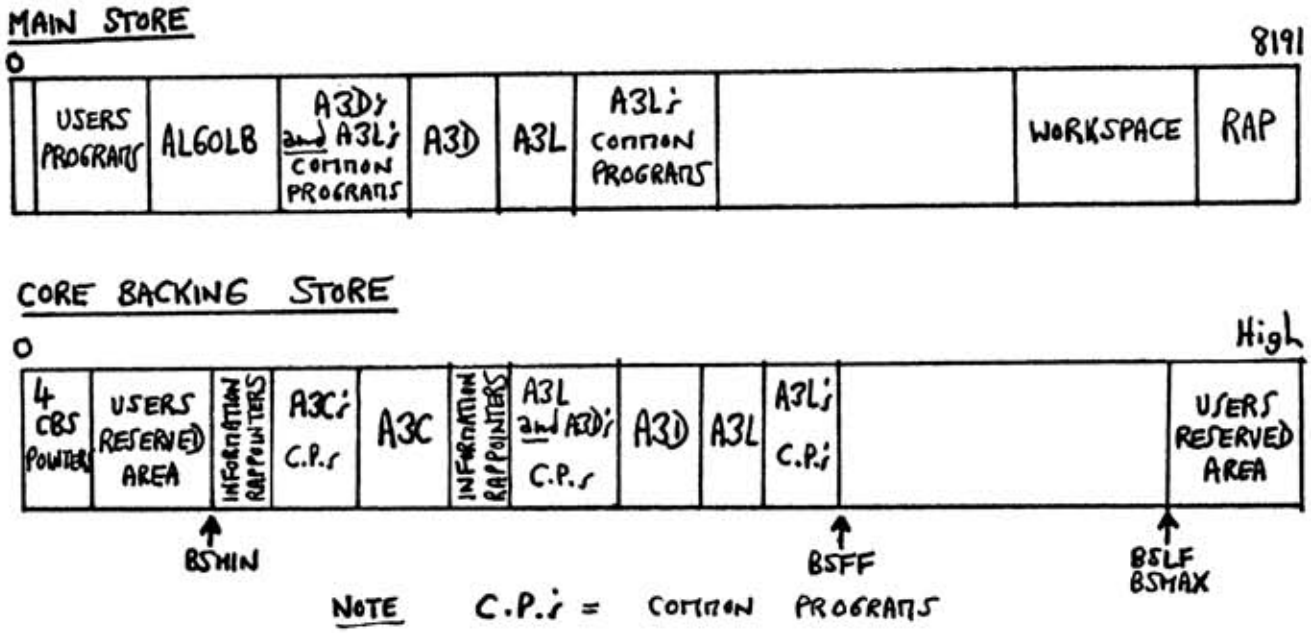
(b) Stage 2

(i) Read into the main store prepared by stage 1 the programs needed during the running of an ALGOL program, i.e. A3D and its auxiliary programs, followed by A3L and then its auxiliary programs (naturally programs used by both A3D and A3L need only be input once if they are input before A3L).

(ii) Enter ALGOLB by typing

ALGOLB;2.

ALGOLB detects that A3C is not present and checks that both A3D and A3L are in store. (Though this is the same entry point as in stage 1 (ii), the effect is different because A3C is not now in store). ALGOLB now copies the section of the main store occupied by A3D, A3L etc. to CBS together with the RAP pointers (see below), specifying the current state of main store, and updates BSFF. It also notes the position and length of the copy of main store on CBS (see Fig.4).



ALGOLB now deletes from main store all the programs input after it (i.e. A3D, A3L etc.), so that the main store once again appears as in Fig.1.

Notes on auxiliary programs

Since the names and entry points of these programs must be stored within ALGOLB any change of name or entry point will require an alteration to ALGOLB.

Part 3 The system is now complete and is dumped as a batch onto magnetic tape. It is only necessary to dump that part of CBS up to the current value of BSFF, which includes the main store copies. The backing store pointers can be examined using the library program PML (see Section 2.2.3.30 of the Manual).

Note that ALGOLB contains information regarding the positions and sizes on CBS of the main store copies and the current values of the backing store pointers. Therefore, it must not be deleted from store. In fact should it be corrupted during any of the preceding stages it will be necessary to restart at Part 1.

Operation of system

Part 1 Input of the system.

The system batch is read down from magnetic tape. Programs may now be compiled, and it will not be necessary to re-read the batch unless so much core-backing store is required by a running program that the system is over-written or if ALGOLB is corrupted. The main store appears as in Fig.1 and the CBS as in Fig.4.

Part 2 Compilation.

To compile a program, ALGOLB is entered by typing ALGOLB.
followed by a series of letters which denote the form of compilation required (see ENTRY POINTS).

(i) First Pass.

ALGOLB resets the backing store pointers (in case this is not the first program to be compiled) and copies A3C and its auxiliary programs from CBS to main store and also resets the RAP pointers. The main store now appears as in Fig.3.
Auxiliary programs are initialised if necessary and control passes to A3C.

(ii) Second Pass.

After the first pass, A3C hands control back to ALGOLB, and if the pass has been error-free A3D, A3L etc. are copied from CBS and the RAP pointers reset. The main store now appears as in Fig.4.
A3L is entered and returns control to ALGOLB when the second pass is finished. If the owncode has been loaded correctly, A3L is deleted (thus any programs in the RAP list originally input after A3L are also deleted) and the compiled program is ready to be entered.

Note Compilation may be stopped at any point and started afresh.

Part 3 Running the compiled program.

(i) Input of binary programs.

In order that a running ALGOL program may be interrupted at any point and repeated, ALGOLB notes the current values of a BSLF and main store last free pointer on the first entry to the compiled program. This means that any SAP-binary programs needed by the running ALGOL program, (as opposed to the auxiliary programs used by A3D) must be input before the program is first entered.

(ii) Entry and exit.

To enter the compiled program ALGOLB passes control to A3D together with the names and entry points of its auxiliary programs. A3D then enters the compiled program and control does not return to ALGOLB until the end of the program is reached or the program obeys a restart procedure or a non-continuable error occurs.

If the program requests a restart, ALGOLB hands control back to A3D, otherwise control is finally returned to RAP.

RAP pointers

The state of the main store is indicated by the following three locations:-

7920	RAP-pointer	points to the head of the first program in the RAP chain (i.e. last one input).
7925	first-free)	indicate the first-free and
7926	last-free)	last-free location in store respectively.

ENTRY POINTS

To create the system type ALGOLB;2. (See CREATION OF SYSTEM).

To compile an ALGOL program type ALGOLB, followed by a series of letters. (See Chapter 6.3 for details).

To repeat or restart a compiled ALGOL program type ALGOLB.R.

Alterations may be made by individual installations to the Executive and the method of entry to suit their particular requirements, (e.g. for batch processing). To delete a program from store, together with all programs input after it where NAME is the name of the program, type

ALGOL.B;3.NAME.

MESSAGES

(1) During creation of the system

Message	Meaning
EXEC A3 err	Either A3C is in store with A3D and/or A3L, or A3D and A3L are not both in store. Restart the appropriate stage.

Message	Meaning
BSMIN= BSMAX=	) the addresses of the first and) last locations of CBS available to the ALGOL system are to be typed.
XRANGE	An incorrect address has been typed for BSMAX or BSMIN, i.e. BSMIN <4 or BSMAX<BSMIN or BSMAX does not exist. Retype the correct value(s).
EXEC error BSOFLO	BSFF>BSLF. Will occur if there is not enough room on CBS during stage 1 or stage 2. Creation of system will have to be restarted.

