CLM-P690

CLM-P690

UKAEA

Preprint

CULHAM LABORATORY
LIBRARY
24 FEB 1983
b

THE OLYMPUS FORTRAN COMPOSITOR

M. H. HUGHES
K. V. ROBERTS

CULHAM LABORATORY
Abingdon Oxfordshire

1982

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publication in a journal or at a conference and is made available on the understanding that extracts or references will not be published prior to publication of the original, without the consent of the authors.

Enquiries about copyright and reproduction should be addressed to the Librarian, UKAEA, Culham Laboratory, Abingdon, Oxon. OX14 3DB, England.

CLM-P690

THE OLYMPUS FORTRAN COMPOSITOR

M H Hughes and K V Roberts

Culham Laboratory
Abingdon, Oxford, OX14 3DB, UK
(EURATOM/UKAEA Fusion Association)

ABSTRACT

This article describes the purpose,structure and use of the COMPOSITOR, which is a word-processing facility that is designed to assist in the construction and maintenance of Fortran programs that conform to the OLYMPUS Standards. It can also be used to tidy up existing Fortran programs.

(to be published in Computer Physics Communications)

November 1982

Title of program: COMPOS

Catalogue number: ACEA

Program obtainable from: CPC Program Library, Queen's University of Belfast, N.Ireland (see application form in this issue)

Computer: PRIME 750. Installation: Culham Laboratory. The program can be used on any computer equipped with the OLYMPUS system, and IBM support routines are provided.

Operating system: PRIMOS

Programming language: ANSI Fortran 66

High speed storage required: 3 PRIME segments of 64K words each. (Minimum size program).

No. of bits in a word: 32

Peripherals used: disc

Number of lines in combined program and test deck:

CPC Library programs used (alternatives):

Catalogue No:	*Title*	*Ref. in CPC*
ABUF	OLYMPUS (ICL 4/70)	7 (1974) 245
ABUJ	OLYMPUS (IBM 370/165)	9 (1975) 51
ABUK	OLYMPUS (CDC 6500)	10 (1975) 167

Keywords

OLYMPUS, Fortran, Documentation, Automatic Code Generation, Word-processing, Text formatting.

Nature of problem

The COMPOSITOR is a word-processing program that converts a free-format Fortran input file to standardized OLYMPUS form. It is used for the semi-automatic construction and maintenance of OLYMPUS software, and can also be used for tidying-up existing Fortran codes in order to make them better structured and more readable. A variety of comment styles is allowed for.

Method of Solution

The input file is read in A-format, line by line, and temporarily converted into unpacked integer format for processing. It is then stored in packed format prior to output.

Restrictions on the complexity of the problem

The COMPOSITOR is written in OLYMPUS form in ANSI Fortran 66 and should run on any type of computer system provided that the OLYMPUS system is installed and that suitable character packing and unpacking routines are available. Versions of these routines are provided for the PRIME and IBM computers. It mainly handles ANSI Fortran 66 code but could readily be extended to deal with ANSI Fortran 77 or dialect statements. It processes a sequence of subprograms one-by-one. The table sizes can be extended if required.

Typical running time

3 seconds/100 lines of input on the PRIME 750.

LONG WRITEUP

1. INTRODUCTION

The inaugural article in this Journal [1] described and analysed a number of techniques to facilitate the understanding of Fortran programs. It was suggested that a program listing should be given the general structure and appearance of a textbook in order to make it as intelligible as possible to the reader. These ideas were subsequently formalized within the OLYMPUS Fortran Programming System [2,3]:(see also other references in the preceding paper [4], denoted in the following by II). In fact, OLYMPUS consists of a complete set of prescriptions for the construction, documentation and operation of portable Fortran programs. Experience with the OLYMPUS system has recently been reviewed [5].

Just as with a textbook, it is necessary to conform to a precise set of typographical conventions if the layout is to be neat, consistent and readable. These are defined in the preceding paper II. Although the conventions adopted in OLYMPUS may seem rather detailed for manual use, the work does not necessarily have to be done by hand at all, since once a precise set of conventions has been established a substantial degree of automation becomes possible. Thus a number of OLYMPUS processors have been developed whose input is intended to be in a progressively more relaxed style, eventually relieving OLYMPUS programmers of much of the routine work of program construction and documentation.

This article describes one such processor, the OLYMPUS Fortran COMPOSITOR, whose purpose is to ensure the preferred standard layout of individual subprograms during:

(i) The initial construction, editing or updating of OLYMPUS programs.

(ii) The conversion of existing, arbitrary Fortran to a tidier form that is more readable and easier to maintain.

Like the chapters in a textbook, OLYMPUS subprograms usually have a standard decimal structure of sections and subsections each with an appropriate heading, (see e.g. II, §3 and Fig.12), and the COMPOSITOR is analogous to the type of text formatting utility that is nowadays often used in the preparation of decimally-structured reports, e.g. the RUNOFF facility [6] on the PRIME computer system at the Culham Laboratory. It is expected that it will assist in the further application of the OLYMPUS system by making it easier to unite new programs, and at the same time be of considerable practical help in a semi-automatic tidying up of a substantial body of existing Fortran code. How to do this is discussed in an earlier paper in this issue [7], denoted in the following by I.

The subsequent article [8], denoted by IV, describes a companion processor called GENSIS, the OLYMPUS Fortran Generator, which automatically constructs indexes and other OLYMPUS documentation, COMMON blocks, and certain utility subroutines.

The present version of the COMPOSITOR is written in ANSI Fortran 66 [9] and should run on any type of computer system that is equipped with the OLYMPUS Control and Utility Package [3]. A small number of input/output and character - handling routines must be implemented in dialect Fortran or in assembler language for a specific type of system, and these are segregated in a support package outside the main Standard Program File (SPF). Versions are provided for the PRIME and IBM systems, and it is explained in Appendix 1 how they should be implemented for other types of computer.

This version of the COMPOSITOR is mainly intended to process programs written to the ANSI Fortran 66 standard, and may therefore require some minor enhancements in order to deal with the additional types of statement that occur in particular Fortran dialects, or in ANSI Fortran 77 [10]. The code is written in OLYMPUS form in such a way that these enhancements can readily be made, and suggestions on how to do this using the EXPERT facility are described in Appendix 2.

