C++ 模板 C++ 信号处理

C++ 预处理器

预处理器是一些指令，指示编译器在实际编译之前所需完成的预处理。

所有的预处理器指令都是以井号（#）开头，只有空格字符可以出现在预处理指令之前。预处理指令不是 C++ 语句，所以它们不会以分号（;）结尾。

我们已经看到，之前所有的实例中都有 **#include** 指令。这个宏用于把头文件包含到源文件中。

C++ 还支持很多预处理指令，比如 #include、#define、#if、#else、#line 等，让我们一起看看这些重要指令。

#define 预处理

#define 预处理指令用于创建符号常量。该符号常量通常称为**宏**，指令的一般形式是：

```
#define macro-name replacement-text
```

当这一行代码出现在一个文件中时，在该文件中后续出现的所有宏都将会在程序编译之前被替换为 replacement-text。例如：

```
#include <iostream>
using namespace std;
#define PI 3.14159
int main ()
{
cout << "Value of PI :" << PI << endl;
return 0;
}
```

现在，让我们测试这段代码，看看预处理的结果。假设源代码文件已经存在，接下来使用 -E 选项进行编译，并把结果重定向到 test.p。现在，如果您查看 test.p 文件，将会看到它已经包含大量的信息，而且在文件底部的值被改为如下：

```
$ gcc -E test.cpp > test.p

...
int main ()
{

    cout << "Value of PI :" << 3.14159 << endl;

    return 0;
}
```

参数宏

您可以使用 #define 来定义一个带有参数的宏，如下所示：

```
#include <iostream>
using namespace std;
#define MIN(a,b) (a<b ? a : b)
int main ()
{
int i, j;
i = 100;
j = 30;
cout <<"较小的值为: " << MIN(i, j) << endl;
return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
较小的值为: 30
```

条件编译

有几个指令可以用来有选择地对部分程序源代码进行编译。这个过程被称为条件编译。

条件预处理器的结构与 if 选择结构很像。请看下面这段预处理器的代码：

```
#ifdef NULL
   #define NULL 0
#endif
```

您可以只在调试时进行编译，调试开关可以使用一个宏来实现，如下所示：

```
#ifdef DEBUG
   cerr <<"Variable x = " << x << endl;
#endif
```

如果在指令 #ifdef DEBUG 之前已经定义了符号常量 DEBUG，则会对程序中的 **cerr** 语句进行编译。您可以使用 #if 0 语句注释掉程序的一部分，如下所示：

```
#if 0
   不进行编译的代码
#endif
```

让我们尝试下面的实例：

实例

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

```
int main ()
{
int i, j;
i = 100;
j = 30;
#ifdef DEBUG
cerr <<"Trace: Inside main function" << endl;
#endif
#if 0
/* 这是注释部分 */
cout << MKSTR(HELLO C++) << endl;
#endif
cout <<"The minimum is " << MIN(i, j) << endl;
#ifdef DEBUG
cerr <<"Trace: Coming out of main function" << endl;
#endif
return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Trace: Inside main function
The minimum is 30
Trace: Coming out of main function
```

和 ## 运算符

和 ## 预处理运算符在 C++ 和 ANSI/ISO C 中都是可用的。# 运算符会把 replacement-text 令牌转换为用引号引起来的字符串。

请看下面的宏定义：

实例

```
#include <iostream>
using namespace std;
#define MKSTR( x ) #x
int main ()
{
cout << MKSTR(HELLO C++) << endl;
return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
HELLO C++
```

让我们来看看它是如何工作的。不难理解，C++ 预处理器把下面这行：

```
cout << MKSTR(HELLO C++) << endl;
```

转换成了：

```
cout << "HELLO C++" << endl;
```

运算符用于连接两个令牌。下面是一个实例：

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

当 CONCAT 出现在程序中时，它的参数会被连接起来，并用来取代宏。例如，程序中 CONCAT(HELLO, C++) 会被替换为 "HELLO C++"，如下面实例所示。

实例

```
#include <iostream>
using namespace std;
#define concat(a, b) a ## b
int main()
{
int xy = 100;
cout << concat(x, y);
return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
100
```

让我们来看看它是如何工作的。不难理解，C++ 预处理器把下面这行：

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

转换成了：

```
cout << xy;
```

C++ 中的预定义宏

C++ 提供了下表所示的一些预定义宏：

宏	描述
__LINE__	这会在程序编译时包含当前行号。
__FILE__	这会在程序编译时包含当前文件名。
__DATE__	这会包含一个形式为 month/day/year 的字符串，它表示把源文件转换为目标代码的日

	期。
__TIME__	这会包含一个形式为 hour:minute:second 的字符串，它表示程序被编译的时间。

让我们看看上述这些宏的实例：

实例

```
#include <iostream>
using namespace std;
int main ()
{
cout << "Value of __LINE__ : " << __LINE__ << endl;
cout << "Value of __FILE__ : " << __FILE__ << endl;
cout << "Value of __DATE__ : " << __DATE__ << endl;
cout << "Value of __TIME__ : " << __TIME__ << endl;
return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Value of __LINE__ : 6
Value of __FILE__ : test.cpp
Value of __DATE__ : Feb 28 2011
Value of __TIME__ : 18:52:48
```

← C++ 模板 C++ 信号处理 →