Installing ALGOL68C on an IBM System/360 or /370

Contents

(These installation instructions are only valid for compiler version 247, translator version 119, ALGOL 68 library version 107 and 370 assembler library version 127.)

1. General

ALGOL68C has been developed for the 360/370 range of computers with the universal instruction set running OS/MVT Release 21. It has also been successfully installed on systems running OS/MFT, OS/VS1, and OS/VS2 (SVS). It is also believed to run under OS/MVS.

No 370-only instructions are used except for STCK (store clock) which is only executed if the CVT indicates that the time of day clock is supported. The boundary alignment feature is not needed.

The currently distributed versions of ALGOL68C are known as "Prerelease" as the system does not implement the full ALGOL68C language. The currently unimplemented features are described in a separate document. The current state of ALGOL68C precludes its use by other than the dedicated user. The full implementation is expected to

be available in the late summer of 1976.

The distribution package consists of a magnetic tape, the printer output of the job that wrote the tape, and printouts of various documents (including this document). Note that, as the tapes are prepared in batches, and the source copies of the documents are updated from time to time, the printed copies may differ from the corresponding copies of those documents on the tape. In case of discrepancy, the printed version is likely to be the most correct.

2. Portability

The ALGOL68C system is designed so that implementing it on a new machine range is straightforward. The compiler produces code in an intermediate language, ZCODE. ZCODE is a simple assembly language for a machine with conventional arithmetic and index registers and a conventional storage addressing scheme.

The compiler reads a simple description of the target machine (e.g. the number and type of registers, the size of INTs, etc.) so that the ZCODE produced is tailored for the target machine.

An implementor of ALGOL68C on a new machine range must write a translator (or an interpreter) for ZCODE, and also must write a run time system. Three to nine man-months are usually needed.

(It should take only a few hours to install the ALGOL68C system on a /360 or a /370 from the distribution tape.)

3. Maintenance

ALGOL68C is maintained as indicated in the Conditions of Distribution by the University of Cambridge Computing Service.

We would be grateful to be notified of any bugs, infelicities, or other helpful suggestions. Please send all material (e.g. program sources, compiler output - messages and ZCODE, dumps, JCL, and the version numbers of the system being used) to

ALGOL68C Maintenance,
Computing Service,
Computer Laboratory,
University of Cambridge,
Corn Exchange Street,
CAMBRIDGE, CB2 3QG,
England.

Existing installations will be notified of major new releases of the system. A standard charge is made for each copy of the system sent out (this charge is currently not greater than 15 pounds Sterling and depends on the postal rate). Please do not send money with the order for the system: an invoice will be sent when the system is dispached.

Requests for the system should be sent to

ALGOL68C Distribution,

Computing Service,
Computer Laboratory,
University of Cambridge,
Corn Exchange Street,
CAMBRIDGE, CB2 3QG,
England.

4. Documentation

The ALGOL68C language is described by "The ALGOL68C Reference Manual" by S.R. Bourne, A.D. Birrell, and I. Walker. The manual is copyright; it may be obtained from

The Computing Service Bookshop,
Computer Laboratory,
University of Cambridge,
Corn Exchange Street,
CAMBRIDGE, CB2 3QG,
England.

and costs 2 pounds (Sterling). Bulk orders can be considered. Section 8.2 (on transput) is not yet complete but it will be forwarded to purchasers of the manual when it is available.

An introduction to ALGOL68C is being prepared.

A Users' Guide to running ALGOL68C programs on the Cambridge 370/165 is in preparation. It is suitable for modification by other installations for describing what JCL to use, etc.

This document is known as the 360/370 installation guide, and may be copied from the distribution tape.

An Implementors' Guide describing how to implement ALGOL68C on a new machine range is in preparation. Two documents describing ZCODE are available from the distribution address given in 3 above - one of these describes ZCODE as it is currently ("old ZCODE"), the other gives a very brief description of the proposed new ZCODE.

The following documents are included on the distribution tape and are also supplied as printouts with the tape:

Character set,
Differences from ALGOL 68,
Features currently unimplemented,
Bugs known and bugs mended,
Changes from previous releases.

It is regretted that no estimates can be given for the availability of any of the above documentation that is in preparation.