(2) During compilation

Message	Meaning
NORUN [A3C exits with accumulator non-zero]	Syntax errors are present in the ALGOL text. Compilation stops and control returns to RAP.
A3Lerr1	Error detected in owncode. Some part of the system is corrupted, (see A3L description - EXIT FROM A3L).
A3Lerr2	Not enough room in main store to hold the compiled program. (See A3L description - EXIT FROM A3L).
A3Lerr3	Value of BSLF used for own backing store arrays has been altered when A3L tests it. (See A3L description - EXIT FROM A3L
A3Lerr4	Not enough room on CBS for own backing store arrays. (See A3L description - EXIT FROM A3L

Message	Meaning
FREE STORE (followed by the freestore limits)	The limits of the main store available to the compiled program are displayed.
Dwait	The compiled program may now be entered. ALGOLB enters a sign key loop. When key 19 of the word generator is changed ">" is displayed and the compiled program is entered. Alternatively, to enter the compiled program, interrupt and type ALGOLB.R. (see ENTRY POINTS).
EXEC err BSOFLO	BSFF>BSLF, auxiliary programs which use CBS have not carried out their own tests and an overflow condition has occurred.
EXEC err CBSerr	BSFF<BSMIN or BSLF>BSMAX The backing store is corrupt or the backing store pointers have been corrupted.

(3) On entry to ALGOLB to compile a program

Message	Meaning
RELOAD SYSTEM	The main store copies on CBS have been overwritten. The system batch on magnetic tape will have to be read down.

(4) At any time

NOPROG (preceded by a program name)	The named program is required by the system but is not in store.

ENTRY TO AUXILIARY PROGRAMS

Since ALGOLB will be input to store before any auxiliary programs, it is not possible to use the SAP macro COMP to enter auxiliary

programs. ALGOLB therefore, contains a subroutine ENTERCP which exactly replaces COMP.

Example

To use ENTERCP to enter a SAC program called INTER at entry point 3 write in ALGOLB the instructions

```
30 <9INTER>:00<+3>
SUBR,ENTERCP
```

These instructions are equivalent to

```
COMP,INTER,3
```

A3C, A3L and A3D are RAP-binary programs (i.e. not written in symbolic assembly code). There are specified entries in ENTERCP for these programs viz:

```
SUBR,A3C*ENTERCP
SUBR,A3L*ENTERCP
SUBR,A3D*ENTERCP
```

TAPE

ALGOLB is issued as a SAC mnemonic tape. Individual installations will have to make their own binary versions. The program, as issued, is sum-checkable.

2.2 ALGOLM

Code SAC 1

Function

Executive program to control the ALGOL3 systems and SAC programs in a magnetic tape installation.

2.1.5.7

Configuration

503 + 2 magnetic tape handlers.

Tape

A tape coded in SAC is supplied; for use with ALGOL3 a binary tape must be assembled using SAP 1 with keys 35 and 36 depressed to prevent a checksum being formed.

Store used

360 locations, including workspace.

Method of use

All entries are from the keyboard.

Entry point 1

Type

ALGOLM.n.

(n is C for compiling. CE for edit and compile, etc. - see Chapter 6.3) with the ALGOL source program tape in reader 1.

Entry point 2

Type

ALGOLM;2.

to make the first free location one greater.

Entry point 3

Type

ALGOLM;3.

to write to magnetic tape that part of the main store above ALGOLM up to, but

excluding the location whose address is in location 7925, (see PROCESS USED).

Entry point 4

Type

ALGOLM;4.

to write to magnetic tape that part of the main store above ALGOLM up to, but excluding the location whose address is in location 7925, (see PROCESS USED).

Entry point 5

Type

ALGOLM;5. NAME

to delete a program NAME from main store together with all programs input after it.

Process used

The ALGOL3 system for magnetic tape has been designed for use with two or three handlers depending on which version of OAST is in store. If OAST (for 2 handlers) is in store, a DUMP2 or DUMP2M batch must be at the beginning of the tape on handler 1.

This is because GMT will always look for the last batch before writing blocks (systems blocks and segments) to handler 1. Owncode is written to the tape on handler 4. If OAST (for 3 handlers) is in store the systems blocks and segments are written to handler 5 and owncode to handler 4. It is suggested that with either version of OAST in store, a batch is held on magnetic tape consisting of the program GMT (for ALGOL3), OAST and ALGOLM, assembled in that order. These three programs must be in main store when the system is created and each time a program is compiled they must occupy the same locations. For this reason it is advisable to have the programs on a magnetic tape batch.

Creating the system

(1) (a) Any programs required in main store during both compilation and running of ALGOL programs should be read into store.

(b) GMT (for ALGOL3), OAST and ALGOLM are read into store in that order.

(c) Type

ALGOLM;2.

to increase by one the address in the first free location. This is to provide one location in which GMT will write a block number when the systems programs are written to magnetic tape.

(d) Input DUMP2 or DUMP2M and dump the main store as a batch to magnetic tape. Read back this batch to main store, if the contents of main store were destroyed when the batch was dumped.

(2) (a) The programs required for the first pass of compilation are read into store, i.e. A3C and its auxiliary programs.

(b) Type

ALGOLM;3.

ALGOLM detects that A3C is in store and that A3D and A3L are not present. It then writes three dummy blocks of five words each to magnetic tape. (If OAST (for 2 handlers) is

in store the program GMT ensures that these blocks are written beyond the DUMP2 or DUMP2M batches on handler 1. If OAST (for 3 handlers) is in store the dummy blocks are written to handler 5. The dummy blocks are written up to avoid trouble should it be necessary to retreat in reading or writing from block 4 onwards on handler 1, i.e. in case OAST (for 2 handlers) is in store.

If there were no dummy blocks it would be possible for the handler to retreat into the DUMP2 batch area in which blocks are not numbered.

At this stage ALGOLM tests to see whether or not core-backing store is present and sets a marker appropriately. If core-backing store is available the limits (see ALGOLB, description 2.1, CREATION OF SYSTEM) of the region available to the ALGOL system are read from the typewriter and checked.

ALGOLM now writes to tape a 9-word information block, numbered 4. This contains the following details:-

00	BLOCK NUMBER	: 00	CONTENTS OF CBS MARKER
00	LENGTH OF STAGE 1	: 00	START ADDRESS OF STAGE 1 IN MAIN STORE
00	0	: 00	CONTENTS OF RAPPOINTER(7920)
00	0	: 00	CONTENTS OF RAP FIRST FREE(7925)
00	0	: 00	CONTENTS OF RAP LAST FREE(7926)
00	0	:	CONTENTS OF CBS FIRST FREE

00	0	:	CONTENTS OF CBS LAST FREE
00	0	:	CONTENTS OF CBS MIN.
00	0	:	CONTENTS OF CBS MAX.

Stage 1 of the system (the section of main store above ALGOLM which consists of A3C and its auxiliary programs) is then written to magnetic tape as block number 5.

ALGOLM now deletes from main store all the programs input after it (i.e. A3C, etc.) so that the main store is now as it was when ALGOLM was read into store.

(3) (a) Programs required during the running of an ALGOL program are read into store (i.e. A3D and its auxiliary programs), followed by those required during the second pass of compilation (A3L and its auxiliary programs).

(b) Type

ALGOLM;4.

ALGOLM detects that A3C is not present in store and that both A3D and A3L are in store. Another 9-word information block containg the details mentioned in 2(b) above, (but relating to A3D, A3L, etc.), is written to tape as block number 6. This is followed by block 7 which comprises the section of main store above ALGOLM up to and including the last program read in, i.e. A3D, A3L and their common programs. ALGOLM now deletes from main store all the programs input after it.

