19.4 Converting Codes to Different Versions

A. Purpose

This program, `m77con`, converts codes from one version to another. It is used to convert between different precisions, and to support the special comments used in some of the Fortran codes for input to a program that converts programs from Fortran to C. Support is provided for precision greater than double for systems that support such a feature. Support is provided for conversion between real programs and integer or complex versions of a program, and also partial double precision and partial quadruple precision. Most MATH77 codes contain the comments only for conversion between the single and double precision cases. Although designed for use with MATH77 codes, `m77con` could be used in other applications that want a portable means to support multiple versions. Any version of a code can be used as input to `m77con` for the purpose of getting another version.

B. Usage

One uses this program by creating a control file named `m77job`, and executing the program `m77con` from the command line. The first part of this section describes the control file which defines what version is desired. The second part describes the comments in the code which indicate how different versions are to be obtained.

B.1 Transforming an Instrumented Code

In the descriptions below, items in brackets are optional; L_i is used for letters (frequently the first letter in the name of a program, which indicates the precision); and T_i is used for letters associated with types. Types supported include: S, D, Q, I, C, Z, and W, which are used for Single precision, Double precision, Quadruple precision (or at least some precision greater than double), Integer, Complex, double precision complex, and quadruple precision complex respectively.

"C..." is a comment and can appear anywhere in the control file. A comment of the form "C>>yyyy−mm−dd..." is treated as a special case. It inserts a date stamp in the file being processed, which is used to track changes in MATH77 codes. Details on this can be found in the code listing for `m77con`.

"MAKE L_{out} [with type T_{out}] [from L_{in} [with type T_{in}]]" is used to define a letter in the names of the output and input files and types associated with these files. L_{out} is a letter used in output file names. T_{out} is the type for the output file. If T_{out} is not given it is assumed to be L_{out} if L_{out} is an allowed type, is assumed to be S or D, for L_{out} = P or X respectively, and otherwise an error results. (P is assumed to be used for partial double precision, and X for partial quadruple precision.) L_{in} and T_{in} are defined as for L_{out} and T_{out}, except they apply to input files. If the "from" clause is missing, L_{in} is D, except when L_{out} is D, in which case L_{in} is S. The case of letters is not significant, except for L_{out}. The case used for L_{out} determines the case used when substituting L_{out} or L_{in} into a file name, and thus the case is significant on systems where case is significant in a file name.

"T Defined by: L_{exp}, *keyword*, *type_conv*, *mach_const*"
"T Defined by: *keyword*, *type_conv*, *conj*, *imag_part*"
One of these lines is needed only if the input or output code makes use of a nonstandard type, namely, Q, Z, or W. Note that W makes use of Q, and thus if it is used, Q must also be defined. The first type of line is used to define real (as opposed to complex) types. L_{exp} is the letter used to define the exponent part of floating point constants, *keyword* is the text used to declare a variable of type T, *type_conv* is the name of the function used for type conversion to type T, *mach_const* is the name of the function used for getting machine constants, *conj* and *imag_part* are the functions used to conjugate or get the imaginary part of a complex expression, respectively. The standard S, D, and C types are initialized as if one had input:
"S Defined by: E, real, real, r1mach",
"D Defined by: D, double precision, dble, d1mach", and
"C Defined by: complex, cmplx, conjg, aimag"

Note that at most one embedded blank is allowed inside *keyword*, and if such a blank is desired it must be included on this line. Internally, 'I' defines *keyword* as "integer", and *type_conv* as "int".

"VERIFY path" if present will compare the lines of the result files with lines in the files of the same name as the result files preceded by "path", and when lines are different, both lines will be printed. The text in "path" is taken exactly as given, and is considered terminated by either a blank or an end of line. This comparison is not sensitive to changes in the special comment lines described in the following subsection, or to differences in the number of blanks.

"SET A_1, A_2, ... A_n" is used to set values for the metavariables defined by the assignments A_i. These assignments, which sometimes are necessary to get the version of a code desired, and the metavariables set by these assignments, are described in the following subsection. One can have any number of "SET..." lines.

"STOP ..." acts like an end of file, i.e. no more lines are read in the control file.

"FILE *name*[[,] *ext*]" is used to specify the name of a file to be processed. A "?" in name may be used to parameterize the name if desired. L_{in} is substituted for the "?" (if any) and a file of this name is processed as the input file. Then L_{out} is substituted for "?" to obtain the name of the output file. If the ", *ext*" is present and one of the first two characters in *ext* is "." this name is further modified by searching backward from the end of "*name*" looking for the first character in "*ext*". When this character is found, this character and the following characters in *name* are replaced by "*ext*". If the ", *ext* is present and neither of the first two characters is a ".", *ext* is used as the name of the output file. The input and output files can have the same name. If the "?" is missing, there is no need to have had a "MAKE..." line. Any characters except "," can appear in "*name*" or "*ext*", as long as the results will serve as valid file names. Examples and Remarks, below, gives samples of FILE statements.