5. Tape Format

The following tape recording parameters are possible:

9 track, 800 bpi NRZI or 1600 bpi PE, unlabelled or IBM standard labels;

7 track, 556 bpi NRZI or 800 bpi NRZI, odd parity, "data convert", unlabelled or IBM standard labels. (Data convert causes three (8 bit) bytes to be written as four 6 bit characters on the tape. If the number of bytes written is not a multiple of three, the record is padded with zeros to fill the last 6 bit character; note that it is not padded to a multiple of three bytes.)

Files are recorded on the tape in fixed length blocked records with a logical record length of 80. The blocksize (physical record length) may be specified by the recipient and it must be a multiple of 80 and not greater than 16320. Carriage controls are not used.

The standard recording parameters for System/360 are:

9 track, 800 bpi NRZI,
3200 bytes maximum block size,
no labels.

The standard recording parameters for System/370 are:

9 track, 1600 bpi PE,
3200 bytes maximum block size,
no labels.

Please note that it is particularly inconvenient to write the ALGOL68C tapes with standard labels so they should only be requested if absolutely necessary.

Files on the tape are in one of three forms:

a) simple - the file consists of only one subfile. One record of the subfile occupies precisely one record of the file. The file may be copied by using a standard IBM utility (e.g. IEBGENER).

b) UPDTE format - there are a number of subfiles in the file. One record of a subfile occupies precisely one record of the file. Documents are preceded by a record starting ./ this being a control record for the IBM IEBUPDTE utility program. IEBUPDTE may be used directly to make a partitioned dataset from this file. Sample JCL for doing this is included on the tape.

c) packed format - there are normally a number of subfiles to such a file. The records of the file and the records of the subfiles do not correspond. A subfile starts with hexadecimal 01. A string enclosed in double quotes follows, this string is the subfile identification, and, for OS/360, is a ddname followed by a parenthethised member name. The string is followed by hexadecimal 02 and this is followed by the text of the subfile. The subfile is terminated by hexadecimal 03. Records of the subfile are separated by the character represented by hexadecimal 1E.

The object modules for the program to unpack (and pack) these files are included on the tape. The subfile idf strings for material on tapes distributed by Cambridge are of the form "UNPACKED(membername)", and the membername will vary.

The record lengths in the subfiles do not exceed 136.

The precise layout of the tape is described by comments at the start of the job that wrote the tape. That job is file 3 on the tape and a printout accompanies the distribution tape.

6. Character codes

Characters on the tape are in a extension of the EBCDIC code differing from that described in the IBM manual "System/370 Principles of Operation" (GA22-7000-3) in that

"ƒ" (opening square bracket) is represented by hexadecimal 42,
"„" (closing square bracket) is represented by hexadecimal 43,
"Ž" (logical or) is represented by hexadecimal 63,
"Û" (logical and) is represented by hexadecimal 64,
"Ü" (uparrow) is represented by hexadecimal 65,
"Ý" (backarrow) is represented by hexadecimal 66.

(The compiler also accepts the following EBCDIC codes:

"ƒ" (opening square bracket) hexadecimal AD,
"„" (closing square bracket) hexadecimal BD.)

The characters "Ž", "Û", "Ü", and "Ý" are available for use as operators. None of these characters need to be represented at an installation.

Similarly, square brackets need not be represented at an installation as round brackets are equally acceptable in every syntactic position except in row declarations. If round brackets are to be used instead of square brackets, a row declaration should be preceded by the row-symbol, e.g. instead of "ƒ„AMODE a" use "ROW()AMODE a".

However, if square brackets are not available, care must be taken when transcribing material from the tape as square brackets are used as the standard sub- and bus-symbols.

The first file on the tape gives details of the character set, and a printout of this file is supplied with the distribution tape.

