Preprocessor definitions

Compilation of the single Fortran code may seem as a one step process at first sicne the most of users will use a simple command

```
gfortran hello.f90 -o hello.x
```

There are only two files in the command above i.e. source code `hello.f90` and the binary output file `hello.x` . The compilation is however much more complicated and involves several steps. Each of the steps may be controlled by programmer.

The first step in the compilation is called preprocessing. The program called preprocessor will parse the code and interpret every directive creating an augmented source code. Preprocessing is a standard routine in `C` , however, it can be also done in Fortran. Below we will examine two most common applications of preprocessor in Fortran.

Preprocessor definitions

Preprocessor allows for definition of variables or rather strings that will be replaced during the preprocessing step into their assigned values. There is no need therefore to define particular variable, its type and scope rather define a preprocessor variable (called macro) and its value (token for which a macro is an abbreviation). General form is

```
#define NAME_OF_ITEM value
```

for example

```
#define BLOCK_SIZE 1024
```

In the example above `BLOCK_SIZE` may be used in the program and the preprocessor will replace it to value 1024 during the compilation. `C` language mathematical header file `math.h` includes number of predefined variables. Fortran lacks such functionality, however the preprocessor may be used to handcraft a similar solution. For example

We may write `math.h` file

```
#ifndef FORTRAN_MATH_H
#define FORTRAN_MATH_H

#define M_E         2.71828182845904523536028747135266250   /* e
*/
#define M_LOG2E     1.44269504088896340735992468100189214   /* log2(e)
*/
#define M_LOG10E    0.43429448190325182765112891891660508 2  /* log10(e)
*/
#define M_LN2       0.69314718055994530941723212145817656 8  /* loge(2)
*/
#define M_LN10      2.30258509299404568401799145468436421   /* loge(10)
*/
#define M_PI        3.14159265358979323846264338327950288   /* pi
*/
#define M_PI_2      1.57079632679489661923132169163975144   /* pi/2
*/
#define M_PI_4      0.78539816339744830961566084581987572 1  /* pi/4
*/
#define M_1_PI      0.31830988618379067153776752674502872 4  /* 1/pi
*/
#define M_2_PI      0.63661977236758134307553505349005744 8  /* 2/pi
*/
#define M_2_SQRTPI  1.12837916709551257389615890312154517   /* 2/sqrt(pi)
*/
#define M_SQRT2     1.41421356237309504880168872420969808   /* sqrt(2)
*/
#define M_SQRT1_2   0.70710678118654752440084436210484903 9  /* 1/sqrt(2)
*/

#endif
```

and use it in the Fortran source

```
program math

#include "math.h"

    print *,'M_E        : ', M_E
    print *,'M_LOG2E    : ', M_LOG2E
    print *,'M_LOG10E   : ', M_LOG10E
    print *,'M_LN2      : ', M_LN2
    print *,'M_LN10     : ', M_LN10
    print *,'M_PI       : ', M_PI
    print *,'M_PI_2     : ', M_PI_2
    print *,'M_PI_4     : ', M_PI_4
    print *,'M_1_PI     : ', M_1_PI
    print *,'M_2_PI     : ', M_2_PI
    print *,'M_2_SQRTPI : ', M_2_SQRTPI
    print *,'M_SQRT2    : ', M_SQRT2
    print *,'M_SQRT1_2  : ', M_SQRT1_2

end program math
```

In the above `#include "math.h"` is a preprocessor directive to include `math.h` file. That file contains number of `#define` statements which create macros for many commonly used numerical values.

Preprocessor conditional compilation

`math.h` file that we have created above contains also an conditional statement of type

```
#ifndef UNIQUE_NAME_H
#define UNIQUE_NAME_H

[body]

#endif
```

Preprocessor conditional is similar to `Fortran` conditional statement `if` with this difference that if the condition is not satisfied the part of code will not be included during the compilation. The example above uses the oposite, it checks if the variable `UNIQUE_NAME_H` has not been defined, and defines it then together with including the code. This is common practice for header files.

Body of the source code may contain different sequence

```
#ifdef UNIQUE_NAME
    [ body 1 ]
#else
    [ body 2 ]
#endif
```

In this case the compiler will use `[body 2]` section unless we define `UNIQUE_NAME` variable during the compilation. If we give `-DUNIQUE_NAME` switch to the compiler then that variable (macro) will be defined and the preprocessor will enable the `[body 1]` part of the code. While in `C` preprocessor is an integral part of the compilation process, we may need to enable it for Fortran.

For example

```
program preprocessor_example

  implicit none

#ifdef BLOCK_ONE
  print *,'Using block 1'
#else
  print *,'Using default block 2'
#endif

end program preprocessor_example
```

compiled using `gfortran` with the command

```
$ gfortran preprocessor.f90 -o preprocessor.x
Warning: preprocessor.f90:5: Illegal preprocessor directive
Warning: preprocessor.f90:7: Illegal preprocessor directive
Warning: preprocessor.f90:9: Illegal preprocessor directive
$ ./preprocessor.x
 Using block 1
 Using default block 2
```

will not use the preprocessor. However, the command

```
$ gfortran -cpp preprocessor.f90 -o preprocessor.x
$ ./preprocessor.x
 Using default block 2
```

will use the preprocessor correctly (`-cpp` switch enables the preprocessor). We may also define the `BLOCK_ONE` macro e.g.

```
$ gfortran -cpp -DBLOCK_ONE preprocessor.f90 -o preprocessor.x
$ ./preprocessor.x
 Using block 1
```