It is hoped in due course to implement OLYMPUS in ANSI Fortran 77 [10], and to produce a version of the COMPOSITOR that is written in Fortran 77 and processes Fortran 77 programs. In this case many of the problems of input/output, character handling and dialect statements that are familiar in Fortran 66 should not arise. It is also hoped subsequently to develop versions of OLYMPUS and its utility programs for other languages such as ADA and Algol 68 since many of the techniques and conventions are of general application.

The COMPOSITOR is intended for interactive on-line use, and some of the on-line commands that are explained in this article are those that apply to the PRIME computer system at Culham. It should be relatively straight-forward to build it into other interactive systems and it could also be used in batch mode.

Section 2 discusses the philosophy of on-line program development, illustrated by the short example, shown in the Test Runs, Figs.4-7, and it is suggested that the style in which the programmer finds it most convenient to type and the style that he likes to read need not be the same, thus giving an extra degree of freedom which word-processor can exploit. In Section 3 we explain how to use the COMPOSITOR in the development of OLYMPUS programs,while in I we show how it can be employed to convert non-OLYMPUS programs to a more structured and readable form without changing the working of the executable statements.

Section 4 outlines the overall structure of the COMPOSITOR program which is similar to that of an OLYMPUS physics program [2,3]. In section 5 we define the data structure in detail. In addition to being in standard OLYMPUS form, the program is fully indexed and commented so that its working can be readily understood from the listing. Section 6 explains the Test Runs.

Appendix 3 indicates how the COMPOSITOR has been built into the PRIME computer system at Culham by making use of the Command Abbreviation facility [11], while paper II defines the details of the OLYMPUS conventions that it fulfills.

2. INTERACTIVE PROGRAM DEVELOPMENT

There appear to be two general requirements for an interactive program development system (Fig.1):

(a) The input file A should be as simple to type as possible.

(b) The program listing B should be as well-organized and readable as possible.

In particular, the programmer should not need to concern himself with the details of layout and numbering conventions while he is constructing file A, but when he is reading the listing B it is convenient to have it laid out in a clear standardized format with a consistent numbering scheme and the main sections well emphasized.

In most present-day programming environments the formats of A and B are essentially the same, except, possibly, for tab facilities and the automatic indentation of blocks in languages such as Algol and Fortran 77. Although context editors do enable the programmer to manipulate the content of the input file A in a very general and powerful way, he remains entirely responsible for the specific details of its layout. The consequence is that a single compromise format is normally used for both A and B which is neither particularly simple to type nor particularly readable, and which has a quality that is much lower than what could be achieved with the available symbols and paper area or is normally expected of typed or printed reports.

Furthermore, even when the experienced programmer does make the considerable effort required to type his program in readable format, as many people nowadays do, there is no mechanism by which this can be standardized and programs written in collaboration by several people or even by the same person at different times can often become quite ragged.

Fortunately the computer itself can help to solve this problem by converting the free format of the input file A into the fixed standardized format of the program file B as an automatic process.

In the OLYMPUS system File A of Fig.1 is a free-format input file which is designed to be easy to type, a short example being shown in the Test Run 1 Input of Fig.4. In particular statements and comments need not be indented, statement numbers are arbitrary, and decimalized sections and subsections need not be numbered but are denoted by # or ## respectively. When File A is processed by the COMPOSITOR by typing the PRIME abbreviation command

COMPOS <filename> (1)

it is replaced in situ by the standard OLYMPUS File B format of Fig.5. (For safety the original File A is preserved in a work file).

Usually File B will require further editing or 'proof-correction' as with a typed or printed paper. This is indicated by the loop in Fig.1. The necessary corrections or additions can be made either in free-format, or according to the OLYMPUS conventions, or by a combination of both, an example being shown in the Test Run 2 Input of Fig.6. Repetition of the command (1) then updates the file to the standardized format of Fig.7 in which the section, subsection and statement numbers have where necessary been altered to take account of additions. This process may be repeated as many times as required, so maintaining a consistent standardized format throughout the development of the program, and during any subsequent maintenance that may be needed.

3. USING THE COMPOSITOR TO CONSTRUCT OLYMPUS PROGRAMS

The COMPOSITOR is designed to assist in the construction of the FORTRAN Section 3 of the OLYMPUS Standard Program File whose overall structure is indicated in II, Table 1. The construction of the DOCUMENTATION Section 1 and of the COMMON Section 2 of the SPF is facilitated by the GENSIS Generator described in IV.

3.1. Input lines

The COMPOSITOR is primarily intended to process input files containing two types of line:

OLYMPUS statements and comments

COMPOSITOR statements and comments

but it can also deal to some extent with arbitrary Fortran statements and comments.

3.2 OLYMPUS input

The standardized formats of OLYMPUS statements and comments are explained in II. These are recognized by the COMPOSITOR and minor deviations from the conventions will be tidied up to some extent, for example incorrect columns in statements, comments, headings and sub-headings. However, any statement or part of a statement that starts after col.10 will retain its position in case it is part of a planned layout.

3.3 COMPOSITOR input

The rules for COMPOSITOR statements and comments are listed in Tables 1 and 2 and only a brief explanation will be given here. COMPOSITOR input is intended to be free-format, but it has to start before col.6, in order to distinguish it from a normal Fortran continuation line (col.6) or statement (col.7). Usually it is convenient to start in col.1. Note that the combinations ## and '' should not contain embedded blanks.

Only COMPOSITOR statements beginning with C in col.1 present any problem. It is necessary to distinguish these from Fortran comments, and the rules listed in Table 2 which must to some extent be a matter of convenience, are intended for this purpose. A string beginning with C will always be recognised as a statement if col.1 is left blank and the user may find it convenient to adopt this convention.

Forms such as Cb̸=, CLb̸= are treated as statements. (b̸ = blank, C in col.1), but the occurrence of two or more blanks would lead to the strings being interpreted as OLYMPUS input lines according to Rules 16 and 17 .

