Директивы препроцессора

Директивы препроцессора-это строки, включенные в код программ, которым предшествует знак хэша (#). Эти строки не являются программными операторами, а директивами для *препроцессора*. Препроцессор проверяет код до начала фактической компиляции кода и разрешает все эти директивы до того, как любой код фактически генерируется обычными операторами.

Эти *директивы препроцессора* распространяются только на одну строку кода. Как только символ новой строки найден, директива препроцессора заканчивается. Нет точки с запятой (;) ожидается в конце директивы препроцессора. Единственный способ, которым директива препроцессора может распространяться более чем на одну строку, - это предшествовать символу новой строки в конце строки обратной косой чертой (\).

определения макросов (#define, #undef)

Для определения макросов препроцессора мы можем использовать #define. Его синтаксис:

```
#define identifier replacement
```

Когда препроцессор встречает эту директиву, он заменяет любое вхождение identifierв остальной части кода на replacement. Это replacementможет быть выражение, оператор, блок или просто что угодно. Препроцессор не понимает собственно C++, он просто заменяет любое вхождение identifierby replacement.

```
#define TABLE_SIZE 100
int table1[TABLE_SIZE];
int table2[TABLE_SIZE];
```

После замены препроцессора TABLE_SIZEкод становится эквивалентным:

```
int table1[100];
int table2[100];
```

#define может работать также с параметрами для определения макросов функций:

```
#define getmax(a,b) a>b?a:b
```

Это заменит любое вхождениеgetmax, за которым следуют два аргумента, выражением замены, но также заменит каждый аргумент его идентификатором, точно так же, как вы ожидали бы, если бы это была функция:

```
// function macro
#include <iostream>
using namespace std;

#define getmax(a,b) ((a)>(b)?(a):(b))

int main()
{
  int x=5, y;
  y= getmax(x,2);
  cout << y << endl;
  cout << getmax(7,x) << endl;
  return 0;
}
```

```
5
7
```

Редактировать и запускать

Определенные макросы не зависят от блочной структуры. Макрос длится до тех пор, пока он не будет определен директивой #undefпрепроцессора:

```
#define TABLE_SIZE 100
int table1[TABLE_SIZE];
#undef TABLE_SIZE
#define TABLE_SIZE 200
int table2[TABLE_SIZE];
```

Это будет генерировать тот же код, что и:

```
int table1[100];
int table2[200];
```

Определения макросов функций принимают два специальных оператора (# и ##) в последовательности замены:
Оператор #, за которым следует имя параметра, заменяется строковым литералом, содержащим переданный аргумент (как если бы он был заключен в двойные кавычки):

```
#define str(x) #x
cout << str(test);
```

Это будет переведено на:

```
cout << "test";
```

Оператор ##объединяет два аргумента, не оставляя пробелов между ними:

```
#define glue(a,b) a ## b
glue(c,out) << "test";
```

Это также будет переведено на:

```
cout << "test";
```

Поскольку замена препроцессора происходит до любой проверки синтаксиса C++, определения макросов могут быть сложной функцией. Но будьте осторожны: код, который сильно зависит от сложных макросов, становится менее читаемым, поскольку ожидаемый синтаксис во многих случаях отличается от обычных выражений, ожидаемых программистами в C++.

Условные включения (#ifdef, #ifndef, #if, #endif, #else и #elif)

Эти директивы позволяют включать или отбрасывать часть кода программы, если выполняется определенное условие.

#ifdef позволяет компилировать раздел программы только в том случае, если определен макрос, указанный в качестве параметра, независимо от его значения. Например:

```
#ifdef TABLE_SIZE
int table[TABLE_SIZE];
#endif
```

В этом случае строка кода int table[TABLE_SIZE];компилируется только в том случае , если TABLE_SIZEранее была определена с #defineпомощью, независимо от ее значения. Если она не была определена, эта строка не будет включена в компиляцию программы.

#ifndefслужит для полной противоположности: код между #ifndef#endifдирективами и компилируется только в том случае, если указанный идентификатор ранее не был определен. Например:

```
#ifndef TABLE_SIZE
#define TABLE_SIZE 100
#endif
int table[TABLE_SIZE];
```

В этом случае, если при получении этого фрагмента кода TABLE_SIZEмакрос еще не определен, он будет определен со значением 100. Если бы он уже существовал, он сохранил бы свое предыдущее значение, так #defineкак директива не будет выполнена.

Директивы #if, #elseи #elif(т. е. "else if") служат для указания некоторого условия, которое должно быть выполнено для компиляции части кода, которую они окружают. Условие, которое следует #ifили #elifможет вычислять только постоянные выражения, включая выражения макросов. Например:

```
#if TABLE_SIZE>200
#undef TABLE_SIZE
#define TABLE_SIZE 200

#elif TABLE_SIZE<50
#undef TABLE_SIZE
#define TABLE_SIZE 50

#else
#undef TABLE_SIZE
#define TABLE_SIZE 100
#endif

int table[TABLE_SIZE];
```

Обратите внимание , как вся структура #if#elifи #elseцепные директивы заканчиваются #endif.

Поведение #ifdefand #ifndefтакже может быть достигнуто с помощью специальных операторов definedи !definedсоответственно в любой #if#elifдирективе or:

```
#if defined ARRAY_SIZE
#define TABLE_SIZE ARRAY_SIZE
#elif !defined BUFFER_SIZE
#define TABLE_SIZE 128
#else
#define TABLE_SIZE BUFFER_SIZE
#endif
```

Управление строкой (#line)

Когда мы компилируем программу и в процессе компиляции происходит какая-то ошибка, компилятор показывает сообщение об ошибке со ссылками на имя файла, в котором произошла ошибка, и номер строки, поэтому легче найти код, генерирующий ошибку.