Operation of system

1. Compilation

To compile a program, type ALGOLM. followed by a series of letters which specify the form of compilation required (see section 3 of chapter 6).

(a) First Pass

ALGOLM first reads from magnetic tape the information block for stage 1 (i.e. A3C, etc.). The values of first free pointer, last free pointer, etc. in main store and core-backing store are then set to the values held in the information block, i.e. restoring the values to those that existed before A3C and its common programs were written to magnetic tape. The systems block for A3C and its common programs is then read into store immediately above ALGOLM. Auxiliary programs are initialised if necessary and control passes to A3C.

(b) Second Pass

A3C returns control to ALGOLM and if the pass has been error-free, the information block and then the systems block for stage 2 (A3D and A3L) are read from magnetic tape. A3D, A3L and their common programs now lie in main store immediately above ALGOLM; the RAP pointers in main store are reset to

the values held in the information block in the same way as for stage 1.
The core-backing store pointers are not altered. A3L is entered and returns control to ALGOLM when the second pass is finished. If the owncode has been loaded correctly, A3L is deleted (thus any programs in the RAP list originally input after A3L are also deleted) and the compiled program is ready to be entered.

Note Compilation may be stopped at any point and started afresh.

(c) Running the compiled program

(i) Input of binary programs.

In order that a running ALGOL program may be interrupted at any point and repeated ALGOLM notes the current values of BSLF and main store last free pointers on the first entry to the compiled program. This means that any SAP-binary programs needed by the running ALGOL program (as opposed to the auxiliary programs used by A3D) must be input before the compiled ALGOL program is first entered.

(ii) Entry and Exit.

To enter the compiled program ALGOLM passes control to A3D together with the names and entry points of its auxiliary programs. A3D then enters the compiled

program and control does not return to ALGOLM until the end of the program is reached or the program obeys a restart procedure or a non-continuable error occurs.

If the program requests a restart, ALGOLM hands control back to A3D, otherwise control is finally returned to RAP.

RAP pointers

The state of the main store is indicated by the following three locations:-

7920	RAP pointer	points to the head of the first program in the RAP chain (i.e. last one input).
7925	first-free	indicate the first-free and last-free location in store respectively.
7926	last-free	

Error messages

Message	Condition and Action
EXEC A3 err	(a) A3C is in store with A3D and/or A3L or (b) A3D and A3L are not both in store. Restart the appropriate stage.
NORUN	There is an error in the source code. Compilation stops.
A3L err1	There is an error in the owncode retrieved from magnetic tape.

Message	Condition and Action
	Some part of the system is corrupt. No continuation possible (see A3L, description, EXIT FROM A3L).
A3L err2	There is not enough room in main store to hold the compiled program. No continuation is possible. (See A3L description - EXIT FROM A3L).
A3L err3	The value of backing store last-free location used for own arrays on CBS is not what A3C assumed. No continuation is possible, (see A3L description - EXIT FROM A3L).
A3L err4	There is not enough room in core-backing store for own arrays. No continuation is possible, (see A3L description - EXIT FROM A3L).
XRANGE	The limits typed in for BSMIN or BSMAX are out of range. Program prints BSMIN or BSMAX again and waits for new limit.
PROGRAM NAME NOPROG	The named program is required by the system but is not in store.

Entry to auxiliary programs

ALGOLM uses the same method as ALGOLB (see description), to enter auxiliary programs, although entries to OAST, which has to be input before ALGOLM in any case, are made in the usual way, i.e. COMP, OAST.

3. AUXILIARY PROGRAMS

3.1 Introduction

The programs whose descriptions follow, are used as common programs by A3C, A3L, A3D and the Executives ALGOLB and ALGOLM, (see Chapter 1.2). By modifying these common programs, local versions can be produced, best suited to the requirements and configuration of a particular installation. Where appropriate, the descriptions include the format of the parameters with which A3C, A3L, A3D and the Executives will enter the common programs and the format of the parameters which are expected to be returned.

The auxiliary programs should be written in SAC1. The systems programs A3L, A3D, A3C form (using the program names and entry points supplied by the Executive), the instructions which will enable them to enter their auxiliary programs directly. To do this, it has been assumed that the head and trigger list of the auxiliary programs are of the form attached to a program assembled under SAP1.

3.2 ALP

Code SAC1

Function

Common program to output characters on the lineprinter.

Configuration - 503 + lineprinter.

Tape

A tape coded in SAC is supplied. For use with ALGOL3 a binary tape must be assembled using SAP1 with key 36 depressed.

Size - 208 locations, including workspace.

2.1.5.7

Method of use

A standard common program entry and exit is made.

Entry point 1

This is the initialisation entry point. The buffer is cleared and pointers set so that the next character input, though entry point 2 will eventually be output at the beginning of a lineprinter line. The first location of the buffer is set to give output on a newline. The entry call is:-

```
COMP, ALP, 1
```

Contents of the accumulator on entry and exit are undefined.

Entry point 2

```
COMP, ALP, 2
```

The character in the accumulator is placed in the next location of the buffer.

On exit the contents of the accumulator are undefined.

Entry point 3

```
COMP, ALP, 3
```

The character in the accumulator is stored in the first location of the buffer, i.e. line spacing can be specified for the next line to be output (see Section 1.4.3 of the Manual).

Output format

The maximum number of characters to a line is 120. The characters in the buffer are output in accordance with the lineprinter code (see Section 1.4.3 of the Manual), with the following exceptions:-

(1) Newline characters (Decimal 2) will output the buffer. If the buffer is full and no newline character is received the buffer is output. When a newline character is received the characters in the buffer are shifted so that output is on the right-hand side of the next line.

(2) Tab character (Decimal 4). Six spaces are output.

(3) Lower case characters (Decimal 97-122). These characters are all converted to upper case characters on output.

(4) (Decimal 123-127). These characters are ignored.

Vertical format

Output is on the next line unless the first buffer location has been set at entry 3.

Error message

The message:-

ALP error LPErr

is displayed, if the lineprinter becomes unavailable. If the error state is cleared the run will continue automatically.

3.3 ALPL

Code SAC1

Function

A. Common program to output characters on the lineprinter.

B. To list on the lineprinter a tape placed in reader 1 until the end of tape is reached or a halt code character (Decimal 76) is read.

Configuration - 503 + lineprinter.

Tape

A tape coded in SAC is supplied; for use with ALGOL3 a binary tape must be assembled using SAP1 with key 36 depressed.

Size - 275 locations, including workspace.

Method of use

Function A

A standard common program entry and exit is made.

Entry point 1

This is the initialisation entry. The first location of the lineprinter buffer is set to give output on a newline. The entry is:-

COMP, ALPL, 1

Contents of the accumulator on entry and exit are undefined.

Entry point 2

COMP, ALPL, 2

The character in the accumulator is placed in the next location of the buffer.

On exit the contents of the accumulator are undefined.

Entry point 3

COMP, ALPL, 3

The character in the accumulator is stored in the first location of the buffer, i.e. line spacing can be specified for the next line to be output, (see 503 Manual, 1.4.3).

Function B

Entry point 4

This is the keyboard entry point. Load the tape to be listed in reader 1 and type

ALPL; 4.

Entry point 5

Type

ALPL; 5.

to output the contents of the buffer.

Last line output (Common program entries i.e. Entry point 2)

The buffer is output if a newline character (Decimal 2) or halt code character (Decimal 76) is met. A standard common program exit is then made.

Last line output (Keyboard entries i.e. Entry point 4)

If key 39 of the word generator is clear when a halt code character (Decimal 76) is met the program will output the buffer and stop.

If key 39 is depressed, the program will output the buffer when a halt code is met and then continue to read characters from reader 1.

