CLM-P 398

OLYMPUS AND PREPROCESSOR PACKAGE FOR AN IBM 370/165

by

M H Hughes, P D Roberts* and K V Roberts

(Submitted for publication in Computer Physics Communications)

ABSTRACT

(Page 1 of the text is a summary of the paper)

*MoD(PE), Aldermaston, Berkshire

UKAEA Research Group
Culham Laboratory
Abingdon
Oxfordshire

June 1974

Title of Program: OLYMPUS

Catalogue number:

Computer for which the program is designed and others upon which it is operable:

Computer: IBM 370/165 Installation: AERE, Harwell

Operating system under which the program is executed: HASP

Programming language: Standard FORTRAN

High speed store required: 72 Kbytes

Is program overlaid? No

Number of magnetic tapes required: None

What other peripherals are used? disk, line printer

No. of cards in combined program and test deck: 2412

Card punching code: EBCDIC

CPC Library programs used: None

Keywords descriptive of problem: Preprocessor, Control, Utility, Package, Initial-value problems, Simulation, Computation, Standard, Framework, Kernel.

Nature of Problem

A standard methodology called the OLYMPUS system has been established for constructing, testing and operating Fortran programs which solve equations describing initial-value problems. A previously-published control and utility package [1] implements this system for the ICL 4/70, and the present package contains the corresponding version for IBM 360/370 series computers. It also includes a FORTRAN preprocessor which will insert labelled COMMON blocks into a source file using only one master copy of COMMON.

Method of Solution

Ref 1 should be consulted for the details of the OLYMPUS system proper. The preprocessor reads the COMMON blocks into store on one channel NA, the source code on another channel NB, and outputs the expanded program as a data set on a third channel NC which is then passed to the compiler.

Restrictions

Different implementations of OLYMPUS are required for different types of computer to take account of differing wordlengths, channel numbers etc. The present package should work on most IBM 360 and 370 systems provided that the control cards are modified to suit local conventions. Since the preprocessor is written in Standard FORTRAN it can, in principle, be used on any computer. However, it has been designed to take advantage of the IBM scheme for concatenating data sets.

LONG WRITE-UP

1. INTRODUCTION

A previous paper [1] has described the OLYMPUS control and utility package for initial value FORTRAN programs and its implementation on an ICL System 4/70 computer at Culham Laboratory. Except for the details of control cards and channel numbers the discussion in ref[1] applies equally well to the present package. However, OLYMPUS programs developed on that machine usually make extensive use of a local facility for file substitution. The facility takes the form of a preprocessor and extends that introduced at Culham several years ago for an IBM 1401/STRETCH system [2], and subsequently incorporated into the ICL KDF9 EGDON system where the Job Organizer automatically intercepted control statements of the form

```
*SUBSTITUTE B
```

and inserted file B into another file A at the point where the statement appeared. The format is slightly different for the ICL System 4/70 preprocessor where the control statement is (where ␣ denotes a blank):

```
//␣SUBSTITUTE␣B
```

On the ICL systems the substitution is recursive and allows files to be manipulated symbolically. The facility is extremely powerful and has many applications. One important application is to FORTRAN COMMON blocks where only one master copy of each block need be supplied however many subroutines there are to be compiled.

To facilitate the movement of OLYMPUS programs from the ICL System 4/70 to IBM 360/370 and similar machines, we have devised a very simple preprocessor for file substitution. The program is written entirely in Standard FORTRAN IV. However, since it is designed primarily for handling COMMON blocks we have not allowed for recursive substitution. To illustrate the use of the processor we have used it to implement the OLYMPUS system on an IBM 370/165. A description of the preprocessor and the organization of OLYMPUS on the IBM 370 is presented below.*

---

*Reference to the IBM 370 should apply equally well to the IBM 360.

2. THE PREPROCESSOR

The principle of operation of our preprocessor is quite simple and involves essentially two steps. First, we read into core from one channel NA all the blocks to be substituted. We have reserved sufficient store for a total of 300 lines for substitution; if this number is exceeded the preprocessing is terminated with an error message. Each block must be preceded by a control statement of the form

//␣BLOCK␣NAME␣← identifier →

where, to maintain compatibility with the ICL System 4/70 the block identifier can comprise up to 28 characters. It is not necessary to delineate the end of a block. To assist with the handling of COMMON blocks various convenience features are included on channel NA. These are described in detail in Section 3.

