CLM - P 354

CULHAM LIBRARY
REFERENCE ONLY

This document is intended for publication in a journal, and is made available on the understanding that extracts or references will not be published prior to publication of the original, without the consent of the author.

CLM - P 354

CULHAM LABORATORY
LIBRARY
-8 APR 1974
b
loan.

2

E MINIMIS MAXIMA

UKAEA RESEARCH GROUP

Preprint

AN INTRODUCTION TO THE OLYMPUS SYSTEM

K V ROBERTS

CULHAM LABORATORY
Abingdon Berkshire

1973

Enquiries about copyright and reproduction should be addressed to the Librarian, UKAEA, Culham Laboratory, Abingdon, Berkshire, England

CLM-P354

AN INTRODUCTION TO THE OLYMPUS SYSTEM

K V Roberts

(Submitted for publication in Computer Physics Communications)

ABSTRACT

A standard methodology has been established for the design, construction and operation of Fortran programs to solve initial-value and other problems. The following two papers describe the OLYMPUS library package which is used in implementing this methodology on the ICL 4/70 at Culham, together with an illustrative example of a laser fusion code called MEDUSA 1 which has been developed in this way. Subsequent papers will describe how OLYMPUS can be installed on IBM and CDC machines. This article provides a brief introduction to the OLYMPUS system and explains why it has been adopted.

UKAEA Research Group
Culham Laboratory
Abingdon
Berks.

November 1973

INTRODUCTION

An earlier article [1] in this journal discussed the problem of publishing scientific Fortran programs. It suggested a number of general principles which, if followed, would make it easier to transfer such programs from one user or one computer system to another, and would enable them to be readily adapted to solve problems that were not envisaged at the time when they were first written.

These principles have now been incorporated into a programming system called OLYMPUS [2,3], which is used by the Computational Physics Group at the UKAEA Culham Laboratory to support the design, construction and operation of all new Fortran programs. So far as possible all programs are built according to a common plan, which makes them more intelligible as well as speeding up most aspects of the programming process. The OLYMPUS system was originally developed for physical initial-value codes which solve partial-differential equations of the general form

$$\frac{\partial f}{\partial t} + G(f) = 0 \tag{1}$$

subject to appropriate initial and boundary conditions, where f is the solution vector and G is a linear or non-linear operator. Many interesting problems in hydrodynamics, plasma physics, astrophysics and other areas of classical physics can be expressed in this form, and it seemed worthwhile to establish a common programming strategy for dealing with them. Once the OLYMPUS system had been developed, however, it was found to be equally useful with only minor changes for a much larger class of computing work, including programs which have no connection with physics at all, for example the automatic documentation and flowcharting codes discussed in ref.[1].

OLYMPUS employs a standard set of files which are installed on-line in a system or user library. These files include a main program, subprograms, COMMON blocks, procedure files and so on. The following paper [2] describes the package which is used to build and test the system on the ICL 4/70 used at Culham, while refs. [4 & 5] describe the corresponding packages which are used for the IBM 360/370 series and the CDC 6000/7000 series respectively. It is intended that OLYMPUS itself should organize most of the system-dependent features such as channel numbers, word and byte lengths, character codes etc., so that once it has been installed and tested on a new computer system the actual transfer of any program in the 'Olympian' family is relatively straightforward. Ref.[6] in this issue describes one such program, MEDUSA 1, a 1-dimensional Lagrangian laser fusion code. Other members of the family will be published in Computer Physics Communications in due course.

We have found it useful to standardize many of the stages in the design, construction, testing, documentation and operation of Fortran programs. Several aspects of the OLYMPUS system are listed in Table 1 and are briefly described in the sections which follow. A more detailed discussion and justification is given elsewhere [2,3]. Preferably this article should be read in conjunction with a listing of an Olympian code, e.g. MEDUSA 1 [6].

TABLE 1

Aspects of the OLYMPUS System

1.	Architecture	6.	Utility Subprograms
2.	CRONUS Dummy Program	7.	Program Development
3.	Notation and Layout	8.	Control
4.	Documentation	9.	System-Independence
5.	Diagnostics	10.	HESTIA Housekeeping Programs.

1. ARCHITECTURE

We think of each program as consisting of two main parts:

INSTRUCTIONS
DATA

The instructions are organized into a main program together with a set of subprograms, while the data is organized into labelled COMMON blocks. It may be convenient to picture these COMMON blocks as operands and the subprograms as operators which act on them. So far as possible each subprogram and each block should have a well-defined purpose, and we find it useful to adopt the broad classification indicated in Table 2. The decimal numbering scheme used in OLYMPUS will be explained in §3, but meanwhile we note that the subprograms are divided into classes, while the labelled COMMON blocks are divided into groups. Only one copy of each labelled block is used throughout the entire program; it is stored on-line as a private file and inserted where required by means of a preprocessor control statement such as

```
// SUBSTITUTE COMPHY                                    (2)
```

This makes the source code shorter and the data structure easier to understand.