2.1.5.7

Output format

The maximum number of characters to a line is 120. The maximum number of lines output on each page is 55. As soon as this number has been output a top of form instruction is given.

The characters in the buffer are output in accordance with the lineprinter code, (see Section 1.4.3 of the Manual) with the following exceptions:-

(1) Newline character (Decimal 2) will output the buffer. If the buffer is full and no newline character is received the buffer is output. When a newline character is received the characters in the buffer are shifted so that output is at the right-hand side of the new line.

(2) Tab character (Decimal 4). Six spaces are output.

(3) Upper-case characters (Decimal 33-59). A vertical bar is output on the next line beneath the character to be printed.

(4) Vertical bar character (Decimal 62). The next significant character in the buffer will be overprinted by a vertical bar.

(5) Halt code character (Decimal 76). The buffer is output. If entry point 4 has been used the word generator is read (see above).

(6) Underline character (Decimal 126). A minus sign (Decimal 30) is output on the following line underneath the appropriate character.

(7) Decimal characters 123-125, 127 are ignored.

Vertical format

Output is on the next line unless the first buffer location has been set at entry 3 or unless 55 lines have been printed on a page. When this number is reached a top of form instruction is given.

Error message

The message:-

ALPL error LPErr

is displayed if the lineprinter becomes unavailable.

If the error state is cleared the run will continue automatically.

3.4 CHARIN

Code SAC1

Function

To input one character from a specified input device. CHARIN is used as a common program by A3D.

Store used - 21 locations, including workspace.

Method of use

CHARIN can only be used as a common program and has one entry point. A parameter specifying the input device to be used must be in the accumulator when entry is made, (see PROCESS USED for the format of the parameter). A standard exit is made, i.e. by the order EXITCP.

2.1.5.7

Process used

Entry is made with a parameter of the form

00 n : 00 0

in the accumulator. n is a number which specifies the input device to be used and will satisfy $1 \leqslant n \leqslant 10$ when used with ALGOL3. The correspondence between device number and input device is given below.

n	input device used
1	tape reader 1
2	tape reader 2
3	typewriter
4 to 10	tape reader 1

One character is input and this is in the least significant seven bits of the accumulator when exit is made.

Tapes

A mnemonic SAC tape is provided; a binary version should be produced using SAP1. The program is sumcheckable.

3.5 CHAROUT

Code SAC

Function

CHAROUT is used as a common program by A3D. It has three functions.

(1) To output a character to a specified output device.

(2) To output error messages originating in A3D.

(3) To set a character in the first location of the lineprinter buffer so that the line on which printing occurs may be controlled, (see Section 1.4.3 of the Manual).

Store used - 164 locations, including workspace.

Method of use

CHAROUT can only be used as a common program and has six entry points. A standard exit is always made, i.e. by the order EXITCP

Entry point 1

This is an initialisation entry point, used by the ALGOL compiler Executive, (see PROCESS USED).

Entry point 2

This is used by A3D to output one character to a specified output device. A parameter consisting of the character and output device number will be in the accumulator when entry is made, (see PROCESS USED for parameter format).

Entry point 3

This is used by the ALGOL3 Executive when the run of an ALGOL program has been terminated, (see PROCESS USED).

Entry point 4

This is used by A3D to set a character in the first location of the lineprinter buffer. The character will be in the accumulator, when entry is made (see PROCESS USED).

Entry point 5

This is used by A3D to display a single character of an error message. The character will be in the accumulator when entry is made, (see PROCESS USED).

Entry point 6

This is used by A3D to display error messages. The message is specified by a number which will be in the accumulator when entry is made, (see PROCESS USED).

Process used

Entry point 2

The parameter that will be in the accumulator when entry is made, is of the form

n : 00 c

where c is the character to be output and n is the number which specifies the output device to be used. It has the value of the device number parameter used in the call of the procedure "punch(n)" in the ALGOL program. n will satisfy $1 \leqslant n \leqslant 524,287\ (2^{19}-1)$. The output device actually associated with a device number is determined by CHAROUT and for CHAROUT ISSUE 1, the correspondence is given below.

n	device used
1	tape punch 1
2	tape punch 2
3	typewriter
4	lineprinter
5 to 524,287	tape punch 1

CHAROUT uses the program ALP to have characters output on the lineprinter (see 3.2).

Entry points 1 and 3

The ALGOL3 Executive will enter CHAROUT at entry point 1 before an ALGOL program is run so that any initialisation required may be performed. In the case of CHAROUT ISSUE 1 this consists of initialising the lineprinter program ALP and of clearing a marker. This marker will be set if a character is output to the lineprinter during the running of an ALGOL program. CHAROUT is entered at entry point 3 when the ALGOL program has run to its conclusion or has been terminated by a non-continuable error message, so that CHAROUT may complete the output of data to buffered devices. In the case of CHAROUT ISSUE 1, the lineprinter marker referred to above is examined and if set then the lineprinter buffer is printed.

Entry point 4

A3D uses this entry point when it has to set a special character in the first location of the lineprinter buffer as a result of a call of the procedures "top of form", "lines(m)", "find(m)" or "overprint" in an ALGOL program. The character will be in the least significant seven bits of the accumulator when entry is made.

As CHAROUT uses ALP to have characters printed on the lineprinter, this character is passed to entry point 3 of ALP (see ALP description, 3.2).

Entry point 5

This entry point is used by A3D to print error messages which cannot be displayed using entry point 6, e.g. the array name and the numbers which follow the message "SUBSCR OFLO". The message is passed over to CHAROUT one character at a time. The character is in the least significant seven bits of the accumulator when entry is made and will be displayed on the typewriter by CHAROUT ISSUE 1.

2.1.5.7

Entry point 6

This is used by A3D to display certain error messages. The message is specified by a number which must be in the accumulator when entry is made. The correspondence between number and message is as follows:-

number	Message
1	INT OFLO
2	SUBSCR OFLO
3	MS NOROOM
4	BS NOROOM
5	FOR MS ARRAY
6	FOR BS ARRAY
7	VAC
8	BOUND ERROR
9	RANGE ERROR
10	LOWBOUND ERROR
11	IOSTRING ERROR
12	DIV ERROR
13	ENTIER ERROR
14	SWITCH ERROR
15	SQRT ERROR
16	EXP ERROR
17	SINE ERROR
18	LOG ERROR
19	TAN ERROR
20	ARCSIN ERROR
21	STRING ERROR
22	READ ERROR
23	PRINT ERROR
24	BUFFER ERROR

number	Message
25	LOG ZERO
26	ALLOC
27	MS ARRAY
28	BS ARRAY
29	ERRCALL LP

After a non-continuable error message (see Chapter 5), A3D will enter CHAROUT to display the name of the last array allocated. This is done by using CHAROUT entry point 6 to display "ALLOC" and by using entry point 5 to display the name of the array, one character at a time. When a continuable error message has been displayed, the state of the accumulator upon exit from CHAROUT determines the action taken by A3D.

(1)	Accumulator zero.	This causes A3D to come to a "Dwait". The run of the ALGOL program may be continued by changing the leftmost F2 key (key 19) of the word generator.
(2)	Accumulator negative.	This causes A3D to display the name of the last array allocated. This is followed by a "Dwait". The run of the ALGOL program may be continued by changing the leftmost F2 key (key 19) of the word generator.
(3)	Accumulator positive.	This causes A3D to display the name of the last array allocated, after which the run of the ALGOL program is terminated.

CHAROUT ISSUE 1 exits with the accumulator zero after a continuable error message. All messages are displayed on the typewriter.