Each time the keywords BLOCK␣NAME are encountered on channel NA the block identifier is entered in a table. We have allowed for 100 entries in the table; again, abnormal termination with an appropriate error message will result if this number is exceeded.

Secondly, having encountered the end- of-file marker on channel NA we read the source deck on a second channel NB. We immediately output each line on a third channel NC. However, if the control statement

//␣SUBSTITUTE␣←identifier→

is encountered on input we check that the identifier is in the table of blocks available and, if it is found, we copy that block on channel NC in place of the SUBSTITUTE card. The expanded data set so constructed on channel NC can subsequently be passed to another job step.

The channel numbers NA, NB and NC are preset to 8, 9 and 10 respectively. At run time these channels must be defined by control cards of the form

```
//G.FT08F001␣DD␣DATA
      COMMON blocks
/*
//G.FT09F001␣DD␣DATA
      Source code
/*
//G.FT10F001 DD DSN=-----,
//␣VOL=-----, SPACE=-----, DCB=-----,
//␣DISP=(NEW,PASS)
```

3. ADDITIONAL FACILITIES

To add some flexibility to the handling of COMMON blocks a number of facilities are incorporated in our preprocessor. These facilities are designed to exploit the standard IBM concatenation scheme which allows any number of data sets to be combined.

3.1 Copying data sets from the input stream

Unfortunately, at our installation the HASP system prevents us from concatenating data sets directly from the input stream. To circumvent this curious restriction the OLYMPUS package for IBM 370 contains a member called COPY which simply copies a data set from the input stream to a temporary file on the disk. At some IBM 360/370 installations this facility will be redundant. Some of the examples shown below assume that temporary input data sets were created in a previous job-step. Figure 1 illustrates the control cards to initiate a copy from the input stream to a data set which is subsequently passed to another job-step.

```
//STEPX␣EXEC␣PGM=COPY
//STEPLIB␣DD␣DSN=LOAD.CUL.OLYMPUS,DISP=(OLD,SHR),VOL=----
//G.FT05F001␣DD␣DATA
-----cards to be copied-----
/*
//G.FT06F001␣DD␣SYSOUT=A
//G.FT07F001␣DD␣DSN=TEMP,DISP=(NEW,PASS),
//            VOL=-----,SPACE=----,
//            DCB=(RECFM=FBS,LRECL=80,BLKSIZE=80)
```

Fig 1. Copy data set from input stream

3.2 Permanently saving blocks

It is often convenient to save the COMMON blocks associated with a program in a permanent data set. Thus, the occurrence of a card

```
//␣SAVE
```

causes all subsequent blocks to be saved on channel NSAVE which is preset to 7; previous blocks are not saved. Channel NSAVE must, of course, be declared on control cards. Figures 2 and 3 indicate the control cards necessary when the preprocessor is used : (a) to create a new program COMMON and use it ; (b) to permanently add or alter certain blocks and use them. In each example, the

OLYMPUS COMMON blocks are stored in a permanent data set called *DATA.CUL.OLYMPUS.COMMON while the program blocks are saved in PROGRAM.COMMON; the temporary data set TEMP is generated in a previous step.

```
//G.FT07F001 DD DSN=PROGRAM.COMMON,DISP=(NEW,KEEP),
//           VOL=----,SPACE=----,
//           DCB=(RECFM=FBS,LRECL=80,BLKSIZE=80)
//G.FT08F001 DD DSN=DATA.CUL.OLYMPUS.COMMON,DISP=(OLD,SHR),VOL=----,
//            DD DSN=TEMP,DISP=(OLD,DELETE)
```

where the temporary data set TEMP contains the card images

```
// SAVE
          ----program COMMON----
```

Fig.2 Create new program COMMON and use it

```
//G.FT07F001 DD DSN=PROGRAM.COMMON1,DISP=(NEW,KEEP),-----
//G.FT08F001 DD DSN=DATA.CUL.OLYMPUS.COMMON,DISP=(OLD,SHR),VOL=----,
//           DD DSN=TEMP,DISP=(OLD,DELETE),VOL=----,
//           DD DSN=PROGRAM.COMMON,DISP=(OLD,KEEP),VOL=----
```

where TEMP contains

```
// SAVE
          ---- new cards or amendments ----
```

Fig.3. Permanently add or alter certain blocks and use them

3.2 Temporarily add or alter certain blocks

The processor is designed such that if a block name appears more than once, the second and all subsequent versions are ignored. Figure 4 shows how, for diagnostic purposes, we can exploit this feature to temporarily add or alter certain blocks.

```
//G.FT08F001 DD DSN=DATA.CUL.OLYMPUS.COMMON,DISP=(OLD,SHR),VOL=----,
//           DD DSN=TEMP,DISP=(OLD,DELETE),VOL=----,
//           DD DSN=PROGRAM.COMMON,DISP=(OLD,KEEP),VOL=----
```