TABLE 2

Program Architecture

INSTRUCTIONS			
	Classes of Subprograms	0	Control
		1	Prologue
		2	Calculation
		3	Output
		4	Epilogue
		5	Diagnostics
		U	Utilities
DATA			
	Groups of Labelled COMMON blocks	1	General Olympus Data
		2	Physical Problem
		3	Numerical Scheme
		4	Housekeeping
		5	I/O and Diagnostics
		6	Text Manipulation

2. THE CRONUS DUMMY PROGRAM

A standard library program called CRONUS is installed on-line in object module or load module form as part of the OLYMPUS system. It contains the subprograms and COMMON blocks indicated in Table 3 and is described in detail in the following paper [2]. CRONUS does not itself perform any computation, but it sets the basic structure for all programs in the OLYMPUS family.

This is achieved in the following way. Subroutine COTROL ⟨0.3⟩ is a standard control subprogram which is used for all Olympian programs and which calls lower-level subprograms with the names shown in columns 1-4 of Table 3. In CRONUS these are simply dummies which perform no actual work; in any 'real' Olympian program they are replaced by programmer-supplied versions with the same names and decimal numbers which automatically supplant the dummy library versions. Thus the programmer would supply a subroutine STEPON ⟨2.1⟩ which organizes the calculation step and calls in other subprograms ⟨2.2⟩,⟨2.3⟩ to do the actual work; a subroutine OUTPUT(K) ⟨3.1⟩ to organize the output, and so on.

TABLE 3

Structure of the CRONUS Dummy Program

SUBPROGRAMS

Class / Subprogram	0	1	2	3	4	5
0	(MAIN)					
1	BASIC	LABRUN	STEPON	OUTPUT	TESEND	REPORT
2	MODIFY	CLEAR			ENDRUN	CLIST
3	COTROL	PRESET				ARRAYS
4	EXPERT	DATA				
5		AUXVAL				
6		INITAL				
7		RESUME				
8		START				

COMMON BLOCKS

[C1.1]	COMBAS	Basic System Parameters
[C1.9]	COMDDP	Development and Diagnostic Parameters

Class 1 which deals with initialization and restart has been planned with some care, since although in many non-Olympian programs these aspects can cause considerable trouble to both programmer and reader alike, we have found that in practice they can largely be standardized and to some extent even constructed automatically.

The Group 1 COMMON blocks [C1.1] - [C1.9] are intended to be standard library versions which are available to all programs. So far only two have been established, containing the variables defined in ref.[2]; others will deal with fundamental physical constants, character codes, and other general-purpose information.

3. NOTATION AND LAYOUT

Standardization of the notation and layout makes the program listing neater and easier to read, and once a new programmer has mastered the rules he can in practice develop and debug codes more quickly, partly because he is relieved from the need to make *ad hoc* decisions and partly because he and his colleagues can readily find their way about the listing and hence locate errors.

Table 4 indicates some of the elements that have now been standardized.

TABLE 4

Notation and Layout

(a) DECIMAL NUMBERING SCHEME

Subprograms, e.g. ⟨2.1⟩ STEPON
Common blocks, e.g. [C3.2] COMTIM
Division of subprograms into sections and subsections
Statement numbers correlated with sections
Line numbers correlated with sections.

(b) NOTATION FOR VARIABLES AND ARRAYS

Initial letters are used to distinguish between { Common/internal
Real/integer/logical
Variable/formal parameter/index

(c) LAYOUT

Standard columns
Spacers and ruled lines

(d) SYMBOLIC NOTATION

Channel numbers	Table sizes
Character codes	Constants
Dimensions	

(e) STANDARDIZATION

Control variables	File names
Subprograms	Fundamental constants
Common blocks	

Decimal numbering has been found particularly useful, giving the program roughly the same structure as that of a well-planned mathematical textbook. In addition to the numbering of subprograms and COMMON blocks we also divide an individual subprogram into sections and subsections, and correlate these with Fortran statement numbers (and also line numbers, when working on-line).

The initial-letter conventions defined in Table 5 allow the reader to see at a glance which variables and arrays are in COMMON and which are local, thus avoiding possible mistakes when introducing extensions to the program. The

TABLE 5

Initial Letters and Array Names

	REAL,COMPLEX	INTEGER (and HOLLERITH)	LOGICAL
Subprogram dummy arguments	P	K	KL
Common variable and array names	A-H,O,Q-Y	L,M,N	LL,ML,NL
Local variable and array names	Z	I	IL
Loop indexes		J	

methodical use of symbolic rather than arithmetical notation enables parameters to be readily updated should the need arise as well as making their meaning clearer.

