C Pre-processor

hitbullseye

- The C preprocessor is a macro preprocessor (allows you to define macros) that transforms your program before it is compiled. These transformations can be the inclusion of header files, macro expansions, etc. All preprocessing directives begin with a # symbol.
- For example,

  #define PI 3.14
- Next slides contain common uses of C Preprocessor

Including Header Files: #include

- The #include preprocessor is used to include header files to C programs. For example,

  #include <stdio.h>

- Here, stdio.h is a header file. The #include preprocessor directive replaces the above line with the contents of stdio.h header file. That's the reason why you need to use #include <stdio.h> before you can use functions like scanf() and printf(). You can also create your own header file containing function declaration and include it in your program using this preprocessor directive.

Macros using #define

- A macro is a fragment of code that is given a name. You can define a macro in C using the #define preprocessor directive. Here's an example.

```
#define c 299792458  // speed of light
```

Predefined Macros in C

- __FILE__
  - This macro expands to the name of the current input file, in the form of a C string constant. This is the path by which the preprocessor opened the file, not the short name specified in ‘#include’ or as the input file name argument. For example, "/usr/local/include/myheader.h" is a possible expansion of this macro.
- __LINE__
  - This macro expands to the current input line number, in the form of a decimal integer constant. While we call it a predefined macro, it’s a pretty strange macro, since its “definition” changes with each new line of source code. __FILE__ and __LINE__ are useful in generating an error message to report an inconsistency detected by the program; the message can state the source line at which the inconsistency was detected.

- __DATE__
  - This macro expands to a string constant that describes the date on which the preprocessor is being run. The string constant contains eleven characters and looks like "Feb 12 1996". If the day of the month is less than 10, it is padded with a space on the left. If GCC cannot determine the current date, it will emit a warning message (once per compilation) and __DATE__ will expand to "??? ?? ????".
- __TIME__
  - This macro expands to a string constant that describes the time at which the preprocessor is being run. The string constant contains eight characters and looks like "23:59:01". If GCC cannot determine the current time, it will emit a warning message (once per compilation) and __TIME__ will expand to "??:??:??".

- __STDC__
  - In normal operation, this macro expands to the constant 1, to signify that this compiler conforms to ISO Standard C. If GNU CPP is used with a compiler other than GCC, this is not necessarily true; however, the preprocessor always conforms to the standard unless the -traditional-cpp option is used. This macro is not defined if the -traditional-cpp option is used. On some hosts, the system compiler uses a different convention, where __STDC__ is normally 0, but is 1 if the user specifies strict conformance to the C Standard.

Preprocessor Directives

- In almost every program we come across in C/C++, we see a few lines at the top of the program preceded by a hash (#) sign. These lines are preprocessed by the compiler before actual compilation begins. The end of these lines are identified by the newline character '\n' , no semicolon ';' is needed to terminate these lines. Preprocessor directives are mostly used in defining macros, evaluating conditional statements, source file inclusion, pragma directive, line control, error detection etc.

Conditional Compilation

- In C programming, you can instruct the preprocessor whether to include a block of code or not. To do so, conditional directives can be used. It's similar to a if statement with one major difference. The if statement is tested during the execution time to check whether a block of code should be executed or not whereas, the conditionals are used to include (or skip) a block of code in your program before execution.
- To use conditional, #ifdef, #if, #defined, #else and #elif directives are used.

Line Control

- Whenever we compile a program, there are chances of occurrence of some error in the program. Whenever compiler identifies error in the program it provides us with the filename in which error is found along with the list of lines and with the exact line numbers where the error is. This makes easy for us to find and rectify error. However we can control what information should the compiler provide during errors in compilation using the #line directive.

Line Control

Syntax: #line number "filename"

- number – line number that will be assigned to the next code line. The line numbers of successive lines will be increased one by one from this point on.
- “filename” – optional parameter that allows to redefine the file name that will be shown.

Error Directive

- This directive aborts the compilation process when it is found in the program during compilation and produces an error which is optional and can be specified as a parameter.

  Syntax: #error optional_error

- Here, optional_error is any error specified by the user which will be shown when this directive is found in the program.

THANK YOU

hitbullseye