Processing a "FILE..." line involves carrying out conversions on the specified input file and placing the result in the output file. If a "VERIFY..." line is present, the output file will be compared with the file specified there. After processing a "FILE ..." line, the next line from the control file, `m77job`, is read, and actions are taken as requested there. All metavariables defined in previous SET instructions become undefined if there is a new SET instruction; otherwise the old values are kept.

B.2 Instrumenting a Code

We use the notation:

L, L_1, ... L_i	Letters
T, T_1, ... T_i	Letters used to denote types
I, I_1, ... I_i	Integer constants
V, V_1, ... V_i	Metavariables
E, E_1, ... E_i	Expressions
A, A_1, ... A_i	Assignments

The types are: S, D, Q, I, C, Z, and W, which are used for Single precision, Double precision, Quadruple precision (or at least some precision greater than double), Integer, Complex, double precision complex, and quadruple precision complex respectively. An "L" equal to one of these letters is used for a code of the corresponding type. Other letters are used for mixtures of types. In particular, P (Partial double) is used for a code that uses some D in a code that is primarily S (perhaps to save on storage), and X (eXtended) is used for a code that is primarily in D, but uses some Q. No other precisions are supported.

There are integer, logical, and string metavariables and constants. An integer constant is defined as in Fortran. The logical constants are .T. .F. .U. .D. .S. .I. and .C. Of course .T. is always true, and .F. is false; .U. is undefined (see below); .D. .S. and .I. have a value of .T. if the upper case of the letter L_{out} in a MAKE command matches the letter between the dots, has a value of .F. if L_{out} is defined but does not match, and otherwise has the value .U.; and .C. has the value .T. if a version of the code used as input to the Fortran to C converter is being generated, and is .F. otherwise. Initially .C. =.F.; its value can be set to .T. with the statement "SET .C." in `m77job`, and can be reset to .F. with the statement "SET ∼.C.". (With T=.T., F=.F., and U=.U., we have, U|F=.U., U|T=.T., U&F=.F., U&T=.U., ∼U=.U. If U is on either side of a relational operator, the result is .U.)

A string is defined as an arbitrary sequence of characters enclosed by either apostrophes (`'`), or quotes (`"`), has the same form as a metavariable (see below), or is .X., .Y., or .N. .X. gives the string consisting of the letter used for floating point exponents in the output, .Y. gives the string consisting of the letter T_{out}, and .N. gives the string consisting of the letter L_{out}. If a logical value of a string is required by the context, a string has the value .U.

A metavariable is any number of nonoperators, not all of which are digits, and not one of the letters "TFUXN" preceded and followed by a ".". Metavariables once defined remain defined until the end of the file. Those metavariables defined in `m77job` remain defined past the end of file, until the next "SET..." line read in `m77job`. Note that metavariables appear only in "C++..." lines and in lines immediately following "C++ Substitute for ..." lines.

Expressions may use the following operators, all of which have the same meaning and precedence as their Fortran counterparts.

Arithmetic:	+	−	*	/	**	
Relational:	<	>	==	/=	<=	>=
Alt. Relational:	=	∼=				
Logical:	| (or)	& (and)	∼ (not)			
Concatenation:	//					
Grouping letters:	{	}				
Other:	(	)	,			

The "{" and "}" are used to define a special operand. If the letter L_{out} from the "MAKE..." line, see "Transforming an Instrumented Code" above, appears inside the {...}, then this operand has the value true, and otherwise has the value false. The letters inside the {...} may optionally be separated by commas. Thus {SP} is true if L_{out} is either S or P, and is false otherwise.

We recommend using "==" rather than "=" in relational expressions. "=" is allowed for compatibility with comments written for an earlier processor. The first "=" in an assignment statement is part of the assignment statement, not part of an expression.

An expression can be formed as one would expect using the operators that are available. A diagnostic is given if one tries to mix types in a way that make no sense. Integer and logical are converted to strings when they appear as an operand of "//".

As assignment has the form: $V = E$, or simply V. The latter form is equivalent to writing V=.T.