2.1.5.7

Tapes

A mnemonic tape in SAC1 is supplied. For use with ALGOL3 a binary tape must be assembled using SAP1. The program is sum-checkable.

3.6 GMT (for ALGOL3)

Code SAC1

Function

The program enables blocks to be written to and read from any specified magnetic tape handler.

Configuration - 503 + Magnetic Tape Handler(s)

Tape

A tape coded in SAC is supplied. For use on ALGOL3 a binary tape must be assembled using SAP1 with keys 35 and 36 depressed to prevent the checksum being formed.

Size - 308 locations, including workspace.

Method of use

Entry point 1

COMP, GMT

This is the initialisation entry point and needs to be entered for each handler used; the handler number must be in the accumulator on entry. Exit is by EXITCP,1.

Entry point 2

COMP, GMT, 2

This is the entry for writing to magnetic tape. The handler number must be in the accumulator on entry and the call must be followed by one parameter word containing the following information:-

00 length of block : 00 start address in main store for transfer.

As the address of the location preceding the parameter is planted in the LINKCP of GMT when the common program entry is made, GMT is able to access the parameters : GMT will eventually exit to the second location after the address in the LINKCP, i.e. by the order EXITCP,2.

On exit the number of the last block written to magnetic tape is in the least significant 19 bits of the accumulator.

Entry point 3

COMP,GMT, 3

This is used for reading and must be given the handler number in the accumulator and two parameter words must immediately follow the common program entry:-

parameter 1) 00 length of block : 00 start address in main store for transfer
parameter 2) 00 0 : block number

Exit is by the order EXITCP,3.

Entry point 4

COMP,GMT, 4

This entry is used by ALGOL3 (program OAST), to set the segment block numbering to start from 8 each time a program is compiled. The contents of the accumulator are undefined on entry and exit.

Entry point 5

COMP,GMT, 5

This entry is used by the program LIBR2. There are certain magnetic tape instructions in LIBR2 so that when it calls GMT to write a block, it is first necessary to set correctly, the GMT identifiers for block numbering ("last block + n" which holds the value of the last block read on handler n-1 and "blockmax + n" which holds the value of the greatest block number written to handler n-1). On entry, the accumulator holds a value which is written into "blockmax + 3" and "last block +3". Exit is by the order EXITCP,1.

Entry point 6

COMP,GMT, 6

This entry is used by ALGOL3 (program LIBR1) to read a block from the current position of the read head on magnetic tape. The handler number must be in the accumulator on entry and one parameter word must immediately follow the common program call:-

00 length of block : 00 start address in main store for transfer.

Format

The first word of each block written contains the block number. This is written in by the routine and is put in the more significant half of the first word.

Blocks are read and written using odd parity, format 2.

Writing and reading may be alternated on any one or more handlers.

Process

The routine divides into three different sections. These

are (i) An initialisation section.

(ii) Writing onto tape.

(iii) Reading from tape.

(i) The initialisation section must be entered before any other entry, to set markers and check the state of the handler.

(ii) The writing section writes given blocks to tape, first checking the state of the handler. It checks that the block is of a suitable length, i.e. ≥ 4 locations and then writes onto tape a block following the last one written. If the handler is 1 and is at the beginning of tape, the routine searches for the last block of the last DUMP2 or DUMP2M batch. This block is five words long with the word "LAST" in the first location. When this block is found, the routine proceeds to write blocks to magnetic tape. Blocks are numbered consecutively beginning at 1.

The parity bit of the control word is checked after writing and during the search for the last block. If set, the position of the block just written is found by, retreating to the block before the one now being written, (unless at the beginning of tape) and advancing to its end. Part of the tape is then erased and writing takes place again. After ten attempts to write an error message is output. An error message is also output when the end of tape marker is reached.

(iii) This section reads a block from tape. It checks that the block number required is within the range of blocks already written onto tape, and also the state of the handler is checked. If no blocks have been written to tape, then no checks are made on the block number range. If the

handler is 1 and is at the beginning of tape, the routine searches for the last block of the last DUMP2 or DUMP2M batch. This block is five words long with the word "LAST" in the first location. When this block is found the routine proceeds to read the required block. The block number is compared with the position of the tape and the appropriate position aimed for by retreating or advancing. A block is then read, and its number checked with the one required. If this is not correct, the appropriate retreats are given or the next block read.

When the correct block number is found, if the parity bit is set on reading, retreats are made and the block read again. After three attempts an error message is output. If the parity bit is not set on reading, a short or long block is checked for. If there is a noise block on error message is output and the next block read. If there is an ordinary short block, ten attempts are made at reading and then an error message output. On detecting a long block an error message is output.

Error messages

Message	Meaning and effect
GMT error MAN H 'n'	The required handler 'n' is in Manual or has no write permit ring. The program continues when this is corrected.
GMT error H EOT	End of tape is reached. No continuation is possible.
GMT error BL TS	The block length given for writing is too short, i.e. <4 location. No continuation is possible.

Message	Meaning and effect
GMT error CANTW	Handler unable to write. No continuation is possible.
GMT error CANTR	Handler unable to read. No continuation is possible.
GMT error NOISE	Noise block encountered. Program continues.
GMT error LTH ER	Long or short block with no parity error found on reading.
GMT error SHORT	Displayed when a short block which is not a noise block is encountered on reading. Program continues.
GMT error BLNOTL	Block number too large, i.e. out of range. No continuation possible.

3.7 INTER

Code SAC1

Function

INTER is used as a common program to supply A3C either directly or via LIBR1 with ALGOL source code. It will obtain source code either from tape reader 1 or from EDIT8 when the editing facility is required (see Chapter 1.3) and will send it to the lineprinter program ALPL when the listing facility is required, (see Chapter 1.3).

It is also used by A3C to display syntactic error messages on the lineprinter.

Tape

A tape coded in SAC1 is supplied; for use with ALGOL3 a binary version should be produced. The program is sum-checkable.

Store used - 44 locations, including workspace.

Method of use

Entry point 1

COMP, INTER 1.

This entry is used for initialising the markers used for the output of error messages and characters to the lineprinter. The common programs ALPL and EDIT8 are also initialised; these are required for certain facilities. (See other entry points).

Entry point 2

COMP, INTER 2.

This entry is used to read a character from reader 1 which is then held in the accumulator on exit. The character read is also passed to the program ALPL (entry point 2) for output on the lineprinter.

This entry gives the compile and print facility.

Entry point 3

COMP, INTER 3.

This entry is used to obtain a character from the program EDIT8 (input from reader 1 or reader 2). The character is in the accumulator on exit. The character is also passed to the program ALPL for output on the lineprinter. This entry gives the edit, compile and print facility.

Entry point 4

COMP, INTER, 4.

This entry is used to obtain a character from the program EDIT8. The character is in the accumulator on exit. This entry gives the edit and compile facility. If, however, a marker is negative indicating that the error output entry has been used (entry point 5), and 31 characters following the error have not yet been displayed, the character will also be passed to ALPL to be displayed on the lineprinter.

Entry point 5

COMP, INTER, 5.

This entry is used by A3C when a syntactic error has been discovered in the ALGOL source program. The message

* error No. n

will be passed, character by character, to INTER at this entry point, where it will be displayed on the lineprinter using ALPL (entry point 2). A marker is set which will cause 31 characters of the ALGOL source code to be sent to the lineprinter if entry point 4 of INTER is being used to supply source code. If A3C's default option for the supply of source code (see Chapter 7.1.1) is being used, then A3C will itself send 31 characters to this entry point for display.

On entry the character to be displayed is in the accumulator

Entry point 6

COMP, INTER, 6.

This is the entry point used by the Executive at the end of the first pass of compilation to ensure that the final contents of the lineprinter buffer are output if necessary.

