Модульное программирование

Шаблоны, директивы препроцессора, пространства имен

Шаблоны, директивы препроцессора, пространства имен

Директивы препроцессора

Препроцессором называется первая фаза компиляции. Инструкции (команды) препроцессора называются директивами. Они должны начинаться с символа ‘#’.

Директива include

Директива include<имя_файла> осуществляет подстановку указанного файла в точку, где она записана.

Шаблоны, директивы препроцессора, пространства имен

Заголовочные файлы обычно имеют расширение h и могут содержать:

- определение типов, встроенных функций, шаблоны, перечисления;
- объявления (прототипы) функций, данных, имен, шаблонов;
- пространства имен;
- директивы препроцессора;
- комментарии.

Шаблоны, директивы препроцессора, пространства имен

В заголовочных файлах не должно быть определение функций и данных. Их принято выносить в файлы реализации. Это не требования языка, это рекомендация.

При указании имен файлов стандартной библиотеки расширение можно опускать. Для большинства старых версий файлов, заимствованных от языка C, в языке C++ есть аналоги файлов без расширения, например,

stdlib.h и cstdlib, stdio.h и cstdio, и т.д.

Шаблоны, директивы препроцессора, пространства имен

#include”file_name.h”
file_name_main.cpp
Файл основной программы, содержащий функцию main

file_name.h
Заголовочный файл(ы)

file_name_1.cpp
file_name_2.cpp
…….
file_name_n.cpp
Файлы реализации

Шаблоны, директивы препроцессора, пространства имен

***Директива #define**

Директива define определяет подстановку в тексте программы. Она используется для определения:

- символических констант:

#define имя текст_подстановки,

Например,

#define PI 3.14

В любом контексте символьная константа PI будет интерпретироваться как число 3.14.

Шаблоны, директивы препроцессора, пространства имен

- макросов, которые выглядят как функции, но реализуются подстановкой из текста в текст программы:

  #define имя(параметры) текст_подсановки

Например,

  #define sqr(x) (x*x)

Использование макросов вносит свои сложности в программы, в частности, особенности передачи аргументов.

Шаблоны, директивы препроцессора, пространства имен

Например, для описанного макроса вызов

```
cout << sqr(y+1) << endl;
```

приведет к получению числа 6, для правильного ответа нужно вызвать следующим образом

```
cout << sqr((y+1)) << endl;
```

Макросы и символические константы заимствованы из языка С, в С++ они не получили широкого применения.

Шаблоны, директивы препроцессора, пространства имен

- символов, управляющих условной трансляцией. Они используются совместно с директивами #ifdef и #ifndef.

Общий формат:

#define имя

Например,

```
#define VERSION 1
#define h_file “head_file.h”
```

Шаблоны, директивы препроцессора, пространства имен

```
#define A

int main()
{
    #ifdef A
     cout << "A" << endl;
    #else
     cout << "B" << endl;
    #endif
    return 0;
}
```

Шаблоны, директивы препроцессора, пространства имен

Имена, объявляемые через директиву define рекомендуется писать прописными символами, чтобы зрительно отличать их от других программных объектов (переменных, функций).

Шаблоны, директивы препроцессора, пространства имен

Директивы условной трансляции

Директивы условной трансляции #if, #ifdef, #ifndef применяются для того, чтобы исключить компиляцию отдельных частей программы. Это бывает полезно при отладке или при поддержке нескольких версий программ для различных платформ.

Шаблоны, директивы препроцессора, пространства имен

Формат директивы #if:

```
   #if константное_выражение
.....
    [ #elif константное_выражение ]
.....
    [ #elif константное_выражение ]
.....
    [ #else ]
   #endif
```

Шаблоны, директивы препроцессора, пространства имен

Исключаемые блоки могут содержать как описания, так и исполняемые операторы. Пример условно исключения различных версий заголовочного файла:

```
#ifdef VERSION == 1
  #define INCLFILE “vers_1.h”
#elif VERSION == 2
  #define INCLFILE “vers_2.h”
#else
  #define INCLFILE “vers_N.h”
#endif
#include INCFILE
```

Шаблоны, директивы препроцессора, пространства имен

В константных выражениях может использоваться проверка, определена ли константа с помощью директивы define, например:

```
#if defined(__BORLANDC__) &&
__BORLAND__ == 0530       // BC5.3
typedef istream_iterator<int, ptrdiff_t> istream_it;
#else
typedef istream_iterator<int> istream_iter;
```

Шаблоны, директивы препроцессора, пространства имен

И еще одно применение директив условной трансляции – временное комментирование фрагмента кода. Иногда используется в целях отладки.

Предопределенные макросы

В C++ определено некоторое количество макросов , предназначенных в основном для того, чтобы выдавать информацию о версии программы или месте возникновения ошибки.

Шаблоны, директивы препроцессора, пространства имен

Например, макрос __splusplus определен, если программа компилируется в среде C++.

```
#ifdef __cplusplus
   cout << " C++ " << endl;
   #else  cout << " no C++ " << endl;
#endif
```

Это макрос использовался в период перехода от C к C++.

Шаблоны, директивы препроцессора, пространства имен

Другие макросы:

- __DATE__ - содержит строку с текущей датой (месяц, день, год), например,

```
cout << __DATE__ << endl;
```

- __FILE__ - содержит строку с полным именем текущего файла, например,

```
cout << __FILE__ << endl;
```

Шаблоны, директивы препроцессора, пространства имен

- __LINE__ - текущая строка исходного текста;
- __TIME__ - текущее время.