If the user wishes to preserve the content of a comment line essentially unchanged, he should edit . (dot) into col.2 (Rule 18). To allow for highlighting techniques that are often used such as strings of asterisks across the page, or comments marked by C*****, the occurrence of two identical non-alphanumeric characters β (other than -) in cols. 2 & 3 of a comment line will force conversion to a C. comment (Rule 20). A line beginning C-- or *-- is however converted to a ruled line with - extending to col.70 (Rule 19), in order to minimize typing in the production of OLYMPUS programs. By these methods it should be possible to preserve most non-OLYMPUS documentation if required, although it is often preferable to convert some of it into standard OLYMPUS form. Note however that comments such as

```
    CABC,    C*ABC,   C*b̸ABC,
```

would be interpreted as statements and should be converted beforehand into, say

```
    C.ABC  or  C.b̸ABC
```

Indexes

Rules 12-14 allow for the construction of indexes with the formats illustrated in Figs.5 and 7. Rule 13 is intended for indexes of internal variables and Rule 14 for indexes of formal parameters, where <type> might be I(input), O(output) or IO(input and output), but these rules may have other applications.

EXPERT and MESAGE

Rule 8 enables an EXPERT call [2,3] of the form

```
          CALL EXPERT (ICLASS. ISUB, <n>)
```

to be generated without the programmer having to provide an argument list, where the values <n> are numbered in sequence through the subprogram, while Rule 9 simplifies the construction of a message call

```
          10                                            72
          ↓                                             ↓
          CALLMESAGE(48H < message >                    )
```

The blank after the word CALL in Rules 8 and 9 is not significant.

INSERT

Rule 11 allows a list of files to be inserted to be typed on one line, and separate C/ INSERT <name> statements will then be generated.

3.4 Output Format

The COMPOSITOR will line up declarations, statements, headings, subheadings and comments according to the standard OLYMPUS column conventions (II), and will automatically generate (or update) section and sub-section numbers. Furthermore all statement labels are updated so that they are correlated with the number of the section or sub-section in which they appear; statements that refer to the updated labels are of course, changed automatically. Because statement labels are automatically allocated, any arbitrary labels may be used in the input file provided that they are distinct.

3.5 Heading and non-executable parts

The layout conventions for the heading part of an OLYMPUS subprogram are defined in II §3.1 and the COMPOSITOR will attempt to obey these,i.e.,by inserting C-blank lines and by arranging the correct columns for the subprogram identification label and title line, and for the VERSION line. A SUBPROGRAM

FUNCTION or PROGRAM statement is lined up on col.10, while subsequent declarations are recognized individually and lined up on col.8. Minor problems may arise for declarations that are not part of standard Fortran and are therefore not recognized by the COMPOSITOR. These can usually be corrected straightforwardly using an editor.

3.6 Structure of the input file

The FORTRAN Section 3 of the SPF consists of a sequence of modules each headed by

C/ MODULE <name>

and terminated by the Fortran

END

statement. However the COMPOSITOR does not require a MODULE statement at the head of each module and works like a compiler, using END as a delimiter and processing a sequence of independent modules one-by-one.

3.7 Diagnostics and restrictions

Only limited diagnostics are provided, which on the PRIME are reported at the terminal. The COMPOSITOR will type the offending statement, and at the same time, will flag the corresponding card image in the output file with a string of asterisks in cols.1-5.

Note that the COMPOSITOR does not perform a comprehensive check of Fortran syntax. Only those syntax errors that inhibit processing will be reported.

Dialect statements that reference statement labels may cause trouble, since if the labels are not recognized they will not be updated and must then be corrected with an editor.

OLYMPUS conventions restrict the numbers of sections (9), sub-sections within a section (9) and statement labels within a sub-section (10).

4. STRUCTURE OF THE COMPOSITOR PROGRAM

The COMPOSITOR package has the SPF (Standard Program File) form shown in II, Table 1, and the program runs under the control of subprogram <0.3> COTROL of the OLYMPUS control and utility package [2,3], which also provides the COMMON blocks [Cl.1] COMBAS and [Cl.9] COMDDP and a number of utility subroutines.

The program has the OLYMPUS structure and consists of 3 main parts:

1.	Prologue	Initialize data structure
2.	Calculation	Read and analyse File A
3.	Output	Construct and output File B

Each 'step' of calculation followed by output corresponds to the processing of one module.

The program deck contains detailed indexes and other documentation together with the MINDEX master index module to enable any necessary modifications to the COMPOSITOR to be made using the GENSIS generator as explained in IV. However, many extensions and adaptations can be made by using the OLYMPUS EXPERT facility described in Appendix 2, and this method is to be preferred since it allows the SPF to remain unaltered.

Utility routines are listed in Table 3. Those in Class Z are part of the SPF and are system-independent. Routines that are system-dependent or may need to be modified are contained in a SUPPORT package that is placed after the end of the SPF.

Note that in order to conform to the rules of ANSI Fortran 66 [9] a dummy main program is provided whose purpose is to reference all the COMMON blocks and so ensure that all COMMON variables and arrays remain defined throughout the run. This replaces the main program of the OLYMPUS package [3], and calls a new OLYMPUS subprogram <0.0> MASTER which for convenience is also provided in the SUPPORT. If extra COMMON blocks are added they should be included in the main program.

The SUPPORT package also contains a version of <0.2> MODIFY which sets the appropriate channel numbers for the PRIME, <0.4> EXPERT which deals with the PRIME $INSERT substitution facility, and three dummies which suppress OLYMPUS messages. Subroutine AREAD uses a dialect Fortran statement which may need to be replaced for use on other types of computer (although it is now part of ANSI Fortran 77).

Subroutines PACK and UNPACK are written in assembly language and versions are provided for the PRIME and IBM computers. PACK is used to pack a sequence of small integers, contained in an array, into successive bytes starting from a given base address, while UNPACK is used to expand a string of bytes into an array of small integers.

Table 3 also lists four OLYMPUS routines that are specifically used by the COMPOSITOR but are not part of the present package.

4.1 Prologue

Only <1.3> PRESET need be mentioned; this sets up character codes, keyword strings and markers, and resets counters. It is called again together with <1.2> CLEAR after each module has been output.

