NAME

fpp − cpp-like reversible preprocessor filter for Fortran and SFTRAN3 code

SYNOPSIS

fpp [**−?**] [**−author**] [**−copyright**] [**−D***name*] [**−D***name*=*value*] [**−debug**] [**−fixed**] [**−free**] [**−help**] [**−U***name*] [**−version**] [**name1=value1**] . . . [**namek=valuek**] [**−−**] [*infile(s)*]

DESCRIPTION

fpp is a preprocessor for Fortran (77, 90, 95) and SFTRAN3, modelled on the ISO Standard C (1989 and 1999) and C++ (1998) preprocessor, but tailored for Fortran use, and for *reversibility*. The output always contains the complete input, except that some code sections may have become comments, or vice versa. This can be more useful than the non-reversible approach used by the C preprocessor.

Reversible preprocessing is convenient when a master source file must be maintained to generate multiple versions, such as for different operating-system, compiler, and architecture variations. The source code for any of these can serve as the master file to create any of the others.

Although several UNIX vendor Fortran compilers offer to run the C preprocessor, **cpp**(1), it cannot be used reliably for Fortran code, because it does not know about Fortran line-length limitations, or Fortran syntax. Fortran does not require type declarations, so from syntax confusion, **cpp**(1) could introduce subtle changes that cannot be detected by the Fortran compiler, and would likely be quite hard to debug.

A similar problem exists with the more general and powerful macro processor, **m4**(1).

fpp follows the preprocessor syntax defined in three ISO language standards, but embeds directives in comments and restricts its operation, so that both input and output files can be processed by any Fortran compiler, or any other Fortran software tool, such as **d2s**(1), **dtoq**(1), **dtos**(1), **fsplit**(1), **ftnchek**(1), **pfort**(1), **pretty**(1), **qtod**(1), **s2d**(1), **sf3lex**(1), **sf3pretty**(1), **stod**(1), **strsf3**(1), **struct**(1), **tidy**(1), or **toolpack**(1). Also, a human can see exactly the same post-processed source code that the compiler does.

In particular, this means that it is often possible to apply **fpp** just *once* to a set of Fortran source files for a given environment, rather than having to run it each time the code is compiled.

OPTIONS

Because the original version of **fpp** was implemented in the **awk**(1) language, which has its own command-line options, **fpp** options were originally prefixed with a **+** instead of a **−**. That syntax is still recognized, as is the GNU/POSIX form with a leading double hyphen. Also, long option names may be abbreviated to any unique leading prefix. Thus, **+author**, **−author**, **−−author**, and **−a** are equivalent.

−− The preceding word on the command line is the last option; all following words are to be interpreted as filenames, even if they begin with a hyphen, or contain equal signs.

−? Display brief usage information on *stderr* and exit with a success status code before processing any input files.

This is a synonym for **−help**.

−author Show author information on *stderr* and exit with a success status code before processing any input files.

−copyright Show copyright information on *stderr* and exit with a success status code before processing any input files.

−D*name* Define the symbol *name* to the value 1.

−D*name*=*value* Define the symbol *name* to *value*.

−debug Turn on debugging output, which is sent to *stderr*. This produces helpful intermediate output from the expression evaluator. Macro definitions are also displayed on *stderr* when they are executed.

−fixed Input is in Fortran 77 fixed form. If this option is specified, it overrides any assumptions based on input file extensions.

Fixed-form input is the default.

In the absence of a command-line option to set the input form, fixed form is automatically selected for files with names ending in *.F*, *.f*, *.F77*, *.f77*, *.FOR*, *.for*, *.FPP*, *.fpp*, *.FTN*, or *.ftn*, or any unrecognized extension.

−free Input is in Fortran 90 and 95 free form. If this option is specified, it overrides any assumptions based on input file extensions.

In the absence of a command-line option to set the input form, free form is automatically selected for files with names ending in *.F90*, *.f90*, *.F95*, or *.f95*.

−help Display brief usage information on *stderr* and exit with a success status code before processing any input files.

This is a synonym for **−?**.

−U*name* Undefine the symbol *name*. If the name is subsequently referenced, it will *silently* evaluate to zero. The existence of a definition can be checked with the **defined** operator, or in

C#ifdef name
C#ifndef name

statements; see below for details.