7. Installing the ALGOL68C system from tape

7.1 Installation

File 5 on the tape is a job to install the ALGOL68C system from the tape. The following job should be run to copy files 5 and 6 to disc:

```
//ALGOL68 JOB (,),'JOB CARD TO SUIT INSTALLATION'
//*
//* COPY FB80 DATA SET
//*
//F PROC N=,                  DATA SET NAME
//       L=,                  TAPE FILE NUMBER
//       SER=<TAPE>,UNIT=TAPE<TRKS>, TAPE SERIAL AND UNIT NAME
```

```
    //       DEN=<DEN>,LAB=<LAB>, TAPE DENSITY AND LABEL TYPE
    //       BLKSIZE=<BLK>,       TAPE BLOCKSIZE
    //       TRAN='<TRAN>'        7TRACK RECORDING TECHNIQUE
    // EXEC PGM=IEBGENER,REGION=80K
    //SYSPRINT DD SYSOUT=A
    //SYSIN DD DUMMY
    //SYSUT1 DD UNIT=&UNIT,VOL=(,RETAIN,SER=&SER),
    //          LABEL=(&L,&LAB),DISP=OLD,DSN=A68C&L,
    //          DCB=(LRECL=80,BLKSIZE=&BLKSIZE,RECFM=FB,DEN=&DEN&TRAN)
    //SYSUT2 DD UNIT=SYSDA,
    //          DISP=(,KEEP),DSN=&N,
    //          SPACE=(TRK,(1,1)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=2480)
    // PEND
    //*
    // EXEC F,L=5,N='AL68.D.JOB'
    // EXEC F,L=6,N='AL68.D.JOB1'
```

The disposition of SYSUT2 may need to be changed from KEEP to CATLG if this is a requirement in the receiving installation. Almost certainly, the data set names 'AL68.D. ... ' will need changing to suit the installation's requirements. The DCB characteristics may need changing for discs other than 3330s.

The job from file 5 (copied to AL68.D.JOB) contains comments describing in detail how to install the system; this description is reproduced here:

```
//*
//* 1. Make the job card suitable for your installation.
//*
//* 2.1 This job constructs the ALGOL68C system on the following data
//*     sets:
//*                A68LIB.OBJ   object module data set
//*                A68LIB.MOD   load module data set
//*                A68LIB.SYS   initialisation and library environments
//*                A68LIB.PROCS catalogued procedures
//*                A68LIB.TESTS tests
//*
//*     The blocksizes for these data sets are optimised for 3330 discs
//*     and should be changed if other discs are used.  Similarly,
//*     the space allocations will need changing for other discs.
//*     These changes should be made in the first step, "X", of the
//*     job.
//*
//* 2.2 If you wish to change the names of these data sets, change only
//*     the dsnames in the first step, "X".  The other steps obtain
//*     the dsnames by using JCL refer-back, e.g. DSNAME=*.X.OBJ .
//*     The data sets are allocated as (NEW,PASS) in the first step
//*     and are kept in the last step, "KEEP", only if the return codes
//*     from the previously executed steps are 4 or less.  Thus, if
//*     the job fails for some reason, correct the reason for the
//*     failure, and run the complete job again.
//*
//* 2.3 If you wish to have the data sets catalogued, change the DISP
//*     in the last step from (OLD,KEEP) to (OLD,CATLG).
//*
//* 2.4 If you already have an ALGOL68C system with data sets of
//*     the same names as those used in the job, you should either
//*     make a new system with different data set names (2.2 above)
//*     or change the data set names in the first step to be
```

```
//*     the temporary data set names
//*
//*                 &&OBJ
//*                 &&MOD
//*                 &&SYS
//*                 &&PROCS
//*                 &&TESTS
//*
//*     and remove the CONDX parameter from the penultimate step,
//*     "COPY".  This will cause the system to be constructed on the
//*     temporary data sets, then to be copied (with the replace
//*     option) to the existing data sets. Because the replace option
//*     is used, the default names, e.g. A68C, Z370, will be replaced
//*     by aliases describing the new system. Execution of the COPY
//*     step will cause the last step, "KEEP", to be omitted, and so
//*     the temporary data sets will be correctly deleted at the end of
//*     the job.
//*
//*     N.B. Ensure that the existing data sets have sufficient space
//*     and directory blocks to receive the new members.  See the first
//*     step, "X", for estimates of requirements.  Note also that all
//*     the data sets must exist before the job is run.
//*
//* 3.  The data set "S.FORTLIB" (see the JCL procedures) is not
//*     distributed as it contains the proprietry FORTRAN routines used
//*     for sin, cos, tan, arcsin, arccos, arctan, exp, sqrt.  It is up
//*     to the installation to arrange linkage to a suitable set of
//*     routines.
//*
//*     ALGOL68C versions of these routines are being developed and
//*     will be distributed with future releases of the system.
//*
//* 4.  Stropping:  The standard job constructs an ALGOL68C system
//*     that supports "case stropping" (that is, tags are written
//*     with lower case letters and digits, and indicants are written
//*     with upper case letters).  The alternative, "prefix-quote
//*     stropping", has tags written with upper or lower case letters
//*     or digits and indicants must be immediately preceded by a
//*     single quote (apostrophe) and are written in upper case
//*     letters.  TO construct a system for prefix-quote stropping,
//*     change the default value for the "Q" symbolic parameter in
//*     the "P" procedure (unpack to pds) from null to Q (i.e. instead
//*     of "Q=,", use "Q=Q,") and in the step LKA68C to link-edit the
//*     compiler, change "INCLUDE(EBCDIC)" to "INCLUDE(QEBCDIC)".
```