2.1.5.7

Entry point 7

COMP, INTER, 7.

This is the entry point used by the program LIBR1 during the first pass when a syntactic error has been discovered by A3C in the ALGOL source program. The message

* error no. n

which was passed to LIBR1 by A3C, will be passed, character by character, to INTER (entry point 7) where it will be passed on to ALPL (entry point 2) for display on the lineprinter. If the error was in a library text the message

* in mt text

will also be sent to INTER (entry point 7) to be displayed. The following 31 characters of the ALGOL source code (from papertape or magnetic tape) are then passed to INTER by LIBR1 for output by ALPL.

A standard exit is made from all entry points, i.e. by the order EXITCP.

3.8 LIBR1

Code SAC1

Function

To pass characters of the ALGOL source program to A3C and to provide a facility for accessing a library of ALGOL texts held in character form on a magnetic tape.

Configuration - 503 + 1 magnetic tape handler.

Tape

A tape coded in SAC1 is supplied. For use with ALGOL3 a binary tape must be assembled using SAP1 with keys 35 and 36 depressed to prevent a checksum being formed.

Size - 513 locations, including workspace.

Entry points

Entry point 1

This is the initialisation entry, made from the Executive using its subroutine, "ENTERCP" (see ALGOLB description, 2.1).

The contents of the accumulator are undefined on entry and exit. A standard exit is made, i.e. by the order EXITCP.

Entry point 2

This is the entry point used by A3C to obtain characters of the ALGOL source program. It will be entered when the program is to be compiled using the library facility. Characters will be read from reader 1 directly or from magnetic tape.

On exit from LIBR1 (which is made by the order EXITCP) one character will be in the accumulator.

Entry point 3

This is the entry point used by A3C to obtain characters of the ALGOL source program, when the editing and listing facilities are required. On exit from LIBR1 (which is made by the order EXITCP) one character will be in the accumulator.

Entry point 4

This is the entry point used by A3C to obtain characters of the ALGOL source program when the listing and library facilities are required. On exit from LIBR1 (which is made by the order EXITCP) one character will be in the accumulator.

Entry point 5

This is the entry point used by A3C to obtain characters of the ALGOL source program when the editing, listing and library facilities are required. On exit from LIBR1 (which is made by the order EXITCP) one character will be in the accumulator.

Entry point 6

This is the entry point used by A3C if it discovers a syntactic error in the ALGOL source code. The message

* error no. n

is passed over character by character and sent on to INTER (entry point 7), for output by ALPL on the lineprinter. A marker is also set to ensure that 31 characters of source code are subsequently passed to INTER.

Process

Unless the editing or listing facility is being used, LIBR1 reads characters from reader 1 and passes the characters on to the compiler until the basic word library is read.

Example library A,B,C;

In the example, LIBR1 will retrieve from magnetic tape (handler 7) library texts A, B and C (previously written up by LIBR2) and any auxiliary texts these may use. All these texts will then be passed on to the

compiler, one character at a time, in the order that the texts were written on the tape. When all the texts have been read, LIBR1 returns to obtain its characters from reader 1. However, when the editing facility is used, LIBR1 does not use reader 1 but obtains characters from the common program INTER, which uses EDIT8 to provide it with the edited character stream.

When the listing facility is used, LIBR1 enters the program INTER which will send the character to ALPL to be listed on the lineprinter (see 3.3 of this chapter) as well as passing characters to LIBR1.

Because of the order in which the common programs used by INTER are called (see Chapter 1, Section 3), it is not possible to edit the text held on magnetic tape or to list it * on the lineprinter. Installations requiring these facilities must modify the order in which the source code is edited, listed and read from magnetic tape.

* The code held on magnetic tape is only listed if a syntatic error is discovered by A3C, in which case the 31 characters of the ALGOL source code (from papertape or magnetic tape) following the error are output on the lineprinter. If the error was in a text on magnetic tape the message

* error no. n

is followed by

* in mt text.

Error messages

Message	Condition and action
LIBR1 error STRING	An incorrect character has been met in the library statement (see Chapter 4.11.1). No continuation is possible.
LIBR1 error NOTEXT	No text associated with the name given in the library statement has been found on the library tape. No continuation is possible.

Message	Condition and action
LIBR1 error LIMIT	The limit of 30 texts called directly or indirectly in one library statement has been reached. No continuation is possible.

3.9 Program name LIBR2

Function

To prepare a library of texts held in character form on one magnetic tape for use with the program LIBR1.

Configuration - 503 + magnetic tape handlers.

Tape

A tape coded in SAC1 is supplied.

A binary tape should be assembled using SAP1 with keys 35 and 36 depressed to prevent a checksum being formed.

Size - 481 locations, including workspace.

Entry points

All entries are from the keyboard.

Entry point 1

Type

LIBR2.

for the first entry point. The program reads the following information from reader 1:-

(a) names of library texts

(b) code of each text

(for format of names and texts, on the paper tape, see Chapter 4.11).

Entry point 2

Type

LIBR2;2.

to prepare a new tape for storing library texts. This entry writes a new dictionary block onto the magnetic tape.

Entry point 3

Type

LIBR2;3.

to obtain a list of names of all texts present on the library tape (handler 7).

Process

Two handlers and tapes are required. A scratch tape on handler 4 is used to sort the texts, and the tape on handler 7 permanently holds the dictionary block and texts.

Form of storage on magnetic tape

The beginning of the magnetic tape on handler 7 contains a dictionary of the form:-

00	0	:	00	N
		A1		
				S1
s1		s2		s3
s4		s5		s6
		A2		
				S2
		etc.		

2.1.5.7

All the blocks written to tape are of 50 words length.

The first word of the first dictionary contains N, the number of library texts on the tape.

The second word now contains A1, the name of a library text.

The third word contains the block number S1, of the first block of the text of A1. The next two words then contain three packed block numbers each, s1 to s6, these being the first block numbers of the auxiliary texts.

This pattern from the second word is then repeated until the block has 12 main texts specified. When more texts are added another dictionary block of 50 words will be written, followed by the new texts and so on until the limit is reached which is decided by the end of tape marker being reached when writing a block, or the last block number exceeding 8191.

Following the dictionary there will be blocks containing the code of the texts. Each text will be packed into 50 word blocks (5 chars/word) and when the end of a text is reached a (H) code is written in the final block.

After the last text, a block of less than 50 words is written, so that this position may be easily detected when adding another text.

The block numbering system is only used with the text blocks, that is, block number one is the first block of the first library text on tape. The block number of each block is contained in the F1 and N1 positions of the first word, and all the dictionary blocks have zeros in these positions.

(See Chapter 4.11 for format of texts as presented to LIBR2 for storage on magnetic tape.)

Effective overwriting of a routine

A text is automatically cancelled from the dictionary when another text is written having the same name. This will be done by changing the block number in the dictionary to point to the latest text instead of the earlier one and then replacing the block numbers of the auxiliary text in the dictionary. The original text will then remain on the tape but there will be no means of reaching it.

Restrictions

A maximum of 8191 blocks may be written on the library tape.

Operating instructions

Step	Typed Instruction	Message output by Typewriter	Tape in reader 1	Remarks
1				Scratch tape on handler 4. Library tape on handler 7.
2	LIBR2;2.	END		New library tape prepared.
3	LIBR2.		names and texts	
		MAX BLOCKNO=n END		n = the number of blocks now held on the library tape.
4	LIBR2;3.	list of names of the texts on the library tape END		

Error messages

Message	Condition and Action
H4MAN	Handler 4 is in Manual or has no write permit ring. The program continues when the error is corrected.
H7MAN	Handler 7 is in Manual or has no write permit ring. The program continues when the error is corrected.
too many names	More than six auxiliary texts have been named in one list. The program continues but the texts associated with this particular list may be incorrect and should be replaced.
XNAME	An incorrect character has been found in the list of names or code of a text. No continuation is possible.
RP	A parity error has been found on magnetic tape when the handler was trying to read. The program tries to read the same block again.
WP	A parity error has been found on magnetic tape when the handler was trying to write. The program tries to write the same block again.

