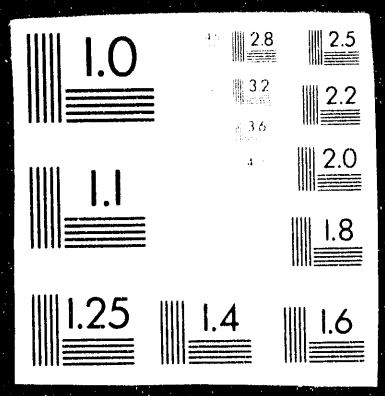
1 OF 1



SLAC--408

DE93 007016

FPP - A Fortran Preprocessor

Adam Boyarski

Stanford Linear Accelerator Center
Stanford University
Stanford, California 94309, USA

November 1992

Prepared for the Department of Energy
under contract number DE-AC03-76SF00515

Abstract

FPP is a preprocessor which aids in porting Fortran source code across differing platforms. It provides conditional compilation features to enable or disable sections of code, and can modify file names in INCLUDE statements to a syntax suitable for a target platform. FPP is written in Fortran 77, and runs on VM/CMS, VAX/VMS, UNIX, and PC/DOS systems.

1. Introduction

When Fortran code is moved from one platform to another, it is frequently found that parts of the code need modification on the new platform. Extensions used on one compiler may not be recognized by the compiler on another platform. System calls, such as calls to get the date and time, or calls to issue a command to the operating system, may differ between platforms. The file name syntax on Fortran INCLUDE statements or OPEN statements may also differ from one platform to another. For all these reasons, it is often impossible to have a fortran file that compiles and executes successfully on all platforms.

One solution to this problem is to keep separate versions of Fortran code on each platform, each modified to work correctly on its platform. But then code maintenance becomes difficult, since a change to one file may require the same change be made to the files on other platforms. Another solution is to write a single master file in a syntax that a preprocessor can recognize to produce Fortran code for each platform. A disadvantage with this method is the need for two source files, a master file and the Fortran file on each platform. Alternately, a preprocessor can be written to use the Fortran code on any one platform and modify it for use on any other platform, without the need for a master file. The latter scheme has been adopted by FPP.

FPP is a preprocessor that provides conditional compilation by means of a control language somewhat like that in CPP, the preprocessor for the C language. However, FPP never deletes or inserts any lines of source code, but instead enables or disables lines of code with the use of the comment character. FPP also modifies file names in INCLUDE statements to syntax appropriate for a target platform. Code processed by FPP for one platform can be used in turn to generate code for any other platform by processing it with FPP on the target platform. Disabled lines of Fortran code on one platform can be enabled when FPP processes the code on another platform. Lines of Fortran are disabled by inserting a "C" in column one, lines are enabled by removing the "C" in column one. There is no need for a "master" source file from which are made the compiled source files for each platform. The compiled file and the master file can be the same on each platform, as long as the file is processed by FPP on each platform before the compilation step.

2. Conditional Compilation

Conditional compilation is done by means of commented control lines in the Fortran source code. These control lines are comments to the Fortran compiler, but are treated as commands by FPP as it processes the Fortran source file. Control statements must begin with the string "C..#" or "CC..#", starting in column one. The user places these control statements in the source code before and after lines that are to be enabled or disabled. The following control commands are recognized:

C..#IFDEF <key> <key> ...
C..#ELIF <key> <key> ...
C..#ELSE
C..#ENDIF

where <key> is a control string. If any <key> on a IFDEF or ELIF control statement matches any of the defined keys supplied on the FPP command line (described later), then the block of code following this control statement up to the next control statement is enabled by FPP. If there is no match, the block of code is disabled by FPP. If no keys match, then the code following the ELSE control statement is enabled. The

ENDIF control statement terminates the IFDEF-ELIF-ELSE control sequence. These control statements may be nested to any level. For example, if we have the following code on VM:

```
     <normal code>
C..#IFDEF VM
     <code for VM>
CC..#ELIF VAX
C      <code for VAX>
CC..#IFDEF UG
C      <code for VAX and UG>
CC..#ELSE
C      <code for VAX, non UG>
CC..#ENDIF
CC..#ELIF RS6000
C      <code for RS6000>
CC..#ELSE
C      <code for all other platforms>
C..#ENDIF
     <normal code>
```

where the conditional block "code for VM" following the "C..#IFDEF VM" control statement is enabled for this VM version, and other conditional lines following the "CC..#ELIF VAX" or "CC..#ELIF RS6000" control statements are disabled (commented out). Note also the nested control statements "#IFDEF UG", "#ELSE", and "#ENDIF" within the VAX block. If this file is sent to a VAX platform, and FPP were invoked on that platform as follows:

```
fpp -DVAX -DUG infile
```

then the above VM code would be changed to:

```
     <normal code>
CC..#IFDEF VM
C      <code for VM>
C..#ELIF VAX
     <code for VAX>
C..#IFDEF UG
     <code for VAX and UG>
CC..#ELSE
C      <code for VAX, non UG>
C..#ENDIF
CC..#ELIF RS6000
C      <code for RS6000>
CC..#ELSE
C      <code for all other platforms>
C..#ENDIF
     <normal code
```

Note that every conditional block of code not contained in the VAX block is commented out with a "C" inserted at column one, *including* the preprocessor command lines before each such block. Also, the nested "#IFDEF UG" has the UG code enabled within the VAX block, because both the VAX and UG keys were supplied with the "-D" option on the FPP command line.

If either the VM or VAX versions of the above code were copied to the RS6000 platform and preprocessed on that platform with "FPP -DRS6000 infile", then in both cases the resultant code would be the same with only the conditional code following the "#ELIF RS6000" control statement being enabled.