7.2 Other files

The job from file 6 (copied to AL68.D.JOB1) may be edited to extract other material from the tape as required.

8. Catalogued Procedures

The job to construct the ALGOL68C system will have built the data set A68LIB.PROCS. This contains JCL procedures suitable for SYS1.PROCLIB (or whatever happens to be the installation's catalogued procedure data set). Of course, the procedures may also be used 'in stream' if a // PEND card is added to the end of each procedure.

The JCL procedures supplied with the library are those used on the 370/165 at Cambridge, and may need to be changed for other installations. Each procedure contains one or more steps, of which there are four types, i.e. compile, translate, link edit and go. There is never any need for users to separate the compile and translate steps.

Because of the difficulties involved in overriding JCL cards, a large number of symbolic parameters are used in the procedures so that overriding JCL cards should not normally be necessary.

8.1 Compile step:

The symbolic parameter &SYSLIB, defaulting to A68LIB.MOD, specifies the load module library to be used for loading modules for execution.

The dataset refered to by DDname INIT is defined as a PDS member with name given by &IDSN(INIT&SYSI), which defaults to A68LIB.SYS(INIT). This dataset contains definitions of the target machine, the default environment, and the strops of reserved words in ALGOL68C source programs. The name of this dataset will normally not be changed by users, but its default value can be changed if a new version of the dataset is to be used.

The dataset referred to by DDname SYSENV contains details of the library environments. It is a PDS with default name A68LIB.SYS. Again, this will not usually need to be changed by users.

Additional environments can be supplied by the user including DD cards in the JCL, e.g.

```
//A68.MYENV DD DSN=MYFILE.ENVIRON,DISP=SHR
```

would be picked up by an ALGOL68C using-directive such as

```
USING FRED FROM "MYENV"
```

An environment dataset defined in this way can be a PDS containing one environment in each member. The using-directive would then take the form

```
USING FRED FROM "MYENV(membername)"
```

The dataset referred to by DDNAME XREF receives cross reference information if the CROSSREF keyword has been included in the PARM field of the EXEC statement. This information can be processed by a seperate program (see 8.5 below) to produce an identifier cross reference listing.

The dataset referred to by DDname CODE receives the ZCODE output from the compiler. The block size of 2498 is a suitable value for 3330s but should be changed for other devices.

The dataset referred to by DDname ENVOUT receives the environment data from 'handles' defined in the source program. It must be defined in the JCL as a physical sequential dataset, and, for programs without any handle in their source text, will most usefully be DSN=NULLFILE (i.e. DD DUMMY) or otherwise a member of a PDS. Note that its disposition is OLD.

The source program is read from the dataset defined by the SYSIN DD

card and compiling information and diagnostics is written to the dataset defined by the SYSPRINT DD card.

8.2 Translate step:

The STEPLIB DD card is as for the compile step.

The dataset referred to by the SYSIN DD card contains the ZCODE output from the compile step. In the procedures, this is always the temporary file &ZCODE.

The object module is written to the dataset defined by the SYSGO DD card which must have DCB parameters acceptable to the linkage editor.

Translation information and diagnostics are written to the dataset defined by the SYSPRINT DD card.

8.3 Link edit step:

The dataset referred to by DDname SYSLIB defines a set of libraries containing modules which may be called by the ALGOL68C program, including the ALGOL68C runtime system. The first library is defined by the symbolic parameter &SYSLIB which also defines the STEPLIB of the compile and translate steps, and is described above. The second library is S.FORTLIB, which contains modules for computing sine, cosine, exponential, etc. It is suggested that IBM FORTRAN double precision routines are used with an interface module, a suitable example of which is to be found on the ALGOL68C distribution tape. Eventually suitable routines will be provided as part of the ALGOL68C system.