−version Display the current version number of **fpp** on *stderr*, and exit with a success status code before processing any input files.

name=value Define the symbol *name* to *value*. This form is deprecated, and may not be supported in the future. Use **−D***name*=*value* instead.

If no input file names are given on the command line, input is assumed to come from *stdin*. In that case, since no filename is available to select the input form, it will be necessary to specify **−free** for free-form source code.

LANGUAGE OVERVIEW

fpp statements, or directives, are specially-formatted Fortran comments of the forms

C#name
C#name args

Since all **fpp** directives are encoded as comments, both input and output files should be compilable without any preprocessing by **fpp**.

Blanks may optionally surround the initial **#** to permit indentation for better visibility, or to reflect conditional statement nesting.

The first column may contain any valid Fortran comment starter: **C**, **c**, *****, or **!**.

Unrecognized **C # word** sequences are *silently* copied to the output, so as to permit the rare case of a **#** in the initial text of a Fortran comment.

Preprocessor names in conditionals and definitions, or set on the command line, consist of letters, digits, and underscores; the first character may not be a digit.

Symbols beginning with two underscores, or an underscore and an uppercase letter, are reserved for the local implementation; see the *PREDEFINED SYMBOLS* section below for details.

Symbols beginning with two underscores are permanent: once defined, they can be neither undefined, nor redefined, by the user.

Following standard Fortran practice, letter case is not significant in directives, or in constants and operators in expressions.

For portability, it is recommended that lower-case letters be used for all directives, and upper-case letters for all defined names; this conforms to three decades of widespread practice in the C programming language.

DEFINITION STATEMENTS

Definitions of names for preprocessor conditionals may be set on the command line:

fpp −D_OS_UNIX −D_SUN386
fpp −D_OS_UNIX=1 −D_SUN386=1
fpp _OS_UNIX=1 _SUN386=1

or in the input file text itself:

C #define _OS_UNIX 1
C #define _SUN386 1

The **cc**(1)-like forms with define and undefine options are supported; these two invocations are equivalent:

fpp -D_OS_UNIX -DWORDSIZE=32
fpp _OS_UNIX=1 WORDSIZE=32

These two are roughly equivalent:

fpp -U_OS_VAXVMS
fpp _OS_VAXVMS=0

They differ in that, although the symbol **_OS_VAXVMS** will evaluate to 0 in a numeric context, a test for definition with

C #if defined(_OS_VAXVMS)

or

C #ifdef _OS_VAXVMS

will select the else-branch in the first case, and the then-branch in the second case.

If the value is omitted, as in

fpp _OS_UNIX= _SUN386=

or

C #define _OS_UNIX
C #define _SUN386

a value of 1 is assumed.

Names can be undefined by

C #undef name
C #undefine name

If the name was not already defined, the request is *silently* ignored.

CONDITIONAL STATEMENTS

The conditional statements supported are:

C#if constant-expression
C#ifdef name
C#ifndef name
C#elseif constant-expression
C#elif constant-expression
C#else [optional comment]
C#endif [optional comment]

Each **C#ifxxx** statement must have a matching **C#endif** following it. The two may be separated by any number of **C#elseif** statements, which may be followed by a single **C#else** statement.

A branch of a conditional is selected when the expression evaluates to a non-zero value; see the *EXPRESSIONS* section below for details.

Code between these statements is preserved, but in the unselected branches of a conditional statement, a non-comment statement will be altered to a comment by prefixing it with an initial **C##**, (or **!##** in free form) shifting the statement right by three columns. In the selected branch, any initial **C##** in columns 1

through 3 is stripped; lines without this prefix are copied verbatim.

Because of Fortran line-length limitations (72 in fixed form, 132 in free form), this means that inside an **fpp** conditional, code lines may not exceed three characters less than the maximum length.

