fpp - A Fortran pre-processor

1. Introduction

Frequently Fortran programmers need to maintain more than one version of a code, or to run the code in various environments. The easiest solution for the programmer is to keep a single source file that has all the code variations interleaved within it so that any version can be easily extracted. This way, modifications that apply to all versions need only be made once.

Source code preprocessors have long been used to provide these capabilities. They allow the user to insert directive statements within the source code that effect the output of the preprocessor. Typically, these special directives may appear as comment lines to a compiler. In general, source code preprocessors permit the user to define special variables and logical constructs that conditionally control which source lines in the file are passed on to the compiler and which lines are skipped over. In addition, the preprocessor's source line editing capabilities allow the user to specify how the source code should be changed according to the value of defined string variables.

Historically, the source code preprocessor found in standard C compilers, cpp, has been used to provide Fortran programmers with these capabilities. However, cpp is too closely tied into the C language syntax and source line format to be used without careful scrutiny. The proposed Fortran PreProcessor, fpp, would provide Fortran-specific source code capabilities that C programmers have come to expect in UNIX environments.

Some of fpp's capabilities could, potentially, be reapplied in other tools. An implementation of fpp would have to parse and analyze Fortran source, evaluate Fortran expressions and generate Fortran output. These are the building blocks that other tools would need to provide an enriched environment for Fortran program development.

The following section specifies the capabilities of fpp in more detail.

2. Detailed Specifications

PreProcessor Source

fpp's input source is standard Fortran with interspersed preprocessor statements. All preprocessor statements begin with a special character, #. It may only appear at the first character position in a line. The # should be immediately followed by the text of the preprocessor command.

Preprocessor directives may appear inside Fortran comments. The preprocessor will also modify the text inside a Fortran comment.

The preprocessor will operate on both fixed and free form source. Fixed form source files would have the extension .F while free form source files would have the extension .F90. Note that the fixed form file name convention applies to both FORTRAN 77 and Fortran 90. Files with such extensions will be preprocessed first before the appropriate compiler is invoked.

If the expansion of a macro or string replacement causes the column width of a line to exceed column 72 (for fixed form) or column 132 (for free form), the preprocessor will generate the appropriate continuation lines.

PreProcessor Variables

Preprocessor variables are defined by directives to have a string value. fpp will replace occurrences of the variable within the source code with its value wherever appropriate. Variables also appear on conditional directives and control the selection of source code lines passed to output.

```
#define name    string
```

defines the value of the variable "name" to be "string", with the result that every occurrence of the token "name" [outside of IMPLICIT statement bounds, text in FORMAT statements & string literals] is literally replaced by "string".

Note: In case of IMPLICIT statement bounds and text in FORMAT statements, macros are not expanded only in case of conflict with valid literals in that context.

"Macro"-like, inline replacements with arguments are also permitted:

```
#define name (arg1 [,arg2]...) string-with-args-inserted
```

Examples:

```
#define SOLARIS_2       .TRUE.

#define CONVERT(TO_FARENHEIT) ((TO_FARENHEIT*9)/5)+32
```

Preprocessor variables are enabled from the point of definition upto the end of the compilation unit. (i.e. global scoping across the file)

Preprocessor variables can be explicitly undefined using

```
#undef name
```

Comments

C style comments enclosed within /* and */ is accepted by the Fortran preprocessor.

Conditional Source Code Selection

This is modeled on the cpp conditional code construct.

```
#if condition1
      block1
[#elif condition2
      block2 ]
  ...
[#else
      blockn ]
#endif
```

or

```
#ifdef name
      block
#endif

#ifndef name
      block
#endif
```

The conditions are fpp expressions involving preprocessor variables (specified by #define statements). Note that the conditions may be specified within parentheses. These fpp expressions are a superset of cpp expressions in that they can evaluate Fortran logical literals. So, .TRUE. is valid in an fpp expression. The preprocessor evaluates these conditions to get a true or false result. The expressions may not evaluate floating point values or include intrinsic functions.

Example:

```
#define SOLARIS_2      .TRUE.
#if (SOLARIS_2)
   CALL solaris_2 (X,Y,Z)
#else
   CALL solaris_1 (X,Y,Z)
#endif
```

Including Files

Sometimes, it is convenient to collect preprocessor variables in a file. This is possible with

```
#include filename
```

Nesting of #include is permitted. #include statement may not appear in a continued line (i.e. the first statement in an included file may not be a continuation line)

Preprocessor Intrinsic Functions

defined(name) provides the definition status of a name. It returns TRUE or FALSE depending on whether the name is defined or not. This is, typically, used in a #if statement:

    #if defined(BIG_MODEL)

Command line options

fpp is typically invoked from the Fortran compiler driver. It can also be invoked stand alone. The following are some of the options that fpp accepts:

-Dname	Define name as 1. This is the same as if a *#define name 1* line appeared in the source file that fpp is processing.
-Dname=def	This is the same as if a *#define name def* line appeared in the source file that fpp is processing.
-Idirectory	Insert 'directory' into the search path for #include files with names not beginning with '/'. 'directory' is inserted ahead of the standard list of 'include' directories. So, #include files with names enclosed in double-quotes (") are searched first in the directory of the file with the #include line, then in directories named with the -I options, and lastly, in the directories from the standard list.
-Uname	Remove any initial definition of 'name', where 'name' is predefined by the preprocessor.
-Ydirectory	Use 'directory' in place of the standard list of directories when searching for #include files.