Three kinds of comments in the source code initiate special processing by the program. (Not counting the "C<<..." comments which are used for date stamps.) Those headed by "C++" are intended primarily to support different versions of a program. Those headed by "C−−" are intended primarily to support different precisions, which involves changing types in Fortran statements. Those headed by "C%%" are used to convert the code to the form needed for the Fortran to C conversion program. The actions for "C++..." lines precede other actions, and thus a "C++..." line could result in turning off a "C−−..." line. Also, if there was no "MAKE..." line, type conversion is turned off, and "C−−..." lines are simply treated like any other comments. These comments allow either case for the "C" and also allow an "*" in column one.

"C++..." lines have the form:

(1)	"C++	Code for E is active"
(2)	"C++	Code for E is inactive"
(3)	"C++	END"
(4)	"C++	Of next I lines, only the first E are active"
(5)	"C++	Default A_1, A_2, ... A_i", or
	"C++L	Default A_1, A_2, ... A_i", or
	"C++(E)	Default A_1, A_2, ... A_i"
(6)	"C++	Current has A"
(7)	"C++	Substitute for V_1, V_2, ... V_i below"
(8)	"C++	Replace A_1, A_2, ... A_i"
(9)	"C++	[With first index E_1,] Save data by elements if E_2"

A line of form (1) or (2) initiates a scope of applicability that is terminated by the next line of form (1), (2), or (3). In either case (1) or (2), if E=.U., no change is made in this line or in the lines in its scope.

In the case of (1), if E=.T. there is no change. If E=.F., the word "active" is changed to "inactive" and in the following scope, columns 1–71 are shifted to columns 2–72 and a C placed in column 1. An error message is printed if the original line was not blank in column 72. (Only the first 72 characters of input lines are read.)

In the case of (2), if E=.F. there is no change. If E=.T. the word "inactive" is replaced by "active" and in the following scope, columns 2–72 are copied to columns 1–71, and a blank placed in column 72.

No other "C++..." line may appear between a line of type (1) or (2) and the next line of type (1), (2), or (3). No code changing action can be active when the end of a Fortran program unit is reached.

Examples from AMACH of lines of type (1), (2) and (3) are,

```
c++ Code for SYS = IEEE is ACTIVE
...
c++ Code for SYS = AMDAHL is INACTIVE
C ...
c++   END
```

In case (4), one must have I (an integer constant) > 0, and E must have an integer value <= I. The following I lines in the code must be valid Fortran statements if column 1 is set to a blank and there must be I lines preceding the last line of the Fortran program unit. This causes the action of storing a ' ' in column 1 of the following E lines, and then storing a 'C' in column 1 for the remaining $I - E$ lines (neither shifts the line).

An example from DIVA of using this feature is (the first two lines given here are one line in the code):

```
c++ Of next 23 lines, only the first
                              KDIM+MAXORD are active
      data B(1)  / 5.00000...0D-1 /
      ...
      data B(22) / 1.97628...5D-3 /
C     data B(23) / 1.81159...2D-3 /
```

In case (5) when no L or (E) is present each assignment, A_i, is executed if and only if the V_i on the left of the "=" in the A_i is not already defined. (Recall that if A_i does not contain an "=" sign, it is as if it had the form "V_i = .T.".) V_i may have been defined either from a "SET..." line in `m77job` or from an earlier "C++" line. When L is present, L must be a sequence of letters and the assignments will be processed as described for the case when no L is present if and only if the current L_{out} matches one of the letters in L. When (E) is present, the line is processed as above if E is a logical expression with the value .T., else the line rest of the line is ignored.

The following example from DRDIVA illustrates defining a default value for NDIG of 4 for type of "S" or "P", and a default value for NDIG of 10 in other cases. If NDIG has been set in `m77job`, then the value set there will be used.

```
c++SP Default NDIG = 4
c++  Default NDIG = 10
```

In case (6), the A must have an "=" sign. If the V on the left side of the "=" in A is not defined this works just like a "C++ Default A" line. If V is defined, then the text on the right of the "=" is replaced by the current value of V.

The following line from AMACH shows that AMACH is currently configured for SYS = IEEE. If SYS is not set in `m77job`, this value for SYS will continue to be used. Otherwise the new value will replace the `IEEE` on this line.

```
c++ CURRENT HAS SYS = IEEE
```

In case (7), the following line is examined for each of the V_i's in turn. If such a metavariable name is found followed by an "=" sign, and an integer value, the integer is replaced by the integer value of V_i. If the "=" sign is followed by a logical value, this value is replaced by the corresponding Fortran logical constant for V_i. A type mismatch in either of these cases results in a diagnostic, with no change. A value of .U. in this case is treated as a mismatch.