For example, the input

```
C#if  _OS_UNIX
C##C        UNIX  code
C##         CALL  GETENV(...)
C#elseif  _OS_VAXVMS
C        VMS  code
              CALL  LIB$TRNLNM(...)
C#endif
```

when **_OS_UNIX=1** produces

```
C#if  _OS_UNIX
C             UNIX  code
              CALL  GETENV(...)
C#elseif  _OS_VAXVMS
C##C        VMS  code
C##         CALL  LIB$TRNLNM(...)
C#endif
```

When neither **_OS_UNIX** nor **_OS_VAXVMS** are defined, the output is

```
C#if  _OS_UNIX
C##C        UNIX  code
C##          CALL  GETENV(...)
C#elseif  _OS_VAXVMS
C##C        VMS  code
C##         CALL  LIB$TRNLNM(...)
C#endif
```

When only **_OS_VAXVMS** is defined, the original input is sent to the output. If by chance both **_OS_VAXVMS** and **_OS_UNIX** were defined, only the UNIX code would be selected, because only the first branch of the conditional would be executed.

Preprocessor conditionals may be nested:

```
C  #  if  _OS_UNIX
C     #  if  _SUN
C           #  if  _SUN4
C           #  elseif  _SUN3
C           #  elseif  _SUN386
C           #  endif
C     #  endif
C  #  elseif  _OS_VMS
C  #  endif
```

Any text following **#else** or **#endif** is ignored; it can be used to document the conditional, usually with the test from the preceding **#if**:

```
C  #  if  _OS_UNIX
C  #  else  NOT  _OS_UNIX
C  #  endif  _OS_UNIX
```

While such trailing text is not permitted by the ISO Standard C/C++ preprocessor, many implementations of the preprocessor silently ignore such text.

fpp directives are not executed if they are in a branch of a conditional that is not currently selected. However, conditional statements are processed to keep track of the current nesting.

On UNIX, the **−Dname** option for **diff**(1) can be used to get output from the comparison of two files that is almost correct input for **fpp**. A simple command pipe **diff -Dxxx file1 file2 | sed -e 's/^#/C#/' >file3** will produce an output file *file3* from which **fpp -Dxxx file3** will recover *file2* and **fpp -Uxxx file3** will recover *file1*.

EXPRESSIONS

Expressions are recognized and evaluated in two circumstances: in the arguments of **C#if**, **C#elseif**, and **C#elif**, and inside the parentheses of **#(...)**.

In expressions, primaries are Fortran integer, floating-point, logical, and character constants, and preprocessor names.

Undefined names *silently* evaluate to zero in arithmetic expressions, and to empty strings in string expressions.

Character strings appearing in arithmetic expressions are converted to numbers, which are zero if the string does not look like a number. Character strings appearing by themselves evaluate to themselves.

Arithmetic expressions are evaluated in *floating-point* arithmetic; for Boolean (Fortran logical) tests, zero is false, and non-zero is true.

The usual Fortran arithmetic operators **+ − * / **** are recognized, along with the C modulus operator **%**; **x % y** is Fortran's **mod(x,y)**. This operator is rigorously defined for all arguments to be **x % y = x - int(x/y)*y**.

The Fortran logical and relational operators are supported, with convenient modern C-like synonyms: **.and.** (**&** and **&&**), **.or.** (| or ||), **.not.** (**!**), **.eq.** (==), **.ne.** (**!=**), **.lt.** (<), **.le.** (<=), **.gt.** (>), and **.ge.** (>=). Letter case in the dotted operators is not significant. Finally, the Fortran character string concatenation operator, //, is handled.

One special name, **defined**, is recognized, in any letter case; it may be used either in functional form, **defined(name)**, or in prefix operator form, **defined name**. It evaluates to 1 if the name is defined (even if the value of the name is zero), and otherwise, to 0. Several **defined** operators can be used in a single expression; that is much more convenient than a series of nested conditionals using **C#ifdef** and **C#ifndef**.

The **#(...)** form is only recognized in a comment line, and the next line is converted to a comment (see the section *MACRO EXPANSION* below); the parentheses hold an expression involving Fortran constants and preprocessor names.

Examples of expressions are

```
C#if  defined(_OS_UNIX)  ||  defined _OS_VAXVMS  ||  (WORDSIZE == 32)
C       REAL A(#(MAXA**2)), B(#(MAXA % 32))
C       INTEGER BITS(#(WORDSIZE))
```

MACRO EXPANSION

fpp supports a *reversible* argument-free macro expansion capability. This involves pairs of lines, the first a comment line containing the macro references as strings of the form **#(constant-expression)**, and the second a non-comment Fortran statement.

The first line of the pair is always exactly preserved in the output, while the second is replaced by the expansion of the comment, with the first character *deleted*, to change the comment into a non-comment. The original contents of the second line are preserved as a comment with the prefix **C-fpp-** in a third output line.