where TEMP simply contains the amendments

Fig.4. Temporarily add or modify blocks

*The prefix DATA.CUL. or LOAD.CUL. is a local naming convention

4. ORGANIZATION OF OLYMPUS ON IBM 370

On the ICL System 4/70 [1], the entire OLYMPUS system resides in a private subroutine library; control cards point to the library which the composer (linkage editor) searches for unresolved external references. Although it is possible to construct such libraries on IBM 370 series machines it is generally a rather tedious task. We therefore exploit the hierarchical structure offered by the equally convenient concept of a load module. A load module is the absolute binary data set output from the linkage editor; this module possesses the very useful property that it can be used subsequently as secondary input to the linkage editor to resolve external references in another module. Although this arrangement means that all unsupplied routines are loaded, regardless of whether they are required, the entire OLYMPUS system takes up rather little core store.

The arrangement which we have found most convenient is as follows: The pre-processor and the OLYMPUS library load modules are saved in a single partitioned data set called LOAD.CUL.OLYMPUS; the two members are

LOAD.CUL.OLYMPUS(PROCESOR)

and

LOAD.CUL.OLYMPUS(LIBRARY)

The member LIBRARY contains the skeleton control structure CRONUS [1] together with the entire library of utility routines. The two labelled COMMON blocks COMBAS and COMDDP associated[1] with the OLYMPUS system are saved in another data set

DATA.CUL.OLYMPUS.COMMON

At our installation both data sets reside on a permanently mounted volume having public access. A detailed description of the implementation is given in the next section.

The way in which a problem program utilizes the processor and library facilities is illustrated symbolically in Figure 5. In general there are two job steps involved. Firstly the pseudo-FORTRAN source deck containing //␣SUBSTITUTE cards (S-deck) is read by the preprocessor which creates a valid FORTRAN data set (F-deck) on channel 10. The latter is input to the compiler in the usual way. In Figure 6 we indicate the control cards necessary at our installation to achieve the sequence of Figure 5. We point to the library programs by means of the JOBLIB card which precedes the first EXEC card. The OLYMPUS library of utility routines is included in the user program by means of an INCLUDE card in the linkage editor statements.