If the module being linked had a named parent module, then that module will be included from the user's own library, which must be concatenated to the system libraries.

SYSLMOD, SYSPRINT and SYSUT1 are the same as for any run of the linkage editor, so should require no explanation.

The output from the translation step is contained in the dataset referred to by the SYSLIN DD card. If a SYSIN DD card is supplied by the user then the dataset it refers to will be concatenated to that referred to by the SYSLIN DD card. This can be used to provide linkage editor control cards and thus pick up routines from specified libraries rather than the defaults.

8.4 Go step:

This is very simple. The only DD card supplied in the procedure is SYSPRINT, to which run-time error messages are written; other datasets may be supplied from the calling job stream.

8.5 Cross reference program:

The information written by the compiler to the dataset referred to by the DDNAME XREF can be processed to produce an identifier cross reference listing. The JCL procedure A68XREF is provided to run this program. Its parameters are similar to those of the compiler procedures.

The cross reference information from the compiler must be supplied on DDNAME FROM. The formatted listing is written to the dataset referred to by the DDNAME TO, and any diagnostics are sent to the dataset referred to by DDNAME SYSPRINT. For a normal run, no parameters or overriding DD cards need be specified.

9. Testing the system

The data set A68LIB.TESTS contains a number of tests. The members JOB1, JOB2, JOB3, JOB4 are jobs to run the tests. The job cards of these jobs will need modifying to suit the installation. On a 370/165, a cpu time of 75 seconds is sufficient to run each of the jobs. The results of the jobs may be compared with members RESULTS1, RESULTS2, RESULTS3, RESULTS4.

10. Writing ALGOL68C programs

This will be covered more fully by the Users' Guide (in preparation).

Programs may be written using either 'case stropping' or 'prefix-quote stropping' depending on how the system has been constructed (7 above).

With case stropping, indicants and reserved words are written in upper case, and tags are written in lower case, e.g. BEGIN MODE FRED = STRUCT(INT a); REAL b.

With prefix quote stropping, indicants and reserved words are written in upper case and are preceded by a single quote, tags are written in either case. For example 'BEGIN 'MODE 'FRED = 'STRUCT('INT A); 'REAL B.

Note that if prefix quote stropping is used, the case of a letter is significant (but this is likely to change to provide compatibility with dot stropping).

In both cases, only letters are allowed in indicants.

It is intended that prefix-quote stropping should be superceded by (prefix) point stropping (which is identical to prefix-quote stropping except that a point is used instead of a single quote). For example, .BEGIN .MODE .FRED = .STRUCT(.INT A); .REAL B . This stropping is the standard accepted by IFIP WG 2.1. With point stropping, the case of a letter is not significant.

The compiler reads a program heading from the parameter channel. The program heading on the parameter channel must be preceded by a single "/" which is used to separate run time options from parameter channel input. For example

```
// EXEC A68CLG,PARMC='/XREF USING USER FROM "SYSENV(REAL)"'
```

(No run-time options are implemented yet.)

The program heading may also precede the program which is read from

SYSIN.

The program heading may consist of any combination of

USING directive: this specifies to the compiler how the program segmentation is to be used (see 11 below).

example: USING FRED FROM "MYENV(FRED1)"

TRACE directive: this is used for debugging the compiler. An integer must follow specifying the level of compiler trace.

example: TRACE 63

KEY directive: This is not useful at present. It must be followed by an integer.

example: KEY 4

XREF directive: This specifies that the compiler is to produce output on the data set specified by the XREF DD card for later processing by the crossreference program.

examples: XREF
CROSSREF

NAME directive: This allows a name to be given to the control section (CSECT) produced by the translator. It also causes an external reference to be constructed when any daughter module is translated. The name is specified as a string of up to eight characters that must form a legal OS CSECT name.

example: NAME "MODULE1"

TITLE directive: This causes the title to be included as part of the compiler output to SYSPRINT to enable multiple compilations in the same job to be distinguished.

example: TITLE this is a title it is syntactically a tag

STRICT directive: The intention of this is to give a warning message if any extensions from ALGOL 68 are used.

example: STRICT

The words USING, FROM, TRACE, KEY, XREF, CROSSREF, NAME, TITLE and STRICT must be stropped.