This peculiar input line pairing is necessary to ensure that the expansion is reversible.

The parentheses around the expression serve to distinguish between macros and **fpp** preprocessor directives in comments, and serendipitously permit the extension from simple names to arbitrary constant expressions that can be evaluated by **fpp**.

Care must be taken in writing the input to ensure that any expected expansion does not make the line longer than 72 characters; **fpp** has almost no knowledge of Fortran, and therefore cannot provide correct line

wrapping for it. However, it will warn about long lines.

Similarly, macro expansion in a multi-line continued statement should avoided, since it introduces comment lines between continuation lines. While such comments are legal in full Fortran 77, they are illegal in subset Fortran 77, and in older Fortrans, and may cause problems for other tools that process Fortran code.

Here is a small example. Given command-line definitions

```
FPTYPE='DOUBLE PRECISION'
MAXA=19
MAXB=25
```

then the input

```
C       #(FPTYPE) A(#(MAXA)), B(#(MAXB),#(MAXA**2))
        REAL A(100), B(255,10000)
```

is converted to the output

```
C       #(FPTYPE) A(#(MAXA)), B(#(MAXB),#(MAXA**2))
        DOUBLE PRECISION A(19), B(25,361)
C-fpp-          REAL A(100), B(255,10000)
```

Fortran 77 **PARAMETER** statements can be used to achieve similar effects, but in more restricted circumstances. In particular, **fpp** permits such expansions to happen in strings:

```
C10000 FORMAT ('Host operating system = #(OS)')
10000 FORMAT ('Host operating system = UNIX')
```

This may be awkward to achieve in standard Fortran.

MESSAGE OUTPUT STATEMENTS

Text can be written to *stderr* with either of

C#message text

C#error text

The difference between them is that **#error** sets an exit code of 1 (on POSIX and UNIX), and also sends the text to *stdout*. This can be used to ensure that a preprocessing error forces a compilation error if an attempt is later made to compile the output source program.

The output of both directives is prefixed with the file name and input line number to identify the origin of the message.

When **#error** is executed, processing is not terminated; instead, **fpp** tries to process the remaining input so as to uncover additional errors in the same run.

OUTPUT OF fpp

The output contains an initial comment header of the form

```
C-fpp- ================================================================
C-fpp- fpp version 1.0 [10-Dec-1990]
C-fpp- Date: Sat Dec  8 23:06:30 MST 1990
C-fpp- Directory: /u/sy/beebe
C-fpp- User: beebe@math.utah.edu
C-fpp- Macro: _OS_VAXVMS=1
C-fpp- Macro: FPTYPE=DOUBLE PRECISION
C-fpp- ================================================================
```

These comments provide a record of the processing, including what symbol definitions and macro values have been selected.

Input comments beginning

C-fpp-

are flushed. Thus, any existing comment header is always replaced by a new header.

Each command-line **name=value** or **−Dname** setting, and each input definition directive

C#define name value

produce an output comment of the form

C-fpp- Macro: name=value

Thus, all output lines beginning

C-fpp- Macro:

document which names have been defined.

A **C#undefine** statement results in output like

C-fpp- Macro: name=--UNDEFINED--

PREDEFINED SYMBOLS

Each implementation of **fpp** predefines a few symbols that can be tested and used in conditionals to select machine-specific code sections. The predefined symbols are provided to **fpp** ahead of any user-defined ones. Since later redefinitions override earlier ones, predefined symbols can always be changed by the user.

Following ISO Standard C/C++, predefined symbols always begin with two underscores, or an underscore and an uppercase letter; such names are reserved for the local implementation. Predefined symbols that do not follow this convention are *forbidden*. This requirement makes it possible to distinguish separate name spaces for the user and for the implementation, preventing surprises from unexpected substitutions that happen when code is moved to a new environment.