4. DOCUMENTATION

It is not difficult to make a program intelligible if the need for this is foreseen from the outset, but documentation of an existing program is likely to involve much more effort, particularly if it is not properly structured so that awkward features have to be explained to the reader. Table 6 lists some standard documentation tools. Each numbered section or subsection (§3) is preceded by an explanatory heading, and other comments are used to explain the purpose of individual details of the code. Headings, comments and statements begin in different standard columns in order to distinguish them more clearly. Indexes of variables, arrays and subprograms are contained in standard files in alphanumeric or decimal order and printed as required. A useful technique is to cross-reference the listing and a program commentary or write-up so that the two can be read together. This can be seen in the MEDUSA 1 code [6], where

TABLE 6

Documentation Techniques

Headings
Comments
Indexes
Program commentary
References, equation numbers, cross-references
Program map

Automatic Documentation

Clarifier
Flowcharter
Selective printing of given statement types.

the listing refers to the equation numbers in the write-up while the write-up refers to the subprogram numbers in the code. Another example of similar documentation techniques used for IBM 360/370 Assembler Language is available in ref.[7] .

Automatic documentation tools [1] are being developed for use with OLYMPUS and other programs and will be published in due course.

5. DIAGNOSTICS

A fairly elaborate set of diagnostic tools has been developed in order to enable programs to be checked out as quickly as possible and preferably on-line. Each program is 'instrumented' while it is being written, using a set of standard utility routines discussed in §6, and then the diagnostics can be switched on and off either by including coded data in a NAMELIST input deck or by inserting a few extra statements in the program. Facilities are available (Table 7) to output messages, to trace the flow of the program, to print

TABLE 7

Diagnostic Techniques

Messages from program
Flow tracing
Print variable and array names and values
Print selected COMMON blocks
Time sections of program.

the names and values of individual real, integer, logical or Hollerith variables or arrays, to print out COMMON blocks in alphanumeric order, to time sections of the program [8], and to switch individual subprograms off if they are found to contain catastrophic errors.

6. UTILITY SUBPROGRAMS

A set of library subprograms called CYCLOPS enables certain standard facilities to be made available to all programs. A typical example is SUBROUTINE MESAGE (KMESS) which prints a 48-character message on the current output channel. This is done by including a single card in the deck, thus relieving the programmer of the need to handle Format statements which may differ from one machine to the next and can lead to errors. Ref.[2] lists all the CYCLOPS subprograms which are available so far, while Table 8 also mentions others that are available at Culham or are in preparation.

TABLE 8

Utility Subprograms

A. CYCLOPS

1. Output
2. Array manipulation
3. Clock
4. Diagnostics

B. Other Utilities

5. Timing
6. Dump and restart
7. Assembler-language facilities in Fortran
8. Character handling
9. Graphics.

7. PROGRAM DEVELOPMENT

The OLYMPUS system enables programs to be developed and tested in a methodical way, either working off-line with cards or on-line using a multi-access system. Table 9 indicates some of the techniques that are commonly used. It is recommended that the data structure should be checked out first,

TABLE 9

Program Development Techniques

1. Use CRONUS as test-bed
2. Use prefabricated elements where possible
3. Compilations and small tests on-line
4. OLYMPUS provides standard diagnostics which can be switched on and off
5. Check individual modules first, with small array sizes and few timesteps
6. Begin by establishing and checking the data structure
7. CYCLOPS routines provide temporary output

Automatic Tools

8. Construction of procedure files
9. Generation of standard sections of program
10. Updating COMMON blocks
11. Editing.

using CRONUS as a test-bed with its dummy STEPON ⟨2.1⟩ and the diagnostic routines for temporary output. Once the data initialization has been checked, the program can be run for one or two timesteps. All these initial tests can be carried out with small array sizes in order to limit the output, particularly when working on-line. Output routines can of course be checked out independently of the rest of the program, again using CRONUS as a test-bed.

Facilities are being introduced for carrying out some of the routine development work automatically.

8. CONTROL

Many large physics programs are in a continual state of development, since ad hoc modifications have to be introduced to enable particular numerical experiments to be carried out. Unless precautions are taken this can lead to untidy coding and to multiple copies of on-line source and object files. Some thought has therefore been given to the way in which Olympian calculations should be controlled and ad hoc modifications made (Table 10).

TABLE 10

Control of Calculations

1. Standard main program ⟨0.0⟩ and COTROL ⟨0.3⟩
2. Default values set in PRESET ⟨1.3⟩
3. NAMELIST data input in DATA ⟨1.4⟩
4. Function subprograms with parameters in COMMON
5. EXPERT ⟨0.4⟩ for ad hoc modifications.

Firstly, all programs share the same main program ⟨0.0⟩ and the same master control subprogram COTROL ⟨0.3⟩ which are available on-line in object module form.