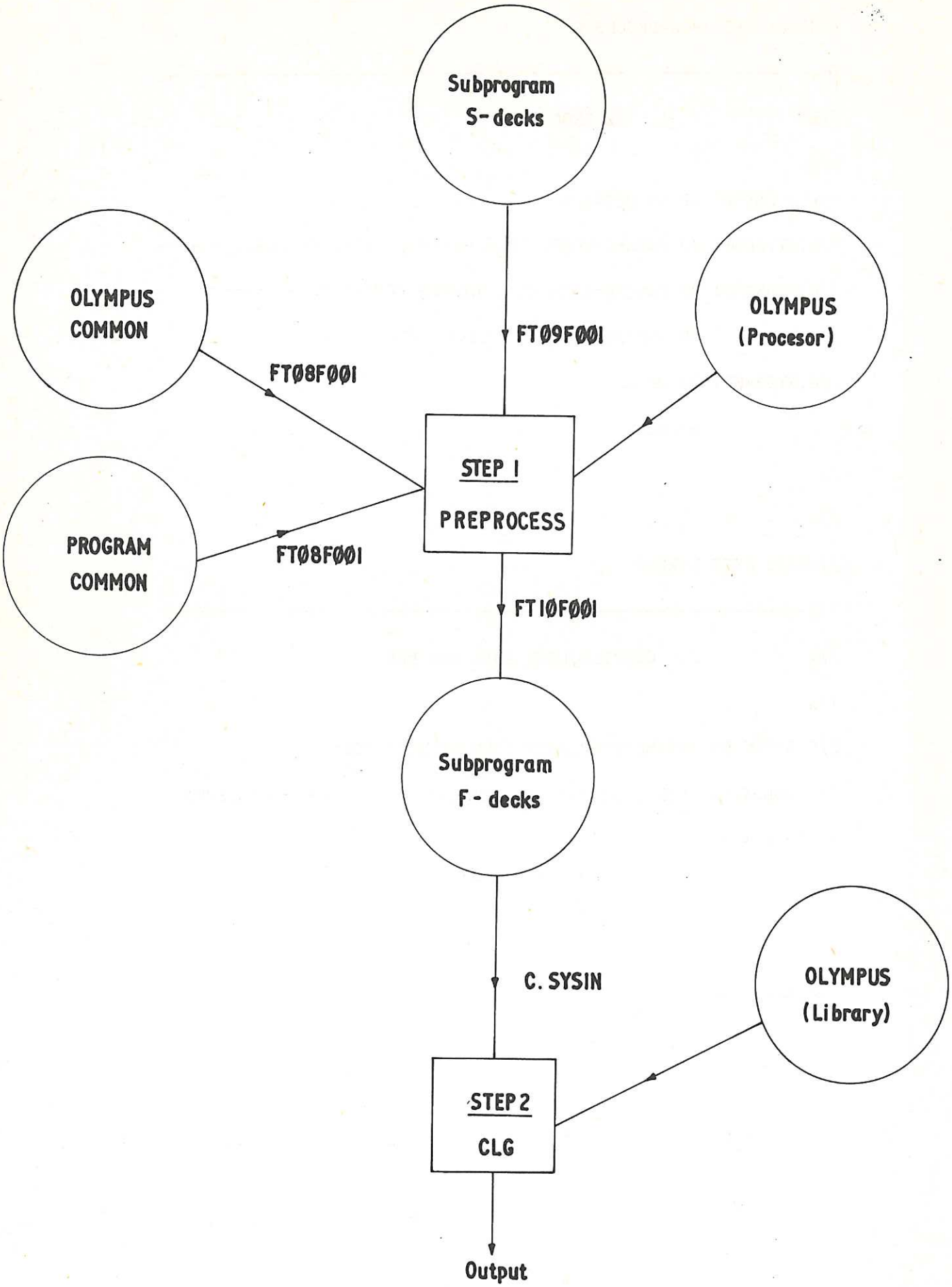

Figure 5. Method of using Preprocessor and Library

```
//JOBLIB DD DSNAME=LOAD.CUL.OLYMPUS,VOL= ----
//STEP1 EXEC PGM=PROCESOR
//*-------------------------------------------------------
//*              1.  PREPROCESS
//*
//G.FTO6FOO1 DD SYSOUT=A
//G.FT1ØFØØ1 DD DSNAME=TEMP, VOL= ------,DISP=(NEW,PASS),------
//G.FTØ8FOO1 DD DSNAME=DATA.CUL.OLYMPUS.COMMON,VOL= -----
//              DD DSNAME=PROGRAM.COMMON,VOL=------
//G.FTØ9FØØ1 DD DATA
              S-deck
/*
//*
//STEP2 EXEC NEWCLG
//*-------------------------------------------------------
//*             2.  COMPILE,LINK EDIT AND RUN
//*
//C.SYSIN DD DSNAME=TEMP,DISP=(OLD,DELETE),VOL=------
//L.DDLOAD DD DSN=LOAD.CUL.OLYMPUS,VOL= ----, DISP=(OLD,SHR)
//L.SYSIN DD *
              INCLUDE DDLOAD(LIBRARY)
              ENTRY MAIN
/*
//G.SYSIN DD *
              program data
/*
```

Figure 6. Typical control cards to use processor and library

(//* indicates a comment)

5. IMPLEMENTATION OF OLYMPUS ON AN IBM 370/165

The OLYMPUS package comprises:

(i) a skeleton control structure called CRONUS which is itself an executable program;

(ii) a library of useful utility routines;

(iii) a test program called MINOS which tests all the utility routines.

In addition there are two standard labelled COMMON blocks called COMBAS and COMDDP together with a block COMTES of test variables for MINOS. These were described in detail in ref [1]. The implementation and testing of OLYMPUS on the IBM 360 comprises 10 job steps; the control cards and the structure of the deck are shown in Figure 7. The individual job steps are considered below.

(1) Compile and link-edit the preprocessor

Referring to Figure 7, the first step involves compilation and link-editing of the pre-processor; we use a catalogued procedure called NEWCL. To save the program we supply a card L.SYSLMOD which overrides the catalogued procedure statement and stores the load module in member PROCESOR of a partitioned data set called LOAD.CUL.OLYMPUS.

(2) Compile and link-edit copy program

The second step, similar to step 1, compiles and link-edits the copy program. The load module is stored in member COPY of the data set LOAD.CUL.OLYMPUS.