The program consists of a serial clause.

11. Separate compilation

The language mechanism for supporting separate compilation is described in the ALGOL68C Reference Manual section 5.5.5 ('handles'). It allows a program to be segmented in a tree-structured manner. a

segmented program must be compiled from the root first (in order that the environments are available in a tree-structured manner). The environment produced by compiling a module which includes a 'handle' is written to the data set described by the DD card with the DDNAME ENVOUT. This data set is most usefully a member of a partitioned data set so that the perhaps many environments for a large program can be conveniently stored together.

When an inner (in the sense of block structuring) module is compiled, all the environments for the more outer modules must be available. The most immediately outer environment is specified in a using directive when the current module is compiled. The more outer modules are accessed by the previous using directives that were "copied" to the environment files when those modules were compiled. Thus the same input DDnames must be available for all compilations of the various segments of the program.

A USING directive has the following form:

```
USING handle FROM string
```

where handle is the 'handle identification' (an indicant) and string is a string denotation. The string is the idf used for opening the file containing the previous environment.

This sounds very complicated, but is in fact very simple, especially when a pds is used for saving environments - see the example in 13 below.

12. Overlays

Separately compiled segments may be overlaid using the facilities of OS/360 and its linkage editor. However, overlaying must be done with some care as it is possible for objects in the store that is being overlaid to be pointed at from somewhere else in the store. It is difficult to describe formally when this condition could arise as it is highly dependent on the implementation of copying of items (which could change from one version of the compiler to another). In general, it will not be safe if

a) the object is yielded by the overlayable segment or the object is assigned within the overlayable segment to a name having a scope greater than that segment,

and b) the object occurs within an overlayable segment and the object is a STRING denotation or a routine text, or if it is a row- or a structure-display, or it derives from one of these (e.g. by ascription),

or c) control is transferred between overlayable segments other than by elaborating ENVIRON (e.g. a segment is entered by elaborating ENVIRON, but return from a segment does not).

Examples:

1) containing assignments:

```
PROC INT p;
```

```
        ENVIRON ONE;

        when the segment associated with the handle contains

        p := INT:( ... );

        such a segment cannot be safely overlaid within the scope of p.

        2) yielding values:

        PROC INT p = ENVIRON TWO;

        The segment associated with the handle cannot be safely overlaid
        within the scope of p.

        Note that in

        PROC p = INT: ENVIRON THREE;

        the segment associated with the handle yields an INT value and,
        other things being equal, may be overlaid safely within the
        scope of p.
```

It is intended that, in future, overlays will be implemented correctly or that a message will be issued when any unsafe condition arises.

13. Example job using separate compilation and overlays

```
//*
//* PREALLOCATE DATA SETS
//*
// EXEC PGM=IEFBR14,REGION=8K,TIME=(,1)
//A DD DSN=ABCD.ENV,DISP=(,KEEP),UNIT=SYSDA,SPACE=(TRK,(1,1,1)),
//      DCB=RECFM=VB
//B DD DSN=ABCD.LIB,DISP=(,KEEP),UNIT=SYSDA,SPACE=(TRK,(1,1,1))
//*
//* ROOT MODULE
//*
// EXEC A68CL,
// ENVOUT='ABCD.ENV(A)',           ENVIRONMENT GENERATED
// DISPL=OLD,NAMEL='ABCD.LIB(A)'   LOAD MODULE
//A68.SYSIN DD *
TITLE root segment  #title for compiler diagnostics#
NAME "A"            #CSECT name for segment#
print("root");
ENVIRON ONE;
ENVIRON TWO
//*
//* FIRST BRANCH
//*
// EXEC A68CL,
// ENVOUT='ABCD.ENV(B)',           ENVIRONMENT GENERATED
// DISPL=OLD,NAMEL='ABCD.LIB(B)'   LOAD MODULE
//A68.ENVIN DD DSN=ABCD.ENV,DISP=SHR  TO ACCESS PREVIOUS ENVIRONMENT
//A68.SYSIN DD *
TITLE branch 1 NAME "B" USING ONE FROM "ENVIN(A)"
```

```
print("b");
STRING s = ENVIRON THREE;
print(s)
//*
//* TWIG
//*
// EXEC A68CL,
// DISPL=OLD,NAMEL='ABCD.LIB(C)'
//A68.ENVIN DD DSN=ABCD.ENV,DISP=SHR
//A68.SYSIN DD *
USING THREE FROM "ENVIN(B)"
" c "
//*
//* SECOND BRANCH
//*
// EXEC A68CL,
// DISPL=OLD,NAMEL='ABCD.LIB(D)',
// PARMC='/USING TWO FROM "ENVIN(A)"'
//A68.ENVIN DD DSN=ABCD.ENV,DISP=SHR
//A68.SYSIN DD *
print("d")
//*
//* LKED (OVERLAYING) AND EXECUTE
//*
// EXEC A68LG,ATTL=OVLY
//LKED.SYSLIB DD
//            DD
//            DD DSN=ABCD.LIB,DISP=SHR
//LKED.SYSIN DD *
 OVERLAY ONE
 INCLUDE SYSLIB(B,C)
 OVERLAY ONE
 INCLUDE SYSLIB(D)  NOTE ROOT IS AUTOMATICALLY LOADED
```