Secondly, all COMMON variables which can be independently set are assigned default values in subprogram PRESET ⟨1.3⟩, so that the user only has to define those values which he wishes to alter.

Thirdly, we employ NAMELIST data input in subprogram DATA ⟨1.4⟩, since this enables the user to specify the alterations in a convenient symbolic, free-format notation. An Olympian program will usually run with no data input at all, other than the 4 standard cards which label the run [2]. An alternative method is to define the data by means of Fortran statements and then to recompile subprogram DATA each time, but this is awkward on the ICL 4/70 because of the time taken to compose (link-edit) the program after one or more of the subprograms has been recompiled.

Fourthly, we define any arbitrary functions (e.g. initial values) by means of function subprograms or statement functions containing adjustable parameters which are placed in COMMON. If a user wishes to alter the form of the function he recompiles that small section of the program; otherwise he inserts the appropriate parameter changes in the NAMELIST data input.

Finally, ad hoc additions or modifications to sections of the code are made using a subprogram called EXPERT ⟨0.4⟩, which is called from many points throughout the program with 3 parameters KCLASS, KSUB, KPOINT which define the location for which the call was made. The user can then provide his own version of EXPERT which contains the additional or modified code, the transfer of data being made via COMMON. An advantage of this scheme is that all the changes are defined in the compilation listing of EXPERT which should of course be laid out neatly, while the original program remains unchanged in object-module form. This enables several people to use a program at the same time for independent calculations without interfering with one another. Subprogram EXPERT also controls the diagnostics and may be useful for other purposes.

9. SYSTEM-INDEPENDENCE

Table 11 indicates the techniques that are used in order to enable programs to be transferred readily from one computer system to another. The OLYMPUS package installs the system on a new computer and tests it out. It is available for IBM [4] and CDC [5] machines, although some changes may need to be made in the control cards for particular installations; these are fairly complex since several job steps must be scheduled one after another and library files require to be created and preserved. For other types of machine it may be necessary to change some of the Format statements which depend on word-length and to adjust channel numbers.

TABLE 11

Techniques for achieving system-independence

1. Modify OLYMPUS package and install in system or private library.
2. Most Olympian problem programs will then run without modification.
3. Standard Fortran is used where possible.
4. System-independent features are handled symbolically to allow rapid changes.
5. A standard set of tests is supplied with the OLYMPUS package and with each individual program.

Once the package has been installed, most Olympian programs should compile and run straightaway provided that their control cards are changed in the appropriate fashion. To facilitate this, Standard Fortran is employed wherever possible, and system-dependent features such as channel number, wordlengths and character codes are handled symbolically where possible. A standard set of tests is supplied with each program to check that any modifications that are necessary have been carried out correctly.

Some programs will require the use of subprograms written in assembler language for efficiency; so far as practicable the intention is to make these part of the OLYMPUS system, so that equivalent versions are available for each type of machine.

10. HESTIA HOUSEKEEPING PROGRAMS

A considerable degree of automation becomes possible once coding techniques have been standardized, and this is the function of the HESTIA series of housekeeping programs which is currently being developed. Typical examples are FORIBM and FORCDC which convert ICL 4/70 programs to IBM and CDC form respectively and punch them out with all necessary control cards. We hope to publish some of the programs in this series in due course.

References

[1] 'The Publication of Scientific Fortran Programs', K V Roberts, Computer Physics Communications 1, 1 (1969).

[2] 'OLYMPUS, A Standard Control and Utility Package for Initial-Value Fortran Programs', J P Christiansen and K V Roberts. CLM-P373, November 1973. To be published in Computer Physics Communications.

[3] 'OLYMPUS System Part I: Design and Construction of Fortran Programs', K V Roberts and J P Christiansen (1973), to be published as a Culham Laboratory Report, available from HMSO.

[4] 'OLYMPUS and Preprocessor Package for an IBM 370/165', M H Hughes, P Roberts and K V Roberts, Computer Physics Communications (to be submitted).

[5] 'OLYMPUS package for a CDC 6500', M H Hughes, Computer Physics Communications (to be submitted for publication).

[6] 'MEDUSA, A One-dimensional Laser Fusion Code', J P Christiansen, D E T F Ashby and K V Roberts. CLM-P374, November 1973. To be published in Computer Physics Communications.

[7] 'Data Organization for 3-dimensional Calculations on the IBM 360/91 using High Speed Drum Transfers', G Kuo-Petravic, M Petravic and K V Roberts, Culham Laboratory Report CLM-R118, available from HMSO.

[8] 'TIMER - A Software Instrumentation Routine for Making Timing Measurements', M H Hughes and A P V Roberts, Computer Physics Communications (submitted for publication).

[9] 'Standard Fortran Programming Manual', Computer Standards Series, National Computing Centre Ltd., Manchester, England (1970).