(3) Preprocess OLYMPUS

The third step is to insert the COMMON blocks COMBAS and COMDDP into the control and utility routines of the OLYMPUS library. We use the program generated in step 1, using //STEPLIB to point to the appropriate load module. The pre-processor reads the COMMON blocks on channel 8 ; these are copied on channel 7 and saved permanently in the data set DATA.CUL.OLYMPUS.COMMON. The pseudo-FORTRAN source deck is read on channel 9 and the expanded FORTRAN is written to a temporary data set TEMP1 on channel 10.

(4) Copy MINOS COMMON block

In step 4 we copy the block COMTES of test variables associated with MINOS from the input stream to a temporary data set MINOS.COMMON

(5) Preprocess MINOS

This step is similar to step 3. However, we now read the standard COMMON blocks on channel 8 from the permanent data set DATA.CUL.OLYMPUS.COMMON generated in step 2; the MINOS COMMON block is also read on channel 8 from the temporary data set created in the previous job-step. We output to another temporary data set TEMP2 on channel 10.

(6) Compile and link-edit OLYMPUS library

This step again invokes the catalogued procedure NEWCL. The compiler takes its input from the scratch file TEMP1 generated in step 3. The load module output from the linkage editor is saved in a second member LIBRARY of the data set LOAD.CUL.OLYMPUS created in step 1.

(7) Compile and link-edit MINOS

The compiler takes its input from the scratch data set TEMP2 generated in step 5. The //STEPLIB card points to the OLYMPUS library and we explicitly include the member LIBRARY in the linkage editor statements to supply unresolved references.

(8) Run CRONUS

This step executes the skeleton control structure CRONUS; there is no input data required.

(9) and (10) Run MINOS

There are two test cases which are run separately. The output should be compared with the Test Output given in ref [1] to check that the package has been implemented correctly; it is then ready for use.

```
// JOB
//STEP1  EXEC NEWCL
//*--------------------------------------------------------------------
//*              1.          COMPILE AND LINK-EDIT PREPROCESSOR
//*
//C.SYSIN     DD    *

          PRE-PROCESSOR FORTRAN

/*
//L.SYSLMOD   DD    DSN=LOAD.CUL.OLYMPUS(PROCESOR),       LOAD MODULE
//              VOL=REF=CUL,DISP=(NEW,KEEP),
//              SPACE=(CYL,(2,1,2))
//L.SYSIN     DD    *                                     L-EDIT STATEMENTS
          ENTRY MAIN
/*
//STEP2  EXEC NEWCL
//*--------------------------------------------------------------------
//*              2.          COMPILE AND LINK-EDIT COPY PROGRAM
//*
//C.SYSIN     DD    *

          FORTRAN FOR COPY PROGRAM

/*
//L.SYSLMOD   DD    DSN=LOAD.CUL.OLYMPUS(COPY),           LOAD MODULE
//          VOL=REF=CUL,DISP=(OLD,KEEP)
//L.SYSIN     DD    *                                     L-EDIT STATEMENTS
          ENTRY MAIN
/*
//STEP3  EXEC PGM=PROCESOR,REGION=80K
//*--------------------------------------------------------------------
//*              3.          PRE-PROCESS OLYMPUS

//*
//STEPLIB     DD    DSN=LOAD.CUL.OLYMPUS,VOL=REF=CUL,DISP=OLD
//G.FT08F001  DD    DATA

          OLYMPUS COMMON BLOCKS

/*
//G.FT09F001  DD    DATA

          OLYMPUS S-DECK

/*
//G.FT10F001  DD    DSN=TEMP1,                            FORTRAN F-FILE
//              VOL=REF=CUL,DISP=(NEW,PASS),
//              SPACE=(TRK,(5,2),RLSE),
//              DCB=(RECFM=FBS,LRECL=80,BLKSIZE=4000)
//G.FT06F001  DD    SYSOUT=A
//G.FT07F001  DD    DSN=DATA.CUL.OLYMPUS.COMMON,          COMMON FILE
//              VOL=REF=CUL,DISP=(NEW,KEEP),
//              SPACE=(TRK,(1,1),RLSE),
//              DCB=(RECFM=FBS,LRECL=80,BLKSIZE=80)
```