14. Error messages

The error messages from the compiler should be self explanatory, at least with the help of the ALGOL68C Reference Manual to explain technical terms. Occurrences of "System error" and "Consult expert" messages should be reported to the ALGOL68C maintenance group (see 3 above).

Error messages from the translator should not occur frequently; the most likely messages are "program too large", or "segment exceeds 12k" in which case the program should be further segmented (The translator will not produce an object module occupying more than 12k because only 3 base registers are used.) "System error" messages from the translator may also occur, please report as above.

The messages from the linkage editor are described by IBM.

The run-time diagnostics need considerable improvement - this will happen in due course. Messages produced at run time include:

"no storage for generator" a heap or local generator was elaborated and insufficient storage was available. Increase the region size. Note that the 'static' stack size is not checked so that the static and dynamic stacks (or the

heap) may clash causing miscellaneous errors (but usually abends 0C4, 0C5 or 0C1).

Note that, on the 360/370, the dynamic stack and the heap are allocated from the same area of storage (the high address end of the region). Mixed use of the heap and the dynamic stack can prevent storage on the dynamic stack from being recovered. The heap is used for all HEAP generators and for dynamically constructed strings (strictly, for the characters contained within the string) and for OS data control blocks. The dynamic stack is used for array elements, and for explicit LOC generators (but this does not include sample-generators). Thus '*ƒ*1:4„INT a' will cause the elements of 'a' to be placed on the dynamic stack, 'REF INT b = LOC INT' will cause 'b' to be on the dynamic stack, but 'LOC INT c' will cause 'c' to be on the static stack.

The following program will cause the dynamic stack allocated for the elements of 'a' not to be recovered:

```
BEGIN
   PROC p = VOID: BEGIN ƒ1:5„INT a; q END;
   PROC q = VOID: BEGIN HEAP INT b; SKIP END;
   p
END
```

"file ended" this message is output by the default file mended routine before it terminates the program. The user has attempted to read past the last character of the file. Either do not read too far, or substitute a new file mended routine. N.B. a call of 'file ended' before reading will not normally be useful as the file is not ended until the user reads past the last line in the file. A useful technique is to assign a routine delivering false and to check for file ended immediately after reading, e.g.

```
on file end(file, (REF FILE f)BOOL: FALSE);

        . . .

read(c); IF NOT file ended(file)
         THEN  . . .
```

Alternatively, although perhaps less elegantly, a jump from the file mended routine may be made, e.g.

```
on file end(file,
            (REF FILE f)BOOL: (GOTO endfile; SKIP));

        . . .

read(c)

        . . .

endfile: . . .
```

"integer out of range for ELEM ... " ELEM for string, bits, or bytes has been called and the 'index' integer is out of range, i.e. zero or negative, or too large.

"obeying faulty program" occurs if the part of a program that caused a compiler error message is elaborated.

Other runtime error messages produced by the library should be self-explanatory. The usual ABENDs may of course occur. In some future release, these will be trapped with STAE and SPIE macros to produce a more meaningful diagnostic.