The following from DRDIVA illustrates using this feature.

```
c++ Substitute for NDIG below
      parameter (NDIG = 10)
```

In case (8), the left side of each A_i must be a constant character string, i.e. either `'xx...x'` or `"xx...x"`, and the right side, E_i, must be an expression that evaluates to a string value. Occurrences (case of letters must match) of the "xx...x" on the left of A_i are replaced by the value for E_i. The replacement is made on "C++ Replace..." lines, and other lines not starting with "C++". If this line precedes a "C−−..." line of type (1), see below, then the replacement applies starting with the first line of the file else it applies starting with the current line.

The following from DRDRAN illustrates this feature.

```
c++S Default NDIG = 6
c++  Default NDIG = 12
c++  Replace "f15.12" = "f"//NDIG+3//"."//NDIG
...
      print'(7x,i7,'','',i5,10x,f15.12)',K...
```

Case (9) is intended primarily to avoid unnecessary mess in Fortran code that would have too many continuation lines for a data statement in Fortran when declared with a single array. The C converter we are using generates much better code when such data statements are declared with a single array instead of a separate statement for each element. E_1 must be a integer expression, and E_2 must be logical. If the part containing E_1 is missing the action is as if E_1 were present with a value of 1. The data statement following this must contain exactly one data item per line. When E_2 = .T., the first line and every following line that is either a continuation line, or is a data statement for the same name, is changed (if necessary) to be a data statement for a single element of the array. The index used for the array element is E_1 for the first data statement and is incremented by one for the following statements. If E_1 is .F. the first data statement line uses only the array name, and following lines that are either continuation lines, or data statements for the same name are changed to be continue statements. There is no limit on the number of continued lines.

The "C−−..." lines have the form:

(1)	"C−−L	Replaces "?": V_1, V_2, ..."
(2)	"C−−T	(Type)Replaces "?": V_1, V_2, ..."
(3)	"C−−&	v_1, v_2, ..."
(4)	"C−−T	Next line special: S_1, S_2, ..."
(5)	"C−−	Begin mask code changes"
(6)	"C−−	End mask code changes"

At most one of (1) or (2) can occur in a file, and all lines of type (3) must follow immediately after a (1) (2) or (3). (No embedded comments.) In case (1) L denotes the value of L_{out} for the current version of the code. (See the description of the "MAKE..." line in Usage above.) If $L = L_{out}$, then no names are changed. When $L = L_{in}$ the L in this statement is replaced by L_{out}. Each of the V_i, or v_i, must be Fortran names with one of the letters replaced by a "?". These names with the "?" replaced by L_{in} are searched for and replaced as described below, starting at the beginning of the file. A line of type (2) must follow immediately after a line of type (1) or (3). The action for this kind of line is similar to that for type (1), but with L, L_{out}, and L_{in}, replaced by T, T_{out}, and T_{in} respectively.

The following excerpt from DWCOMP illustrates lines of type (1) and (3).

```
c--D replaces "?": ?WCOMP,?WATAN,...
c--&   ?WSQRT,?WEXP,?WSIN,?WCOS,...
```

Case (4) is used to indicate either that something different from the standard conversion is to be used on the next line, or that conversion to or from integer or one of the complex types is allowed. Conversions to or from integer types occur only on lines so marked. For a conversion to or from a code corresponding to the first letter of one of the pairs, (C, S), (Z, D), (W, Q), the conversion is done as if the second letter applied, except for lines marked in this manner. T indicates the type to assume for the input on the following line. The T is replaced by the value used for the type of the output on the next line. The S_i is either a single L_i, or an expression of the form L_i => T_i. If L_{out} is the same as any of these L_i, then the following line is treated as special based on this S_i. If there is no "=> T_i" part then presumably L_i is I, C, Z, or W, and the following line is modifiable to suit these types. If the "=> T_i" part is present, then the following line is to be converted as if the output type were T_i. If none of the L_i match the L_{out} from the "MAKE..." line, the following line is treated just like any other line.

The following from DRDIVA illustrates specifying metavariables with a higher than usual precision when L_{out} is "P" or "X". In all other cases the declaration here would be what one would expect from the letter specifying the precision.

```
c--D Next line special: P=>D, X=>Q
      double precision TSPECS(4), Y(IYDIM)...
```

?VECPR is one of the few codes in MATH77 which allows all the real precisions, and also allows integer. The first pair of lines below get converted to the second pair when DVECPR is converted to IVECPR.

```
c--D Next line special: I
      double precision  V(N)

c--I Next line special: I
      integer           V(N)
```

Lines of type (5) and (6) mark the beginning and end, respectively, of a set of lines in which all code modifications due to "C−−" commands will be inhibited.