#lineДирективa позволяет нам контролировать обе вещи, номера строк в файлах кода, а также имя файла, которое мы хотим, которое появляется при возникновении ошибки. Его формат:

```
#line number "filename"
```

Где numberновый номер строки, который будет назначен следующей строке кода. С этого момента количество последовательных строк будет увеличиваться одна за другой.

"filename" это необязательный параметр, который позволяет переопределить имя файла, которое будет отображаться. Например:

```
#line 20 "assigning variable"
int a?;
```

Этот код будет генерировать ошибку , которая будет отображаться как ошибка в файле"assigning variable", строка 20.

Директива об ошибках (#error)

Эта директива прерывает процесс компиляции, когда она найдена, генерируя ошибку компиляции, которая может быть указана в качестве ее параметра:

```
#ifndef __cplusplus
#error A C++ compiler is required!
#endif
```

Этот пример прерывает процесс компиляции, если имя макроса __cplusplusне определено (это имя макроса определено по умолчанию во всех компиляторах C++).

Включение исходного файла (#include)

Эта директива усердно использовалась в других разделах этого руководства. Когда препроцессор находит #includeдирективу, он заменяет ее всем содержимым указанного заголовка или файла. Существует два способа использования#include:

```
#include <header>
#include "file"
```

В первом случае *заголовок* указывается между угловыми скобками <>. Он используется для включения заголовков, предоставляемых реализацией, таких как заголовки, составляющие стандартную библиотеку (iostream, string,...). Являются ли заголовки фактически файлами или существуют в какой-либо другой форме, *определяется реализацией*, но в любом случае они должны быть должным образом включены в эту директиву.

Синтаксис, используемый во втором#include, использует кавычки и включает *файл*. *Файл* ищется в определенной *реализацией* манера, которая обычно включает текущий путь. В случае, если файл не найден, компилятор интерпретирует директиву как включение заголовка, как если бы кавычки ("") были заменены угловыми скобками (<>).

Директива Pragma (#pragma)

Эта директива используется для указания различных параметров компилятору. Эти параметры специфичны для платформы и используемого компилятора. Обратитесь к руководству или ссылке вашего компилятора для получения дополнительной информации о возможных параметрах , которые вы можете определить #pragma.

Если компилятор не поддерживает конкретный аргумент#pragma, он игнорируется - синтаксическая ошибка не генерируется.

Предопределенные имена макросов

Следующие имена макросов всегда определены (все они начинаются и заканчиваются двумя символами подчеркивания, _):

макрос	значение
__LINE__	Целочисленное значение, представляющее текущую строку в компилируемом файле исходного кода.
__FILE__	Строковый литерал, содержащий предполагаемое имя исходного компилируемого файла.
__DATE__	Строковый литерал в виде "Ммм дд гггг", содержащий дату начала процесса компиляции.
__TIME__	Строковый литерал в виде "hh:mm:ss", содержащий время начала процесса компиляции.
__cplusplus	Целочисленное значение. Все компиляторы C++ имеют эту константу, определенную в некотором значении. Его значение зависит от версии стандарта, поддерживаемого компилятором: • **199711L**: ISO C++ 1998/2003 • **201103L**: ISO C++ 2011 Несоответствующие компиляторы определяют эту константу как некоторое значение длиной не более пяти цифр. Обратите внимание, что многие компиляторы не полностью соответствуют друг другу и поэтому эта константа не будет определена ни как одно из приведенных выше значений.
__STDC_HOSTED__	1 если реализация является *размещенной реализацией* (со всеми доступными стандартными заголовками) 0, в противном случае.

Следующие макросы опционально определяются, как правило, в зависимости от того, доступна ли функция:

макрос	значение
__STDC__	В C: если определено 1, реализация соответствует стандарту C. В C++: определена реализация.
__STDC_VERSION__	В C: • **199401L**: ISO C 1990, Ammendment 1 • **199901L**: ISO C 1999 • **201112L**: ISO C 2011 В C++: реализация определена.
__STDC_MB_MIGHT_NEQ_WC__	1 если многобайтовое кодирование может дать символу другое значение в символьных литералах
__STDC_ISO_10646__	Значение в форме yyyymmL, указывающее дату стандарта Unicode с последующей кодировкой wchar_tсимволов
__STDCPP_STRICT_POINTER_SAFETY__	1 если реализация имеет *строгую безопасность указателя* (см. get_pointer_safety)
__STDCPP_THREADS__	1 если программа может иметь более одного потока

Конкретные реализации могут определять дополнительные константы.

Например:

```
// standard macro names
#include <iostream>
using namespace std;

int main()
{
  cout << "This is the line number " << __LINE__;
  cout << " of file " << __FILE__ << ".\n";
  cout << "Its compilation began " << __DATE__;
  cout << " at " << __TIME__ << ".\n";
  cout << "The compiler gives a __cplusplus value of " << __cplusplus;
  return 0;
}
```

```
This is the line number 7 of file /home/
Its compilation began Nov  1 2005 at 10
The compiler gives a __cplusplus value o
```

Предыдущий: **Исключения** Указатель Далее: **ввод/вывод с файлами**