Figure 7. Control cards to create and test OLYMPUS library

```
//STEP4  EXEC PGM=COPY
//*--------------------------------------------------------------------
//*              4.          CREATE DATA SET FOR MINOS COMMON
//*
//STEPLIB     DD    DSN=LOAD.CUL.OLYMPUS,VOL=REF=CUL,DISP=OLD
//G.FT05F001 DD    DATA

          MINOS COMMON BLOCK

/*
//G.FT06F001 DD    SYSOUT=A
//G.FT07F001 DD    DSN=MINOS.COMMON,

//        VOL=REF=CUL,DISP=(NEW,PASS),SPACE=(TRK,(5,2),RLSE),
//        DCB=(RECFM=FBS,LRECL=80,BLKSIZE=80)
//STEP5  EXEC PGM=PROCESOR,REGION=80K
//*--------------------------------------------------------------------
//*              5.          PRE-PROCESS MINOS
//*
//STEPLIB     DD    DSN=LOAD.CUL.OLYMPUS,VOL=REF=CUL,DISP=OLD
//G.FT08F001 DD    DSN=DATA.CUL.OLYMPUS.COMMON,          COMMON FILE
//             VOL=REF=CUL,DISP=(OLD,KEEP)
//             DD    DSN=MINOS.COMMON,VOL=REF=CUL,DISP=(OLD,DELETE)
//G.FT09F001 DD    DATA                                  FORTRAN S-DECK

          MINOS FORTRAN

/*
//G.FT10F001 DD    DSN=TEMP2,DISP=(NEW,PASS),
//             VOL=REF=CUL,SPACE=(TRK,(5,2),RLSE),
//             DCB=(RECFM=FBS,LRECL=80,BLKSIZE=4000)
//G.FT06F001 DD    SYSOUT=A
//STEP6 EXEC NEWCL
//*--------------------------------------------------------------------
//*              6.          COMPILE AND LINK-EDIT LIBRARY
//*
//C.SYSIN     DD    DSN=TEMP1,                            FORTRAN F-FILE
//             VOL=REF=CUL,DISP=(OLD,DELETE)
//L.SYSLMOD   DD    DSN=LOAD.CUL.OLYMPUS(LIBRARY),        LOAD MODULE
//             VOL=REF=CUL,DISP=(OLD,KEEP)
//L.SYSIN     DD    *                                     L-EDIT STATEMENTS
         ENTRY MAIN
/*
//STEP7 EXEC NEWCL
//*--------------------------------------------------------------------
//*              7.          COMPILE AND LINK EDIT MINOS

//*
//*
//C.SYSIN     DD    DSN=TEMP2,                            FORTRAN F-FILE
//             VOL=REF=CUL,DISP=(OLD,DELETE)
//L.SYSLMOD   DD    DSN=MINOS(DISKPGM),                   LOAD MODULE
//             VOL=REF=CUL,DISP=(NEW,PASS),
//             SPACE=(CYL,(2,1,1),RLSE)
//L.DDLOAD    DD    DSN=LOAD.CUL.OLYMPUS,VOL=REF=CUL,DISP=OLD
//L.SYSIN     DD    *                                     L-EDIT STATEMENTS
         INCLUDE DDLOAD(LIBRARY)
         ENTRY MAIN
/*
```

Figure 7. (contd.)

```
//STEP8  EXEC PGM=LIBRARY
//*--------------------------------------------------------------------
//*              8.          RUN CRONUS
//*
//STEPLIB     DD    DSN=LOAD.CUL.OLYMPUS,VOL=REF=CUL,DISP=OLD
//G.FT06F001 DD    SYSOUT=A
//*
//STEP9  EXEC RUNG
//*--------------------------------------------------------------------
//*              9.          RUN MINOS TEST 1
//*
//STEPLIB     DD    DSN=MINOS,VOL=REF=CUL,DISP=OLD
//*
//G.SYSIN     DD    *

        DATA FOR MINOS TEST 1

/*
//*
//STEP10 EXEC RUNG
//*--------------------------------------------------------------------
//*              10.         RUN MINOS TEST 2
//*
//STEPLIB     DD    DSN=MINOS,VOL=REF=CUL,DISP=(OLD,DELETE)
//*
//G.SYSIN     DD    *

        DATA FOR MINOS TEST 2
/*
//
```

Figure 7. (contd.)

6. REFERENCES

1 Roberts K V and Christiansen J P. 'OLYMPUS: A Standard Control and Utility Package for Initial-Value Fortran Programs'. (1973) CLM-P373 Computer Physics Communications Vol.7 No.5 (1974).

2 Roberts K V. 'Scientific Computing and Operational Research', (1965) Culham Report CLM-R 45, available from HMSO.