3. File names on INCLUDE Statements

The INCLUDE statement has been implemented in many Fortran 77 compilers, and is part of the Fortran 90 standard. However, the file name conventions differ from platform to platform, and so the INCLUDE statement cannot be written in a platform independent way. Also, the INCLUDE syntax in Fortran 77 compilers is not always compatible, some allow single quotes around the file name, some allow double quotes, and some allow parentheses around the file name. It is sometimes desirable to prefix the file name with a directory path which specifies where in the file system the included file is to be obtained. Path names are again platform dependent. It would be desirable for a preprocessor to be able to edit the prefix and suffix of file names on INCLUDE statements. FPP has this ability.

The options "-P" and "-S" on the FPP command line specify the prefix or suffix for included file names, as described in the next section. The following example illustrates how FPP can modify INCLUDE file names. If the file "myfile" contains a Fortran statement like:

```
INCLUDE (jumbo)
```

and FPP is invoked on VM as follows:

```
fpp -P' -S.cmn' myfile
```

or on a VAX or UNIX platform as*

```
fpp "-P'" "-S.cmn'" myfile
```

then the resultant line of code would appear as:

```
INCLUDE 'jumbo.cmn'
```

where the old prefix "(", and old suffix ")" are stripped out and the new prefix "'" and suffix ".cmn'" are inserted.

It is sometimes desirable to be able to include files from several directories. This can be done with FPP using the "-I" option, which provides a means for specifying a list of directories to be searched for include files. When an include file is found in a directory, the path string is inserted between the prefix and the file name on the INCLUDE statement. The Fortran compiler will then be able to access the file from the path given on the INCLUDE statement.

If the "jumbo" include file were in another directory in a UNIX system, say in "../includes", then

```
fpp -I../includes "-P'" "-S.cmn'" myfile
```

would change the line of code in the source file to

```
INCLUDE '../includes/jumbo.cmn'
```

*Note -- On UNIX or VAX, any command line token with an apostrophe should be surrounded by quotes.

4. FPP Description

Following is a description of the FPP command, and the FPP options.

SYNTAX:

```
fpp [options] Filename
```

Filename specifies a file name, with or without an extension. (On VM, dots must separate the file name, the file type, and the file mode). If the extension is not supplied, defaults are provided on each platform. For example, if Filename is given as "MYFILE" on VM/CMS, then a ".FORTRAN.*" extension is added by FPP. On a VAX the default extension is ".FOR", and on UNIX it is ".f". The default extension, as well as default option settings, may be seen on any platform by issuing fpp without any argument.

OPTIONS

-Dkey Defines a key control string for conditional compilation. This option is repeatable.

-Ooutname FPP writes its output to the file "outname". If this option is not given, and FPP detects that changes were made, then Filename is updated.

-Idirectory Inserts "directory" into the search path for INCLUDE files. This option is repeatable. The search order starts with the current directory, then directories in the order given by the -I options, until the INCLUDE file is found. This option is useful for those platforms having compilers that cannot search subdirectories for INCLUDE files (e.g. sun, next). This option is ignored when the -K option is present.

-Pprefix supplies the "prefix" string to be prefixed to the file names on INCLUDE statements. FPP strips off the old prefix and adds the new one on each INCLUDE statement. If this option is not supplied, a default prefix for each platform is provided. This option is ignored when the -K option is present.

-Ssuffix supplies the "suffix" string to be suffixed to the file names on INCLUDE statements. FPP strips off the old suffix and adds the new one on each INCLUDE statement. If this option is not supplied, a default suffix for each platform is provided. This option is ignored when the -K option is present.

-U Change INCLUDE file names to upper case. This option is ignored when the -K option is present.

-L Change INCLUDE file names to lower case. This option is ignored when the -K option is present.

-M Leave file names on INCLUDE statements in mixed case. This is the default if neither the -U nor -L options are given. This option is ignored when the -K option is present.

-K Preserve the file name syntax and case on INCLUDE statements. This option disables the -I,-P,-S,-U,-L,-M options.

5. ERROR MESSAGES

The error messages from FPP are shown below. FPP returns a STOP code of 101 in each case.

No output file name specified.
More than one input file given.
Can not parse command line.
File does not exist.
Can not open input file.
Can not open output file.
#IFDEF nesting overflow.
Incorrect #IFDEF ... #ENDIF structure...
Unknown fpp command.
Internal FPP error.
Error while deleting input file.
Error while closing input file.
Error while closing output file.
Error while copying fpptemp file.

6. FPP Availability

FPP is installed on the SLACVM (U disk), SLACVX (in user [AMB]), and the SCS UNIX cluster (Next, sun4, and aix) in /usr/local/bin. On VM, an exec file (FPP EXEC) is used for passing command line arguments to the FPP MODULE. When installing FPP on a VAX, the file (FPP.CLD) must be edited, and the DCL command "set command fpp" issued in order to make fpp into a DCL command.

FPP is available on the internet by anonymous ftp from heplib.slac.stanford.edu in /pub/fpp.tar.Z (It may move to another subdirectory in the future).

FPP is written in Fortran77 and is itself written with FPP conditional compilation statements. The following keys are built into the FPP source file at the time of this writing: VM_CMS, VAX_VMS, RS6000_XLF, SUN4, NEXT_ABSOFT, UNIX, and PC_F77L. Once an FPP executable exists, the FPP source file may be converted to any other by issuing "fpp -Dkey fpp" with the key for that system.

When installing FPP on some other platform, the FPP Fortran file may have to be editted. The statements for getting the command line arguments and some OPEN statements may have to be changed, and new DATA statements added for defining defaults for the new platform.

END

DATE
FILMED
3/8/93