The complete set of definitions is always recorded in the output header; they can easily be displayed on *stdout* by giving **fpp** an empty input file:

fpp*/dev/null*

fpp always predefines *exactly one* major operating-system symbol:

_OS_PCDOS
_OS_TOPS20
_OS_UNIX
_OS_VAXVMS

For **_OS_UNIX**, exactly one minor operating-system variant may also be defined:

_AIX
_AIX370
_BSD
_DARWIN
_FREEBSD
_GOULD
_HPUX
_IRIX
_LINUX
_MACH
_MIPS
_NETBSD
_OPENBSD
_OSF1
_RHAPSODY
_STARDENT
_SUNOS
_ULTRIX

Additional architectural variants may be defined on some systems:

_GNU_LINUX
_HPPA

_IBM_3090
_IBM_PS_2
_IBM_RS_6000
_IBM_RT
_IRIX64
_M68K
_MACOSX
_NEXT
_POSIX
_PPC
_STARDENT_1500
_STARDENT_3000
_SUN3
_SUN386
_SUN4
_VAX
_X86

The operating system name as returned by **uname**(1) is recorded as the value of **_OS_NAME**, after collapsing characters other than letters, digits, period, and hyphen to underscores.

The operating system level, usually a string of digits separated by periods and/or hyphens, is recorded as the value of **_OS_LEVEL**. On most systems, it too is obtained from **uname**(1).

The host CPU architecture is recorded as the value of **_ARCH**, one of

Alpha
Convex
Cray
Gould
IA-64
IBM-3090
MIPS
Motorola-68K
PA-RISC
PowerPC
SPARC
Stardent
VAX
unknown
x86

Host byte addressing order is defined by one of these:

_BIG_ENDIAN
_LITTLE_ENDIAN

Big-endian addressing is used by IBM, Motorola, and most RISC systems. Little-endian addressing is used by the Intel x86, HP/Intel IA-64, DEC VAX, and Compaq/DEC Alpha architectures. Although a few RISC architectures support both endian orders, a fixed choice is always made by the operating system to ensure consistent byte ordering in binary files and network traffic.

Host floating-point architecture must be defined by

_IEEE_754

on those machines that have IEEE 754 floating-point arithmetic.

If the Fortran implementation supports NAMELIST I/O, the symbol

_NAMELIST

must be defined.

To ensure standardization, all such names must be registered with the author of **fpp**, and will be listed in

these manual pages.

Following ISO Standard C/C++, four standard permanent symbols are always defined; these each have two leading and two trailing underscores. Permanent symbols always begin with two underscores, and once defined, may not be undefined, or redefined, by the user.

__DATE__ Current calendar date in the form *Mmm dd yyyy*. The month field is alphabetic, and the day number field has a leading blank if the day is less than 10.

__FILE__ Current input file filename.

__LINE__ Current input file line number.

__TIME__ Wall-clock time in the form *hh:mm:ss*.

In addition to those four, **fpp** sets two related values:

__ISO_DATE__ Ten-character ISO 8601 date in the form *YYYY-MM-DD*.

__TIMEZONE__ Three-letter time zone abbreviation, such as MDT for Mountain Daylight Time, or GMT for Greenwich Mean Time.

The symbols **__DATE__**, **__ISO_DATE__**, **__TIME__**, and **__TIMEZONE__** are set only once, at the start of execution of **fpp**. These values can be conveniently used to generate output stamped with the time of compilation:

```
C       WRITE (*,*) 'Processed on #(__DATE__) at #(__TIME__) #(__TIMEZONE__)'
      WRITE (*,*) 'Processed on ??? ?? ???? at ??:??:?? ???'
```

might produce

```
C       WRITE (*,*) 'Processed on #(__DATE__) at #(__TIME__) #(__TIMEZONE__)'
      WRITE (*,*) 'Processed on Dec 10 1990 at 09:10:07 MST'
C-fpp-        WRITE (*,*) 'Processed on ??? ?? ???? at ??:??:?? ???'
```

DIAGNOSTICS

Diagnostics are issued to *stderr* if unclosed conditionals, out-of-place conditional branches, errors in preprocessor expressions, or long lines, are detected. Attempts to redefine permanent macros (any that begin with two underscores) produce an error message. Debug output requested by a command-line option will be sent to *stderr*.

Directives

C#error text

that are executed send their text argument to *stderr* and to *stdout*, and cause a later exit code of 1 (on POSIX and UNIX).

Directives

C#message text

that are executed send their text argument to *stderr*.

Diagnostic messages have the format

filename:linenumber:message

commonly used by UNIX, POSIX, and GNU software. Advanced text editors, such as **emacs**(1), recognize that format, and allow the user to move to the error location with a couple of keystrokes.