4.2 Calculation

This is controlled by <2.1> STEPON which calls the main lower-level routines shown in Table 4 and Fig.2. Utility routines listed in Table 3 have been omitted.

STEPON reads the input file A line by line until a read error, end-of-file or an END statement has been detected. Subprogram <2.2> OLYMPS is then used to check for an OLYMPUS statement or comment, and <2.3> COMPOS for a COMPOSITOR statement or comment. A type marker NTYPE is then set and determines the processing of the line using the subroutines <2.4> to <2.11> , the contents being packed into the array NSTORE. Sections and sub-section numbers are established and statement numbers calculated and stored.

4.3 Output

This is controlled by <3.1> OUTPUT which controls the main lower-level routines shown in Table 5 and Fig.3. In fact most of the work is done by <3.2> OUTBUF which first calls <3.14> HDPART to deal with the heading part and the processes and outputs the stored lines one by one according to type.

5. DATA STRUCTURE

In addition to the standard OLYMPUS COMMON blocks [C1.1] COMBAS and [C1.9] COMDDP, there are labelled COMMON blocks listed in Table 6. These are contained in Section 2 of the SPF, and are referenced in Section 3 by C/ INSERT statements. In order to compile the COMPOSITOR the necessary insertions must be made either automatically or by hand.

Blocks [C6.1]COMCHR and [C6.3]COMSPC contain respectively the alpha-numeric characters Ø-9, A-Z and a number of other special characters that the COMPOSITOR needs to recognize. These use the internal character code of the computer, i.e. normally ASCII or EBCDIC, although other codes may also be encountered. Strings of characters are read, stored and written in packed format, (i.e. A4 on the PRIME or IBM computers), but are unpacked by the utility routine UNPACK and manipulated during the processing stage in right-adjusted form, one character/word. Normally they will be integers in the range 0-255. The integer values are set in §1 of PRESET using UNPACK.

It is assumed that the integers MO-M9 representing the digits are monotonically increasing with MO<M9, and that the integers MA-MZ representing the upper-case letters are monotonically increasing with MA<MZ, and that in neither case are other relevant characters intermixed with them. This is true for ASCII, but in EBCDIC there are gaps between MI and MJ, and between MR and MS, which in some implementations might be used for non-standard characters.

If any problems arise it is suggested that the processing should be carried out in a private internal code. This is done for the GENERATOR and the code can be set up by a method to be explained in IV.

Other characters that are not individually represented in the COMMON blocks can be read, stored and output by the COMPOSITOR if they can be handled by the computer system, but they are not specifically recognized. Lower-case letters are an example.

The number of characters/word is NCHWD, set in §3 of <1.3> PRESET, and if this differs from 4 it should be overwritten in EXPERT as explained in Appendix 2 and appropriate versions provided for the utility routines PACK, UNPACK, AREAD and AWRITE.

Block [C6.5]COMKEY contains arrays of keyword strings in unpacked integer form, one character/word, with any blanks omitted. They are set in §1 of PRESET using UNPACK, and the list can readily be extended as explained in Appendix 2 if extra keywords are needed.

Block [C4.4]COMMAX contains table sizes and other maximum values. They are set in §3 of PRESET. Array sizes can be changed by altering them here and also in module MINDEX, and re-running GENSIS (IV) to update the COMMON blocks. The numbers of sections (9) and sub-sections (9) are however a consequence of the OLYMPUS notation and increasing these would need some other changes to the program in order to accommodate 4-digit or 5-digit statement labels.

[C4.1]COMARK and [C4.2]COMSEC contain markers, current values and pointers, together with the arrays LABIN and LABOUT which hold the statement labels of files A and B respectively. [C5.1]COMBUF contains the input-output buffers and working store, and the array NSTORE which holds the packed contents of the subprogram after it has been processed and before it is output. There is one string for each line, which is stored in the form.

<length><type><contents><length>

6. TEST RUNS

Two test runs are contained in Section 4 of the SPF, with the input and output reproduced in Figs.4-7 respectively. Fig.4 shows the initial input of File A, which is processed by the COMPOSITOR to become File B of Fig.5. This is then further edited by the programmer (Fig.6), and reprocessed to give the final output of Fig.7.

Appendix 1. Input-output and character-handling

Character-handling has always been somewhat awkward in ANSI Fortran 66 [9]. The method adopted here is believed to be consistent with the formal rules of that language and has worked well in practice for many years on a range of different computer systems. However, it is not consistent with the standard version of the new ANSI Fortran 77 [10], since character-handling is one area in which Fortran 66 is not a true language subset (ref.[10] Appendix A). and the Hollerith data type has been deleted from the new standard although it is retained as an optional extension (ref.[10] Appendix C).

Minor problems may therefore occur with compilers that have been updated from Fortran 66 to Fortran 77, and if these do arise it may be advisable for users to convert the Hollerith strings and integer character variables in both OLYMPUS and the COMPOSITOR to the new Fortran 77 CHARACTER type. This should be a straightforward task, but it is hoped to issue Fortran 77 versions of these and other OLYMPUS packages in due course.

Characters are handled by the present Fortran 66 version of the COMPOSITOR in two integer formats, either packed with NCHWD characters/word, or unpacked with one character/word, right-adjusted. NCHWD is set to 4 in §3 of <1.3> PRESET, (corresponding to a 32-bit word of 4 8-bit bytes), but this default value can be overwritten by a subsequent call to <0.4> EXPERT as explained in Appendix 2.

Two subroutines PACK and UNPACK convert between these formats. Subroutine <A3> PACK(KSTR,KI,KCHARS,KN) packs KN characters from array KCHARS into array KSTR, starting from the byte location KI relative to the base address of KSTR, where the lowest byte location in the array KSTR is counted as 0.

Conversely, subroutine <A4> UNPACK (KSTR,KI,KCHARS,KN) extracts a string of KN packed characters, starting in byte position KI of array KSTR, and plants them as right-adjusted integers in array KCHARS.

According to the rules of ANSI Fortran 66 [9] it is allowable for the actual arguments corresponding to the dummy array names KSTR and KCHARS to be either array names (e.g. IARRAY) or array element names (.e.g. IARRAY (3)). What is passed to the called subroutine is the storage address of the actual argument, so that references to IARRAY or to IARRAY (1) give the same result. This arrangement gives considerable power and flexibility to the use of PACK and UNPACK, but some care may be needed in translating the COMPOSITOR program to languages with a more sophisticated syntax than Fortran 66.