3.10 OAST

Code SAC1

Functions

A. Outputs blocks of owncode to magnetic tape.

B. Inputs blocks of owncode from magnetic tape.

C. Outputs segments to magnetic tape.

D. Inputs segments from magnetic tape.

Configuration - 503 + two magnetic tape handlers.

Tape

A tape coded in SAC is supplied; for use in ALGOL3 a binary tape must be assembled using SAP1 with keys 35 and 36 depressed to prevent the checksum being formed.

Special Note Two versions of OAST are available:-

(a) OAST (FOR 2 HANDLERS)
(b) OAST (FOR 3 HANDLERS).

With version (a) only two handlers need to be used for the ALGOL3 system. The tape on handler 1 will contain DUMP2 or DUMP2M batches at the beginning of the tape, followed by the systems blocks containing A3C, A3L and A3D with their auxiliary programs (see Chapter 1.2.3) after which the segments of a segmented program will be placed. The tape on handler 4 will hold the owncode passed to OAST by A3C.

With version (b) three tape handlers are required. Handler 1 for the main store batch contains the Executive, ALGOLM, handler 4 for the owncode and handler 5 for the systems blocks and segments.

Method of use by ALGOL3

A. Entry point 1

The Executive uses this entry point to initialise OAST for owncode output. It is used immediately before the first pass of compilation of an ALGOL program begins.

The contents of the accumulator on entry and exit are undefined.

Entry point 2

A3C uses this entry point to pass owncode to OAST. The owncode is written to magnetic tape in blocks of 65 words. The accumulator on entry contains 39 bits of owncode and on exit is undefined.

Entry point 3

Since OAST does not know when it has received all the owncode, this entry point is used by the Executive, ALGOLM, when the first pass of compilation has been completed. OAST writes its buffer containing the final owncode to magnetic tape as a 65-word block; on entry the contents of the accumulator are undefined, but on exit the accumulator holds

00 m : n

where m is the number of words containing owncode in this final block and n is its block number.

B. Entry point 4

ALGOLM uses this entry point to initialise OAST for owncode input. It is used immediately before the second pass of compilation begins. On entry the accumulator holds:-

00 m : n (see entry point 3)

on exit the contents of the accumulator are undefined.

Entry point 5

A3L uses this entry point to obtain owncode which is read back from magnetic tape. On entry the contents of the accumulator are undefined; on exit the accumulator contains 39 bits of owncode.

C. Entry point 6

The Executive (ALGOLM) uses this entry point to initialise OAST for segment input and for segment output. The contents of the accumulator are undefined on entry and exit.

Entry point 7

This is the entry used by the Executive (ALGOLM) and the loading program (A3L) to write blocks to magnetic tape. The entry is:-

COMP, OAST, 7

The accumulator on entry contains the parameter 00 a : 00 b where a is the length of the block and b the main store address of the start of the block. On exit the least significant 19 bits of the accumulator contain the number of the block just written away on magnetic tape.

D. Entry point 8

This entry is used by the Executive (ALGOLM) and the dynamic routines (A3D) to read blocks from magnetic tape. The entry is:-

COMP, OAST, 8
parameter 1) 00 p : q
parameter 2) 00 0 : 00 r

p is the length of the block to be read
q is the block number on magnetic tape
r is the main store address of the start of the block to be read.

On exit, the accumulator contains the main store start address of the block.

Entry point 9

This entry is used by the Executive program (ALGOLM) to set the block numbering for segments on magnetic tape to

start from 8. It is used immediately before the start of the second pass of compilation. Accumulator contents are undefined on entry and exit.

Process

Output of owncode

OAST receives 39 bits of owncode at a time from the compiler (A3C) and fills a 65-word buffer with the owncode. When full the buffer is written to magnetic tape (handler 4) using the program GMT (for ALGOL3).

Input of owncode

The blocks of owncode on magnetic tape (handler 4) are read down into a buffer using the program GMT (for ALGOL3). The owncode in the buffer is passed to A3L, commencing with the last 39 bits received from A3C, since the owncode must be passed to A3L in the opposite order to which it was received. When the buffer is empty another block is read from magnetic tape.

Output of segments

Segments or blocks of code are written to magnetic tape (handler 1 or 5) using the program GMT (for ALGOL3).

Input of segments

Blocks written to magnetic tape by GMT (for ALGOL3) are read down as specified by the parameters.

Store used

OAST uses 179 locations, including workspace.

Error messages

Message	Cause and Action
OAST error NOMOC	A3L has entered OAST to obtain owncode when all the owncode sent up by A3C has been retrieved. Non-continuable error.

3.11 OCBS

Code SAC1

Functions

A. Outputs owncode to backing store

B. Inputs owncode from backing store

C. Outputs segments to backing store

D. Inputs segments from backing store.

Configuration - 503 + core-backing store.

Tape

A tape coded in SAC1 is supplied. For use with ALGOL3 a binary tape must be assembled using SAP with key 36 depressed. The program is sum-checkable.

Entry points used by ALGOL3

Entry point 1 - not used.

Entry point 2

The compiler (A3C) enters OCBS with 39 bits of owncode in the accumulator which are written to the first-free location of backing store. The accumulator is undefined on exit.

2.1.5.7

The exit order is

EXITCP,1.

Entry point 3

A3L enters OCBS to obtain owncode. 39 bits of owncode are retrieved from backing store and are in the accumulator on exit. The exit order is

EXITCP,1.

Entry point 4

A3L enters OCBS to have a segment written to core-backing store with the following parameter in the accumulator:-

00 segment length : 00 start address of segment in main store.

On exit the start address of the segment on core-backing store is in the accumulator. The exit order is

EXITCP,1.

Entry point 5

A3D enters OCBS here to have a segment retrieved from core-backing store and placed in main store. A3D plants the address of the location containing its entry instruction to OCBS in the LINK CP of OCBS. Immediately following this entry instruction are two parameters specifying the segment required. The format is:-

parameter 1) 00 segment length : start address of segment in CBS
parameter 2) 00 : 0 : start address of segment in main store.

The accumulator is undefined on exit, which is made by the order

EXITCP,3.

Process

Location 0 and 1 of backing store are used to hold the address of first-free and last-free location respectively on backing store. In writing owncode and segments to core-backing store the contents of locations 0 and 1 are checked each time. When first-free is equal to last-free an error message is displayed.

Owncode is written to backing store starting from the first-free location and working towards a higher location. On retrieval the first 39 bits of owncode passed over to A3L are those in the (current first-free -1) location i.e. the last location filled. As each 39 bits of owncode are read the backing store first-free location is reduced by 1. Segments are written to backing store starting from location last-free - n, where n is the length of the segment - 1. Writing and retrieval of segments is by a block A.D.T. transfer. Owncode is written and read using single word transfers.

Error message

The message

OCBS error BS full

is displayed when backing store first-free is equal to last-free.

Store used - 74 locations, including workspace.