SEE ALSO

awk(1), **c++**(1), **CC**(1), **cc**(1), **cpp**(1), **cxx**(1), **d2s**(1), **diff**(1), **dtoq**(1), **dtos**(1), **emacs**(1), **f77**(1), **f90**(1), **f95**(1), **fsplit**(1), **ftnchek**(1), **g++**(1), **g77**(1), **gawk**(1), **gcc**(1), **lf95**(1), **m4**(1), **mawk**(1), **nagf90**(1), **nagf95**(1), **nawk**(1), **pfort**(1), **pgf77**(1), **pgf90**(1), **pretty**(1), **qtod**(1), **s2d**(1), **sf3lex**(1), **sf3pretty**(1), **stod**(1), **strsf3**(1), **struct**(1), **tidy**(1), **toolpack**(1), **uname**(1), **xlf**(1), **xlf90**(1), **xlf95**(1), **xsf3**(1).

American National Standards Institute, *American National Standard programming language FORTRAN: approved April 3, 1978*, ANSI X3.9-1978, New York, 1978. Revision of ANSI X3.9-1966.

S. P. Harbison and G. L. Steele, Jr., *C: A Reference Manual*, 4th ed., Prentice-Hall, 1995.

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed., Prentice-Hall, 1988.

American National Standards Inst., *American National Standard for Information Systems — Programming Language — C*, ANSI X3.159-1989, New York, 1990.

International Organization for Standardization, *ISO/IEC 9899:1990: Programming languages — C*, Geneva, Switzerland, 1990.

International Organization for Standardization, *International standard: information, technology, programming languages, Fortran*, ISO/IEC 1539:1991, Geneva, 1991.

IEEE, *9945-2: 1993 (ISO/IEC) [IEEE/ANSI Std 1003.2-1992 and IEEE/ANSI 1003.2a-1992] Information Technology — Portable Operating System Interface (POSIX(®)) — Part 2: Shell and Utilities*, New York, 1993.

American National Standards Institute, *ANSI/ISO/IEC 1539-1:1997: Information technology — Programming languages — Fortran — Part 1: Base language*

Jeanne C. Adams, Walter S. Brainerd, Jeanne T. Martin, Brian T. Smith, and Jerrold L. Wagener, *Fortran 95 Handbook: Complete ISO/ANSI Reference*, MIT Press, Cambridge, MA, 1997. ISBN 0-262-51096-0

International Organization for Standardization, *ISO/IEC 14882:1998: Programming languages — C++* Geneva, Switzerland, 1998.

International Organization for Standardization, *ISO/IEC 9899:1999: Programming languages — C*, Geneva, Switzerland, 1999.

AUTHOR

Nelson H. F. Beebe
Center for Scientific Computing
University of Utah
Department of Mathematics, 322 INSCC
155 S 1400 E RM 233
Salt Lake City, UT 84112-0090
USA
Email: `beebe@math.utah.edu`, `beebe@acm.org`,
`beebe@computer.org`, `beebe@ieee.org` (Internet)
WWW URL: `http://www.math.utah.edu/~beebe`
Telephone: +1 801 581 5254
FAX: +1 801 585 1640, +1 801 581 4148

AVAILABILITY

fpp is freely available; its master distribution can be found at

```
ftp://ftp.math.utah.edu/pub/misc/
http://www.math.utah.edu/pub/misc/
```

in the file *fpp-x.yy.tar.gz* where *x.yy* is the current version. Other distribution formats are usually available at the same location.

That site is mirrored to several other Internet archives, so you may also be able to find it elsewhere on the Internet; try searching for the string *fpp* at one or more of the popular Web search sites, such as

```
http://altavista.digital.com/
http://search.microsoft.com/us/default.asp
http://www.dejanews.com/
http://www.dogpile.com/index.html
http://www.euroseek.net/page?ifl=uk
http://www.excite.com/
http://www.go2net.com/search.html
http://www.google.com/
http://www.hotbot.com/
http://www.infoseek.com/
```

```
http://www.inktomi.com/
http://www.lycos.com/
http://www.northernlight.com/
http://www.snap.com/
http://www.stpt.com/
http://www.yahoo.com/
```

COPYRIGHT