The C Preprocessor

Andrea Mignone
Physics Department, University of Torino

AA 2018-2019

The C Preprocessor

- The C preprocessor is a macro processor that is used automatically by the C compiler to transform your program before actual compilation.
- In many C implementations, it is a separate program invoked by the compiler as the first part of translation.
- It allows you to define macros which are brief abbreviations for longer constructs;
- The C preprocessor provides four separate facilities:

  1) Inclusion of header files
  2) Macro expansion
  3) Conditional compilation
  4) Line control

- Preprocessor directive always start with a ‘#’ symbol, example:

```
#define    TRUE       1
```

1) The #include directive

- The #include directive tells the preprocessor to insert the contents of another file into the source code at the point where the #include directive is found.
- Include directives are typically used to include the C/C++ header files for functions that are held outside of the current source file.
- The syntax is

```
#include <header_file>

// or

#include “header_file”
```

- If a header file is included within <>, the preprocessor will search a predetermined directory path to locate the header file.
- If the header file is enclosed in "", the preprocessor will look for the header file in the same directory as the source file.
- Note: it is **BAD** idea to include a function through an include directive.

2) The #define directive for constants

- The #define directive allows the definition of macros within your source code;
- Macros allow constant values to be declared for use throughout your code:

```
#define   TRUE          1
#define   FALSE         0
#define   CONST_c       2.99792458e10  // speed of light

int main()
{
  int a;
  double csq = CONST_c*CONST_c;

  cout << “Enter 0 or 1 “;
  cin >> a >> endl;
  if (a == TRUE) cout << “You typed true !!”
}
```

- Beware that macro definitions are *not* variables and are understood as simple replacement. They cannot be changed by your program code like variables;
- It’s a good practice to define macro constant names in uppercase;
- There’s no semicolon character at the end of a preprocessor statement.

2) The #define directive for Function-like macro

- The #define directive can also be used with arguments allowing a function-like construct to be used.
- As an example, consider using a macro for degrees-to-radians conversion:

```
...
#define   DEG2RAD(x)    ( (x)*M_PI/180.0 )

int main()
{
  double alpha = 30.0;
  cout << sin(DEG2RAD(x)) << endl;
  ...
}
```

- This is expanded in-place, so that repeated multiplication by the constant is not shown throughout the code.
- The macro here is written as all uppercase to emphasize that it is a macro, not a compiled function.

2) The #define directive for Function-like macro

```
...
#define   DEG2RAD(x)    ( (x)*M_PI/180.0 )

int main()
{
  double alpha = 30.0;
  cout << sin(DEG2RAD(x)) << endl;
  ...
}
```

- IMPORTANT: the argument is enclosed in parenthesis to avoid the possibility of incorrect order of operations when it is an expression instead of a single value:

```
// ! INCORRECT
#define   DEG2RAD(x)    x*M_PI/180.0

sin(DEG2RAD(30+60))  expands to  sin(30+60*M_PI/180.)
```

```
// CORRECT
#define   DEG2RAD(x)    (x)*M_PI/180.0

sin(DEG2RAD(30+60))  expands to  sin( (30+60)*M_PI/180.)
```

3) Conditional Compilation

- In some occasions, you may instruct the preprocessor whether to include certain part of the code or not. To do so, conditional directives can be used:

```
#define SESSION 2  // Choose session number (1, 2 or 3)

int main()
{
  ...
#if SESSION  == 1
  ...stuff for 1st session...
#endif
#if SESSION  == 2
 ...stuff for 2nd session...
#endif
...
}
```

- In the previous example, only the part of the code enclose in the second `#if...#endif` statement will be compiled.
- Note that `#if` statement is not tested at runtime but during the compilation stage.

3) Conditional Compilation

- Optionally you may also use the #else directive:

```
#if expression
   conditional codes if expression is non-zero
#else
   conditional if expression is 0
#endif
```

- Or you can also add nested conditionals to your #if...#else using #elif

```
#if expression
   conditional codes if expression is non-zero
#elif expression1
    conditional codes if expression is non-zero
#elif expression2
    conditional codes if expression is non-zero
... .. ...
#else
   conditional if all expressions are 0
#endif
```