The "C%%" lines are documented here so that those looking at the library source will understand their purpose. These lines only have a special meaning when they are not being treated in a special way for some other reason. In particular, if a "C++ END" statement would be acceptable, then such lines are not treated in a special way. When .C. = .T., the "C" is deleted from column one in any sequence of consecutive lines containing "C%%", and a C is inserted at the start of the first line (one line only) that does not start in this way. If .C. = .F., and the lines are not being treated in a special way for some other reason, then consecutive lines starting with "%%" have a C inserted in column 1, and the next line has the character in column 1 deleted. These lines provide a way of indicating changes required to get the code converted to C, without using the "C++" mechanism, which makes the code difficult to read when used frequently.

C. Examples and Remarks

If one wanted to get the partial double precision version of the integration program DIVA, Chapter 14.1, one could set `m77job` to the following for a unix machine. (For a PC replace the `.f` with `.for`.)

```
MAKE p
FILE ?iva.f
FILE ?ivag.f
FILE ?ivadb.f
```

To convert `amach` for a VAX running UNIX, `m77job` would contain the following to obtain the output file `amach.f` from the input file `amach`.

```
SET SYS = VAX
FILE amach.f
```

D. Functional Description

The program `m77con` converts between different versions of Fortran 77 source code that satisfy conventions used for MATH77. It combines functionality from earlier programs: "The Specializer" [1], "MARVEL" [2], and "CHGTYP" [3].

The code is processed a line at a time, first doing actions on "C++..." lines, then for other lines doing the actions called for by "C++..." lines (except for replacing strings), then doing actions on "C−−..." lines, and then doing other replacements. Processing restarts with the first line after initial processing of "C−−" lines of types 1–3.

Except for strings defined by "C++ Replace..." lines, all comparisons done by this code are case insensitive, all replacements are done in the same case as for the text found, and blanks are removed before comparisons are made in the source code.

Let "in" denote a string, keyword, variable name, or function name we are searching for, and "out" denote the corresponding replacement. When conversion results in substitutions, if "out" is longer than "in", characters to the right of "in" are shifted right to leave enough space to insert "out". A diagnostic results if in the process nonblank characters are shifted beyond column 72 (and thus lost). If "out" is shorter than "in", then "out" is padded with blanks so as to occupy the same space as "in".

When type conversion is on (as it is if there is a "MAKE..." line in the control file), then all lines, except those with "++" or "−−" in columns 2 and 3, are examined in turn for the following:

If the "in" *keyword* is found, it is replaced by the "out" *keyword*.

Then "in" default function names for type conversion and getting machine constants (see "*T* Defined by:..." lines in Usage above) are searched for, and when found are replaced by the corresponding names of the "out" functions. When searching for names (here and below) they are only found if they are preceded and followed by a nonletter or nondigit. Function names which do not have "mach" as their last four letters are found only if followed by a '('.

Then the V_i's and v_i's generated from the "C−−L...", "C−−T...", and "C−−&" lines are searched for. When found, L_{out} (or T_{out} for names defined on "C−−T (Type)..." lines or "C−−&" lines following such lines) replaces the letter in the position of the "?", preserving the case of the letter that was there.

Then, those strings defined by the "C++ Replace..." lines are searched for and replaced if found.

Then, floating point constants are converted. These are assumed to consist of (not a letter).(0 or more digits)(the letter used for floating point constants in the current version)(an optional + or −)(a digit). If such a string is found, the letter is replaced as specified.

Whether type conversion is on or not, lines consisting of only "END" are searched for. When found, `m77con` checks if there are any "C++..." actions still active or if the processing is not in "usual precision" mode. In either of these cases a diagnostic is given and the processing returns to normal mode.

References

1. Fred T. Krogh, **A Language to Simplify Maintenance of Software which Has Many Versions**. Internal Computing Memorandum 359, Jet Propulsion Laboratory (April 1974).

2. Charles L. Lawson, **MARVEL – A Tool for Maintaining Multiversion Software**. Internal Report 463, Jet Propulsion Laboratory, Pasadena, CA (June 1980). Revised September 1989.

3. Charles L. Lawson, **CHGTYP – A Tool for Developing Fortran 77 Software in Single-Precision and Double-Precision Versions**. Internal Report 524, Jet Propulsion Laboratory, Pasadena, CA (April 1988). Revised May 1989.

E. Error Procedures and Restrictions

The program prints a variety of error messages on the standard output. An error in the control file causes the program to stop. In most cases of errors in the special comments of the files being processed, processing continues. After 10 comparison errors in a single file, no more comparison errors are printed for that file.

F. Supporting Information

The source language is ANSI Fortran 77.

Design and code due to F. T. Krogh, JPL, October 1994. Multiple minor modifications through April 1996.