Usually, it is necessary (and advisable for reasons of computing speed) to implement PACK and UNPACK in assembler language, although some dialects of Fortran provide appropriate bit-manipulation functions such as SHIFT, AND, OR from which they can be constructed. Assembler-language versions are provided in the SUPPORT package for the PRIME and IBM computers.

A typical use for UNPACK is to set up a keyword, as for example in §1 of <1.3> PRESET:

```
      CALL UNPACK (11HEQUIVALENCE, 0, MEQUIV,11)
```

which plants the 11 integers that represent the characters of the word EQUIVALENCE into words MEQUIV(1) to MEQUIV(11). The integer representation of the characters may depend on the particular computer but for the COMPOSITOR program this should not matter since no alphanumeric sorting is performed.

72-column lines are read and written by two low-level Fortran routines <A1> AREAD and <A2> APUNCH. The versions supplied use A4 format and can be used for both PRIME and IBM computers. They may need to be altered for machines with different byte or word lengths. The READ statement in AREAD is dialect Fortran 66 since it makes use of error and end-of-file conditions.

Appendix 2. Modifications using the EXPERT facility

One of the significant features of OLYMPUS programs [2,3] is that calls can be made to an auxiliary subroutine EXPERT in the form

```
CALL EXPERT (ICLASS, ISUB, IPOINT)
```

where <ICLASS.ISUB> is the decimal number of the calling routine from which the call is made.

The parameters of the call are coded as a jump within EXPERT to a point at which additional Fortran statements can be introduced. The use of INSERT statements provides access to the COMMON storage of the program, and additional COMMON variables and arrays may also be defined within EXPERT if required, (in which case they should also be added to the dummy main program of the support package).

In the published PRIME version this facility is used to recognize and convert $INSERT to C/ INSERT at the input stage, and to convert C/ INSERT to $INSERT at the output stage, so enabling the PRIME file insertion facility to be used. Coding is also included to handle the *CALL and other directive statements used on CDC and CRAY computers.

The additional Fortran statements included in EXPERT can either overwrite or supplement those of the program proper. For example if the number of characters/word is changed, say, from 4 to 8, the statement

```
NCHWD = 8
```

should be brought in by an EXPERT call in PRESET. The present version of EXPERT indicates where this should be placed.

The advantage of this method is that it allows for very flexible adaptations to an existing program without changing the main body of the code which can be preserved as a fixed binary load module.

Appendix 3. Using the COMPOSITOR on the Culham PRIME

A simple way to use the COMPOSITOR is as follows. Suppose that the programmer is working in a sub-directory [12] that contains File A under some name <fname>. He can then process this by typing the command

```
COMPOS <fname>
```

The processed version File B will overwrite the original, which will be preserved in file

```
*>TEMP>COMPIN
```

The new <fname> can be edited and the command repeated as many times as required. Diagnostic messages will be displayed at the terminal.

To make use of this facility the programmer should set up the PRIME abbreviation entry [11]

```
COMPOS EXEC KBCCPG>COMPOS
```

under his username. He should also create a lower-level sub-directory TEMP in the area of file-space in which he is working.

The Culham PRIME version of the COMPOSITOR accepts *INSERT, $INSERT and C/ INSERT statements, converting all of these to $INSERT for direct use with the PRIME Fortran compiler.

TABLE 1.

Rules for COMPOSITOR Statements and Comments

1.	COMPOSITOR input must start in cols. 1-5.
2.	No blanks are required before labelled or unlabelled statements.
3.	Comments start with *
4.	Continuation lines start with &
5.	Headings start with #
6.	Subheadings start with ##
7.	A numerical label with no statement generates CONTINUE.
8.	CALL EXPERT requires no parameter list.
9.	CALL MESAGE(<message>) is accepted, (<message> ≤ 48 chars.)
10.	*MODULE <name> is converted to C/ MODULE <name>.
11.	*INSERT <name1>,<name2>, ... generates a C/ INSERT sequence.
12.	A line ''INDEX OF <purpose> produces a C. heading line.
13.	A line ''<variable> ƀ <purpose> produces a C. index entry.
14.	A line ' <variable> ƀ <type> ƀ <purpose> produces a C. index entry.

TABLE 2.

Rules for Lines with C in Column 1

These are treated as Fortran statements except as follows:

15. A line beginning C/ is treated as an OLYMPUS control statement.

16. A line beginning Cƀα is treated as an OLYMPUS comment,where α is blank or any character except =

17. A line beginning CLƀα is treated as an OLYMPUS heading or subheading, where α is blank or any character except =

18 A line beginning C. is reproduced unchanged.

19. A line beginning C-- or *-- generates a ruled line.

20. Any other line Cββ<string> converts to C.β<string>, where β is any non-alphanumeric character.

NOTE ƀ denotes a blank

Table 3. Subsidiary Routines

	COMPOS Program		
Z1	MATCH (7)	Compare character strings	L
Z2	STORE (3)	Store character string	
Z3	CREAD (4)	Fetch card image	
Z4	CWRITE (2)	Output card image	
Z5	CONVRT (3)	Integer - string conversions	
Z6	LABREF (1)	Check for label reference	L
Z7	GETREF (5)	Fetch label reference	
Z8	DECLAR (5)	Check for Fortran declaration	
Z9	JOURNL	Write offending statement to terminal	
	SUPPORT Package		
	(main)	Defines all COMMON blocks	
0.0	MASTER	OLYMPUS master subprogram	
0.2	MODIFY	Modify basic data if required	P
0.4	EXPERT (3)	Modify standard operation of program	P
1.4	DATA	Dummy to suppress OLYMPUS message	
1.5	AUXVAL	Dummy to suppress OLYMPUS message	
1.6	INITAL	Dummy to suppress OLYMPUS message	
1.8	START	Dummy to suppress OLYMPUS message	
4.1	TESEND	Dummy to suppress OLYMPUS message	
A1	AREAD (4)	Read card image	SIP
A2	APUNCH(2)	Output card image	SIP
A31	PACK (4)	Pack character string	AI
A3P	PACK (4)	Pack character string	AP
A41	UNPACK(4)	Unpack character string	AI
A4P	UNPACK(4)	Unpack character string	AP
	OLYMPUS		
0.1	BASIC	Initialise basic control data	
0.3	COTROL	Control the run	
U1	MESAGE (1)	Print 48-character message on output channel	
U15	RESETI (1)	Reset integer array to specified value	