3.12 ERINT

Code SAC1

Function

This program is used in conjunction with the procedures "noflo" and "oflo" available in ALGOL3 ISSUE 2 (see Chapter 5). It enables the run of an ALGOL program to be continued when a floating-point overflow

is generated. If the procedure "noflo" is currently active in the ALGOL program, no message will be displayed. If the procedure "oflo" is called, the message

ERINT 1
Dwait

will be displayed, after which, the program may be continued. In both cases the value $(1-2^{-29})\times 2^{255}$ is taken as the result of the operation which generated the error interrupt.

Store used - 73 locations, including workspace.

Method of use

ERINT is written as a SAC common program. When used with ALGOL3 ISSUE 2, this program need only be in store when the ALGOL program is running, so that it could be input immediately after the display of "Dwait" at the end of compilation of the ALGOL program. However, it may be input with the common programs used by A3D or before the ALGOL3 Executive (see Chapter 6, Operating Instructions).

There are two entry points:-

Entry point 1

This is an initialisation entry used by A3D. It modifies RAP or RAPMT and the program head of ERINT (see PROCESS USED). Exit is made by the order

EXITCP.

Entry point 2

This is the point at which RAP or RAPMT enters ERINT when an error interrupt is generated. Unless the interrupt has been caused by a floating-point overflow ERINT will return control to RAP. If there has

been a floating-point overflow, ERINT will cause the program to continue with or without an error message (see PROCESS USED).

Process used

Entry point 1

A3D will search main store for the program ERINT as soon as the run of an ALGOL program is started, and if it finds it, A3D will enter ERINT at entry point 1. Here ERINT modifies RAP so that RAP will enter ERINT as soon as an error interrupt is detected. Since this entry has to be made to the second entry point of ERINT, the main entry instruction in the program head of ERINT is also modified so that it contains a jump to the second entry point.

Entry point 2

When an error interrupt is detected, RAP will enter ERINT at the main entry point (in the program head) which will have been modified to cause a jump to the second entry point of ERINT. Here the error interrupt locations are examined (see Sections 1.2.3, 1.2.4 and 1.2.5 of the Manual) to ensure that the error interrupt was caused by a floating-point overflow. ERINT also searches through the list of programs in main store to ensure A3D is in store.

If either of these conditions is not satisfied, ERINT returns control to RAP which will display the error message (for the format of the error message see Note below). However, if both are satisfied, ERINT will examine a marker in A3D which is set when the procedure "noflo" is active. If this marker is not set then the message

ERINT1
Dwait

is displayed.

The ALGOL program may then be continued if the leftmost F2 key (key 19) of the word generator is changed.

If the marker is set, i.e. "noflo" has been called, then no error message will be displayed and the ALGOL program will be continued.

Exit is made from ERINT using the 66 instruction (see Section 1.2.4). Upon continuation the overflow register and Auxiliary Register have the values they had when the floating-point overflow was generated and $(1-2^{-39}) \times 2^{255}$ (i.e. the largest positive number the 503 can hold), is taken as the result of the operation that caused the error.

Tapes

A mnemonic tape coded in SAC is supplied. For use with ALGOL3, a binary tape must be produced using SAP1. The program is sum-checkable.

Note Once entry point 1 of ERINT has been used, RAP will remain in the modified form, unless it is re-input from paper tape. However, unless a program called ERINT is in store, RAP or RAPMT may be used in the normal manner, except that should an error interrupt be generated, the format of the message displayed will be:-

ERINTm

(where m is the number of the error interrupt) instead of

ERRINT m

after which RAP will come to a typewriter demand.

The symbol ">" (see 2.2.1.4 of the 503 Manual) will not be displayed by this modified version of RAP or RAPMT when entry to a program has been made.

3.13 CBIT

Code RAP-binary.

Function

To make it possible to read in several binary programs on the same paper tape, by typing IN. once.

Tape

A RAP-binary tape is provided.

Method of use

CBIT is in two parts, CBITA and CBITB. To prepare a set of binary programs for continuous input, copy the following tapes onto a single reel of paper tape:-

(1) CBITA
(2) All binary programs to be input
(3) CBITB

Any characters on a tape not normally read (e.g. the final character '80' on a RAP-binary tape) should be removed when the tapes are connected together. A small section (2 or 3 inches), should be left between the separate programs.

Operating instructions

Tape in Reader 1	Typed Instruction	Displayed Message
Combined binary tape	IN.	*CBITA NAME of PROG1 NAME of PROG2 etc. *CBITB

Errors

(1) If for some reason, a CBIT combined binary tape is not completely read in (e.g. if the operator decides it is the wrong tape, or the tape tears), then an error state is caused in RAP. Instead of outputting '?' after a Manual interrupt RAP will output the symbol '↑'.

(2) When RAP (or RAPMT) is in the above error state, and the operator reads in a normal binary program tape, the program name will be displayed on the typewriter as usual, but control passes to the RAP input routine, instead of the RAP typewriter demand. This will not cause an error, but will be inconvenient for the operator, because he will need to press message interrupt to regain control.

(3) To correct the RAP error state either:-

(a) Input another CBIT combined binary tape completely. (Note that CBITA or CBITB can be input any number of times).

(b) Input a copy of CBITB attached to the end of a combined binary tape, or a separate copy of CBITB.

(b) Re-input RAP.

(4) If any one of the binary tapes fails to read in correctly, because of a misread, '*' will be displayed as usual and there will be a RAP demand. The complete tape should be read in again after the programs just read in have been cancelled or 'RESET' has been typed.

(5) If any of the following errors occur, NOROOM, ERRSUM or NOPROG, then in place of the usual message, '*' will be displayed. This will be followed by a RAP demand.

Process used

CBITA and CBITB are made up of RAP action words which change RAP. CBITA changes RAP to allow continuous input of binary programs, and CBITB restores RAP to its original form. Neither CBITA nor CBITB occupies space in store and do not appear on the RAP list.

Versions of RAP

CBIT can be used on RAP (Issue 2) and RAPMT.

Appendix 1: ALGOL3 FOR INSTALLATIONS WITHOUT A LINEPRINTER

1. INTRODUCTION

ALGOL3 is designed to use the lineprinter in three ways:-

(i) To list the compiling ALGOL program if required.

(ii) To list syntactic errors in the ALGOL source program.

(iii) To output the results of the running ALGOL program when the appropriate device setting procedure is called.

For this reason, the Executive and the auxiliary programs INTER and CHAROUT contain instructions referring to the lineprinter programs ALP and ALPL.

Modified versions of ALGOLB, ALGOLM, INTER and CHAROUT are available for installations which do not have a lineprinter.

2. ALGOLB and ALGOLM (No lineprinter)

The difference between these versions and those described in Chapter 7.2 is that a demand for the listing facility at compile time will be ignored. The other facilities asked for are executed.

e.g. ALGOLB,CELP. is equivalent to
ALGOLB.CEL.

3. CHAROUT (No lineprinter)

The differences between this version and that described in Chapter 7.3.5 are:-

(i) If the output device number received at entry point 2 is greater than 3, tape punch 1 will be used. This means that a call of "punch(4)" or "lineprinter" in the ALGOL source program is equivalent to a call of "punch(1)".

(ii) The initialisation of ALP and the entry to ALP to output the final contents of the lineprinter buffer have been removed.

(iii) An immediate exit is made from entry point 4 of CHAROUT. This means that calls of the procedures "top of form", "lines(m)", "overprint" and "find(m)" have no effect.

4. INTER (No lineprinter)

The differences between this version and that described in Chapter 7.3.8 are:-

(i) ALPL is not initialised and not entered at the end of the first pass of compilation.

(ii) The entry to ALPL to list the compiling ALGOL source code has been removed (see Entry point 2 and Entry point 3 of INTER).

(iii) Syntactic error messages are displayed on the typewriter not the lineprinter. The message will be followed by 31 characters of the ALGOL source code.