Preprocessor in C ++

Preprocessors are directives, which provide instructions to the compiler to preprocess the information before starting the actual compilation.

Preprocessors are directives, which provide instructions to the compiler to preprocess the information before starting the actual compilation.

All preprocessor directives start with #, and only white-space spaces can appear before a preprocessor directive on a line. Preprocessor directives are not commands in C ++, so they do not end with a semicolon.

You have seen a preprocessor **directive** as **#include** in all examples. This macro is used to include a Header file in the source file.

There are some preprocessing directives supported by C ++ such as **#include, #define, #if, #else, #line** , . Below, we will show important preprocessor directives in C ++:

Define preprocessor in C ++

#Define preprocessor directive creates constant symbols. The constant symbol is a macro and the general pattern of this preprocessor directive in C ++ is:

```
#define  ten_cua_macro ten_thay_the
```

When this line appears in a file, all macros that appear later in this file will be replaced by ten_thay_the before the program is compiled. For example:

```
#include using namespace  std ; #define  PI  3.14159 int  main  () {  cout   "Gi
```

Suppose we have a source file, then compile it with the –E option and direct the result to test.p. Now, if you test the test.p, it will have a lot of information and at the bottom, you will refine the replaced value as follows:

```
$gcc  - E test . cpp  >  test . p  . int  main  () {  cout   "Gia tri cua PI la
```

Function-Like Macro in C ++

You can use the #define preprocessor directive in C ++ to define a macro that takes the following parameters:

```
#include using namespace  std ; #define  MIN ( a , b ) ((( a )( b )) ?  a  :  b
```

Compiling and executing the above code will produce the following results:

```
Gia  tri nho nhat la  16
```

Conditional compilation in C ++

There are a number of preprocessing directives that can be used to compile selections among parts of your source code. This process is called conditional compilation.

Preprocessing instructions are quite similar to the if selection structure. You consider the following code:

```
#ifndef  NULL  #define  NULL  0 #endif
```

You can compile a program for debugging purposes and can disable or enable this debug using a macro in C ++, as follows:

```
#ifdef  DEBUG cerr   "Bien x = "   x    endl ; #endif
```

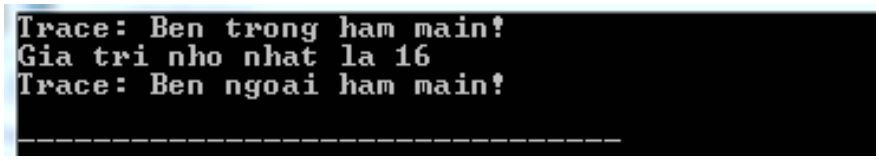
Cerr command to be compiled in the program if the constant DEBUG symbol has been defined before the #ifdef DEBUG directive. You can use the #if 0 command to annotate part of the program, as follows:

```
#if 0  khong duoc bien dich phan code nay  #endif
```

Try the following example:

```
#include using namespace  std ; #define  DEBUG  #define  MIN ( a , b ) ((( a )(
```

Compiling and running the above C ++ program will produce the following results:



The operators # and ## in C ++

The # and ## preprocessor operators are available in C ++ and ANSI / ISO C. The # operator announces that the replacement text will be converted to a string surrounded by quotation marks.

Consider the following macro definition:

```
#include using namespace  std ; #define QTM (  x  ) #x int  main  () {  cout   (
```

Compiling and running the above code will produce the following results:

```
Lap  trinh C ++
```

Now, we consider how it worked. It is simple to understand that the C ++ preprocessor converts the following line:

```
  cout   QTM ( Lap  trinh C ++)   endl ;
```

Line:

```
  cout   "Lap trinh C++"   endl ;
```

The ## operator is used to join a sequence of two tokens. For example:

```
 #define  CONCAT (  x ,  y  )  x  ## y
```

When CONCAT appears in the program, its parameters are concatenated and used to replace the macro. For example, CONCAT (HELLO, C ++) is replaced by "HELLO C ++" as in the following program:

```
 #include using namespace  std ; #define  concat ( a ,  b )  a  ## b int  main (
```

Compiling and executing the above code will result:

```
 Hello  C ++
```

We now consider how they worked. Simple is the following C ++ preprocessor:

```
  cout    concat ( x ,  y );
```

Line:

```
  cout    xy ;
```

Money macro defined in C ++

C ++ provides a number of pre-defined macros as listed below:

Macro Description__LINE__ Contains the current line number of the program when it is being compiled __FILE__ Contains the current file name of the program when it is being compiled __DATE__ Contains a month / day / year string as the date the source code is compiled __TIME__ Contains one string hour: minute: second is the time the program is compiled
Here is an example for all the above macros in C ++:

```
 #include using namespace  std ; int  main  () {  cout   "Gia tri cua __LINE__ la
```

Compile and run the above C ++ program to see the results:

According to Tutorialspoint

Previous article: Template in C ++

Next lesson: Signal Processing (Signal Handling) in C ++