Notes

The number of arguments is in brackets
A = Assembler language
B = IBM
L = Logical function
P = PRIME
S = System dependent Fortran

Table 4. Class 2 Routines

2.1.	STEPON	Read and process File A
2.2	OLYMPS	Check for OLYMPUS input
2.3	COMPOS	Check for COMPOSITOR input
2.4	PCOMNT	Process comment
2.5	PINSRT	Process INSERT
2.6	PMODUL	Process MODULE
2.7	PHEAD	Process heading/subheading
2.8	PDECL	Process declaration
2.9	PSTAT	Process executable statement
2.10	PLABEL	Process labelled statement
2.11	PCNCRD	Process continuation line
2.12	PINDEX	Process index entry

Table 5. Class 3 Routines

3.1	OUTPUT	Construct and output File B
3.2	OUTBUF	Output buffer
3.3	OUTCOM	Output comment
3.4	OSPF	Output SPF statement
3.5	OHEAD	Output heading/subheading
3.6	OSTAT	Output executable statement
3.7	OLABEL	Output labelled statement
3.8	OUTDO	Output DO statement
3.9	OUTIF	Output IF statement
3.10	OUTGO	Output GO TO statement
3.11	OUTIO	Output I/O statement
3.12	ODECL	Output declaration
3.13	OCNCRD	Output continuation line
3.14	OINDEX	Output index entry
3.15	HDPART	Construct subprogram heading part

TABLE 6. COMMON Blocks

1. General OLYMPUS Data

C1.1	COMBAS	Basic system parameters
C1.2	COMDDP	Development and diagnostic parameters

4. Housekeeping

C4.1	COMARK	Markers
C4.2	COMSEC	Section/Sub-section markers
C4.3	COMADM	Administrative variables
C4.4	COMMAX	Maximum values

5. I/O and Diagnostics

C5.1	COMBUF	Buffers

6. Text Manipulation

C6.1	COMCHR	Character codes
C6.3	COMSPC	Special characters
C6.5	COMKEY	Character strings

REFERENCES

[1] The Publication of Scientific Fortran Programs, K.V.Roberts, Comput. Phys. Commun. 1 (1969) 1.

[2] An Introduction to the OLYMPUS System, K.V.Roberts Comput. Phys. Commun. 7 (1974) 237.

[3] OLYMPUS - A Standard Control and Utility Package for Initial-Value Fortran Programs, J.P.Christiansen and K.V.Roberts, Comput. Phys. Commun. 7 (1974) 245.

[4] OLYMPUS Conventions, M.H.Hughes and K.V.Roberts, CLM-P685

[5] Experience with the OLYMPUS System, K.V.Roberts, in the Relationship between Numerical Computation and Programming Languages, ed. J.K.Reid (North-Holland 1982).

[6] PRIME: The New Users Guide to EDITOR and RUNOFF, D.P.Vern, PRIME Computer Inc. (1978).

[7] The Regeneration of Fortran Software, K.V.Roberts, CLM-P683.

[8] The OLYMPUS FORTRAN GENERATOR, M.H.Hughes and K.V.Roberts, CLM-P689

[9] USA Standard Fortran, USA X3.9 - 1966,(USA Standards Institute, New York, March 1966).

[10] American National Standard Programming Language FORTRAN, ANSI X3.9 - 1978, American National Standards Institute Inc. New York, 1978).

[11] PRIMOS COMMANDS Reference Guide FDR3108 - 101B, Prime Computer Inc. (1980), pp. 3-6 to 3-10.

[12] loc. cit. ref. [11], pp. 2-6 to 2-9.

[13] OLYMPUS and Preprocessor Package for an IBM 370/165, M.H.Hughes, K.V.Roberts and P.D.Roberts, Comput. Phys. Commun. 9 (1975) 51.

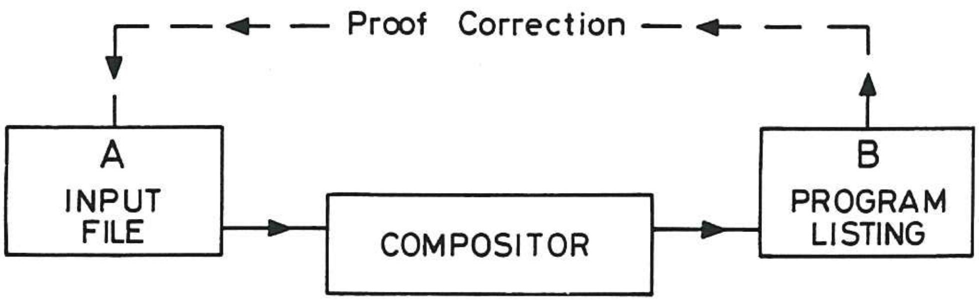

Fig.1 Iterative use of the COMPOSITOR. File A is designed to be easy to type at the terminal, while File B is designed to be easy to read, but may be proof-corrected and re-processed any number of times.

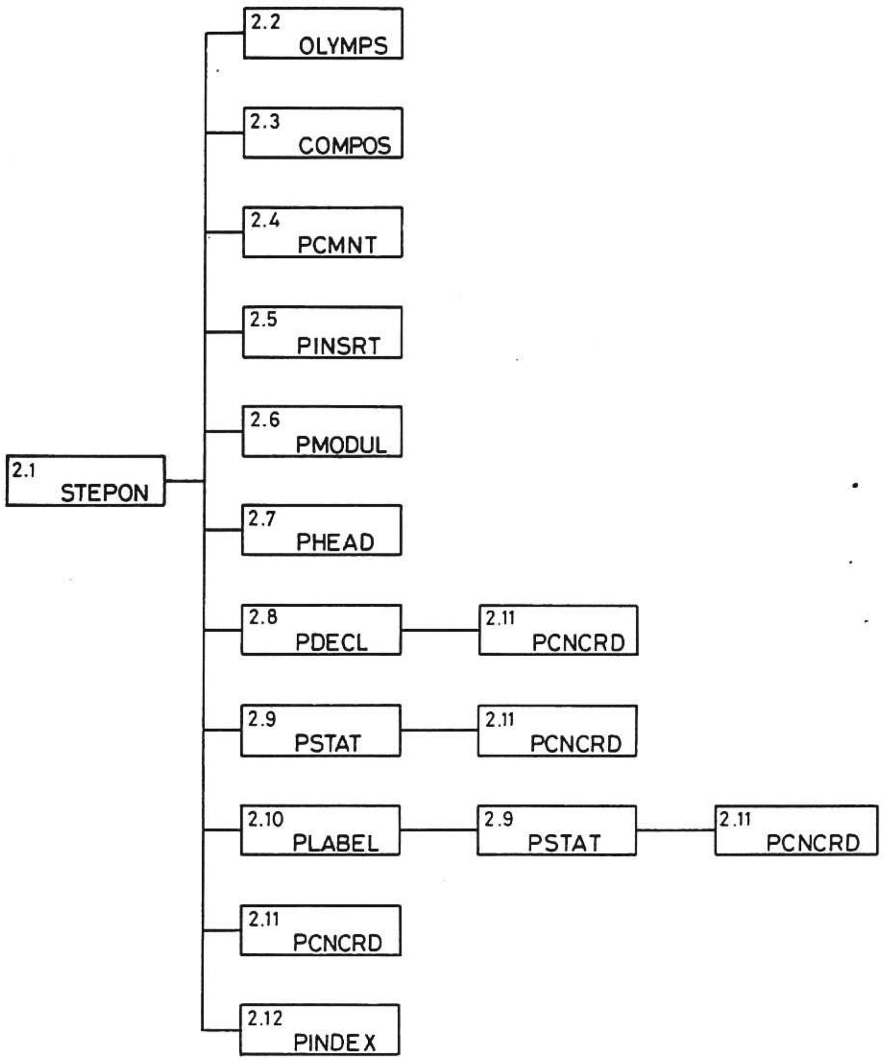

Fig.2 Block diagram of the Class 2 input and analysis part.

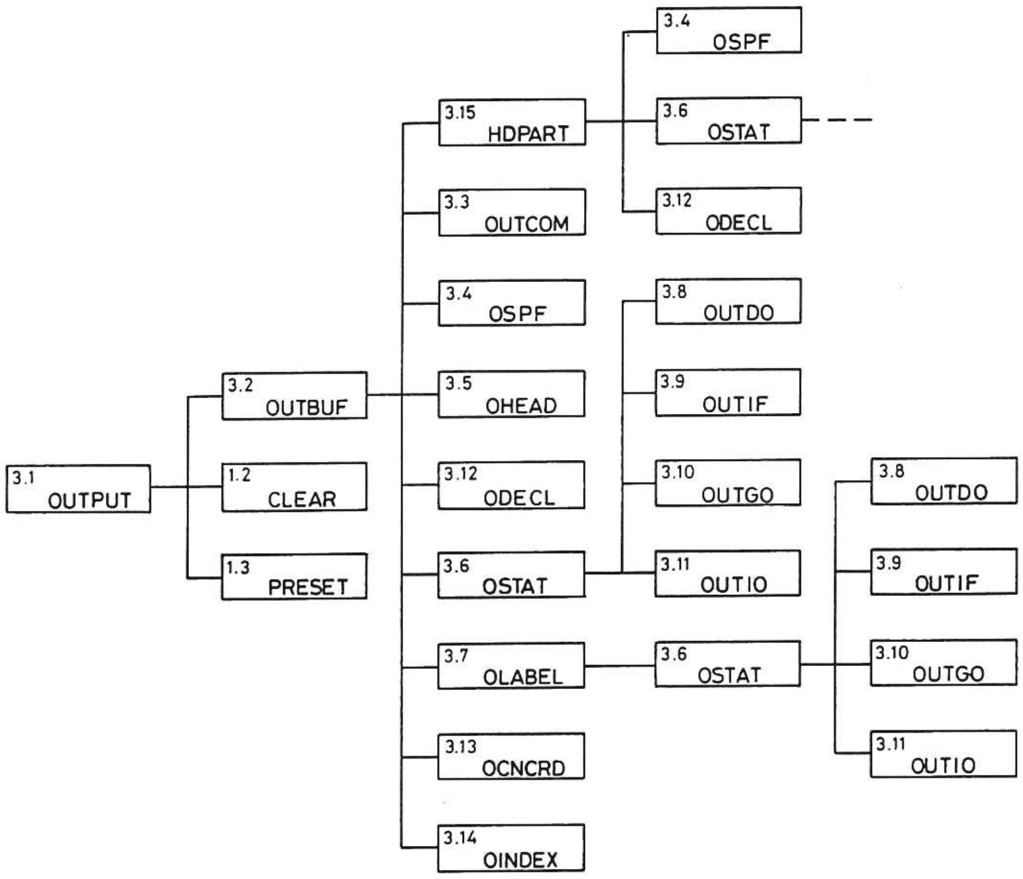

Fig.3 Block diagram of the Class 3 construction and output part.

```
SUBROUTINE CONVRT(KCHR,KNUM,K)
*Z.10 Convert unpacked characters to integer and vice-versa
DIMENSION KCHR(1)
*--
*INSERT COMCHR,COMSPC
*--
DIMENSION INUM(1)
EQUIVALENCE (INUM(1),M0)
C
GO TO (1,2),K
*--
£Character string to integer
1
*Clear integer to zero
KNUM=0
*Construct 6-digit integer
ITEN=1000000
*Scan over digits, highest first
DO 3 J=1,6
ITEN=ITEN/10
ICHR=KCHR(J)
*Check against characters M0 to M9
DO 4 J1=1,10
IJ=J1
4 IF(ICHR.EQ.INUM(J1)) GO TO 5
*Character not found - make it zero
IJ=1
5 IJ=IJ-1
*Accumulate integer
KNUM=KNUM+ITEN*IJ
3
C
RETURN
2
END
```

Fig.4 COMPOSITOR input. (# is reproduced here as £).

```
C
         SUBROUTINE CONVRT(KCHR,KNUM,K)
C
C Z.10 Convert unpacked characters to integer and vice-versa
C
       DIMENSION   KCHR(1)
C-----------------------------------------------------------------------
C/ INSERT COMCHR
C/ INSERT COMSPC
C-----------------------------------------------------------------------
       DIMENSION   INUM(1)
       EQUIVALENCE (INUM(1),MO)
C
         GO TO (100,104),K
C
C-----------------------------------------------------------------------
CL              1.         Character string to integer
C
  100    CONTINUE
C     Clear integer to zero
         KNUM=0
C     Construct 6-digit integer
         ITEN=1000000
C     Scan over digits, highest first
         DO 103 J=1,6
         ITEN=ITEN/10
         ICHR=KCHR(J)
C     Check against characters M0 to M9
         DO 101 J1=1,10
         IJ=J1
  101    IF(ICHR.EQ.INUM(J1)) GO TO 102
C     Character not found - make it zero
         IJ=1
  102    IJ=IJ-1
C     Accumulate integer
         KNUM=KNUM+ITEN*IJ
  103    CONTINUE
C
         RETURN
  104    CONTINUE
         END
```

Fig.5 Processed Version of Fig.4.

```
C/ MODULE Z10
C
          SUBROUTINE CONVRT(KCHR,KNUM,K)
C
C Z.10 Convert unpacked characters to integer and vice-versa
C
'KCHR IO Character string
'KNUM IO Integer
'K I Choice of call
•
        DIMENSION KCHR(1)
C-------------------------------------------------------------------
C/ INSERT COMCHR
C/ INSERT COMSPC
C-------------------------------------------------------------------
        DIMENSION   INUM(1)
        EQUIVALENCE (INUM(1),M0)
C
          GO TO (100,104),K
C
C-------------------------------------------------------------------
CL               1.          Character string to integer
C
  100     CONTINUE
C      Clear integer to zero
          KNUM=0
C      Construct 6-digit integer
          ITEN=1000000
C      Scan over digits, highest first
          DO 103 J=1,6
          ITEN=ITEN/10
          ICHR=KCHR(J)
C      Check against characters M0 to M9
          DO 101 J1=1,10
          IJ=J1
  101     IF(ICHR.EQ.INUM(J1)) GO TO 102
C      Character not found - make it zero
          IJ=1
  102     IJ=IJ-1
C      Accumulate integer
          KNUM=KNUM+ITEN*IJ
  103     CONTINUE
C
          RETURN
£Integer to character string
  104     CONTINUE
•Clear character string to blank
CALL RESETH(KCHR,6,MBLANK)
•Construct integers
I1=KNUM
•Scan over digits, lowest first
DO 11 J=1,6
I=7-J
I2=I1/10
•Current right-hand digit
INT=I1-10*I2
KCHR(J)=INUM(INT+1)
•Remove right-hand digit
I1=I2
IF(I1.EQ.0) RETURN
11
•
RETURN   •
C
•--
''INDEX OF LOCAL VARIABLES
•
''I Character counter
''I1 Remaining part of integer
''I2 Right-most digit removed
''ICHR Current character
''IJ Current digit
''INT Digit 0 to 9
''INUM Array of characters M0 to M9
''ITEN Current power of 10
''J Counts digits or characters 1 to 6
''J1 Scans digits 0 to 9
C
          END
```

Fig.6 Proof-corrected version of Fig.5, with additional statements and illustrating the construction of indexes.

```
C/ MODULE Z10
C
      SUBROUTINE CONVRT(KCHR,KNUM,K)
C
C Z.10 Convert unpacked characters to integer and vice-versa
C
C. KCHR           Character string                           IO
C. KNUM           Integer                                    IO
C. K              Choice of call                             I
C
      DIMENSION   KCHR(1)
C-------------------------------------------------------------------
C/ INSERT COMCHR
C/ INSERT COMSPC
C-------------------------------------------------------------------
      DIMENSION   INUM(1)
      EQUIVALENCE (INUM(1),M0)
C
      GO TO (100,200),K
C
C-------------------------------------------------------------------
CL             1.         Character string to integer
C
  100 CONTINUE
C    Clear integer to zero
      KNUM=0
C    Construct 6-digit integer
      ITEN=1000000
C    Scan over digits, highest first
      DO 103 J=1,6
      ITEN=ITEN/10
      ICHR=KCHR(J)
C    Check against characters M0 to M9
      DO 101 J1=1,10
      IJ=J1
  101 IF(ICHR.EQ.INUM(J1)) GO TO 102
C    Character not found - make it zero
      IJ=1
  102 IJ=IJ-1
C    Accumulate integer
      KNUM=KNUM+ITEN*IJ
  103 CONTINUE
C
      RETURN
C
C-------------------------------------------------------------------
CL             2.         Integer to character string
C
  200 CONTINUE
C    Clear character string to blank
      CALL RESETH(KCHR,6,MBLANK)
C    Construct integers
      I1=KNUM
C    Scan over digits, lowest first
      DO 201 J=1,6
      I=7-J
      I2=I1/10
C    Current right-hand digit
      INT=I1-10*I2
      KCHR(J)=INUM(INT+1)
C    Remove right-hand digit
      I1=I2
      IF(I1.EQ.0) RETURN
  201 CONTINUE
C
      RETURN
C
C-------------------------------------------------------------------
C.                       INDEX OF LOCAL VARIABLES
C
C. I              Character counter
C. I1             Remaining part of integer
C. I2             Right-most digit removed
C. ICHR           Current character
C. IJ             Current digit
C. INT            Digit 0 to 9
C. INUM           Array of characters M0 to M9
C. ITEN           Current power of 10
C. J              Counts digits or characters 1 to 6
C. J1             Scans digits 0 to 9
C
      END
```

Fig.7 Final processed version of the subprogram, with indexes of formal parameters and local variables.