Шаблоны, директивы препроцессора, пространства имен

Области действия и пространства имен

Каждый программный объект имеет область действия и время жизни, которые определяются видом и местом его определения. Существуют следующие разновидности областей действия:

- блок;
- прототип функции;

Шаблоны, директивы препроцессора, пространства имен

- функция;
- файл;
- группа файлов, в пределе включающая все файлы программного проекта (глобальная область действия);
- класс;
- пространство имен (часть глобальной области).

Шаблоны, директивы препроцессора, пространства имен

Блок. Программный объект, определенный внутри блока, по области действия является локальным. Область действия такого объекта начинается в точке его объявления и заканчивается в конце блока. Класс памяти такого объекта считается auto, при выходе из блока, память из под освобождается.

Шаблоны, директивы препроцессора, пространства имен

Если же программный объект определен внутри блока как static, то время жизни его максимально и совпадает со временем выполнения программы.

Рассмотрим пример:

```
int i = 20;
int main()
{
  int i = 10;
    {    // блок
         // int i = 30;      и это тоже можно
         cout << "i_local:" << i << endl;
         cout << "i_global:" << ::i << endl;
    }
return 0;
}
```

Шаблоны, директивы препроцессора, пространства имен

В глобальной области объявить блок таким образом нельзя.

Прототип функции. Идентификаторы, указанные в списке параметров прототипа функции, имеют областью действия только прототип функции. Поэтому их можно опускать при описании.

Шаблоны, директивы препроцессора, пространства имен

Функция. Программные объекты, определенные в блоке функции, имеют область действия и время жизни точно такие же как в обычном блоке.

Параметры функции, передаваемые по значению, имеют областью действия всю функцию, а время жизни – время работы функции.

Параметры функции, передаваемые по ссылке, имеют область действия и время жизни, определяемое соответствующими аргументами в вызове функции.

Шаблоны, директивы препроцессора, пространства имен

Локальные объекты, объявленные в теле функции, действуют в пределах конкретного блока. Время жизни – время работы функции. В период работы функции эти объекты хранятся в программном стеке. От вызова к вызову их значения не сохраняются.

Если есть необходимость сохранить значение локальных объектов, их необходимо объявить с модификатором класса памяти static.

Шаблоны, директивы препроцессора, пространства имен

В этом случае переменные будут храниться в сегменте данных программы и время их жизни совпадает со временем работы программы.

Пример:

```
double func(double d)
{
    static double temp = 3.5;
    cout << " Temp: " << temp++ << endl;
    return d*temp;
}
```

Переменная static double temp будет сохраняться от одного вызова функции к другому.

Шаблоны, директивы препроцессора, пространства имен

Файл. Программный объект, определенный с описателем класса static вне любого блока, функции, класса, имеет областью действия, начинающуюся в точке его объявления и заканчивается в конце файла. В область действия попадают вложенные блоки.

Если во внутреннем блоке определен объект с таким же именем, тогда внешний объект становится невидимым.

Шаблоны, директивы препроцессора, пространства имен

Но обратиться к нему можно через оператор разрешения области видимости -::.

Класс. Компоненты класса (поля, методы), за исключением статических, имеют областью действия класс. Время жизни компонентов класса определяется промежутком времени от создания объекта до его разрушения.

Шаблоны, директивы препроцессора, пространства имен

Пространства имен (именованные области).

В C++ есть возможность явным образом задать область действия имен как часть глобальной области с помощью оператора namespace.

Шаблоны, директивы препроцессора, пространства имен

Пространства имен

Пространство имен (именованная область) служит для логического группирования определений, объявлений и ограничения доступа к ним. Чем больше объем программы, тем актуальнее использование именованных областей. Их удобно использовать в больших программных проектах.

Шаблоны, директивы препроцессора, пространства имен

Общий формат объявления именованной области следующий:

```
namespace [имя_области]
 { // определения и объявления }
```

Одно и то же пространство имен может объявляться многократно, причем все последующие будут пониматься как продолжения предыдущих.

Шаблоны, директивы препроцессора, пространства имен

Продолжение именованных областей можно делать в различных файлах.

Рассмотрим простой пример:

```
namespace demo
{
    int I = 1;   //    определение переменной
    int k = 0;  //    определение переменной
    //      прототип функции
    void fun_1(int);
    //      определение функции
    int fun_2(int I, int j)
    {
    //
    }
}
```

Шаблоны, директивы препроцессора, пространства имен

Дальнейшее расширение пространства

```
namespace demo
{
   // int i = 2; ошибка, повторное объявление
   void func_1(double);
}
```

Если имя области не задано (анонимная область), компилятор определяет его самостоятельно с помощью уникального идентификатора.

Шаблоны, директивы препроцессора, пространства имен

Нельзя получить доступ из именованной области одного файла к неименованной области другого файла.

В именованной области логичнее всего помещать объявления объектов, а их определения выносить в файлы реализации, например,

```
void demo:: func_1(double d)
{
   //   тело функции
}
```

Шаблоны, директивы препроцессора, пространства имен

Такой прием обеспечивает разделение интерфейса и реализации.

Объекты программы, определенные внутри пространства имен, становятся доступными с момента объявления пространства. Обратиться к ним можно с помощью имени области и оператора доступа к области видимости, например, `demo:: i == 100;`

Шаблоны, директивы препроцессора, пространства имен

Если какое-либо имя из именованной области используется часто, его можно сделать доступным с помощью оператора using, например, using demo:: I;.

После чего к нему можно обращаться без указания области видимости.

Шаблоны, директивы препроцессора, пространства имен

Если требуется открыть всю область видимости, используется оператор

using namespace demo;

Вспомните, например,

using namespace std;

Шаблоны, директивы препроцессора, пространства имен

Именованные области могут быть вложены друг в друга.

Шаблоны, директивы препроцессора, пространства имен

Шаблоны, директивы препроцессора, пространства имен

Шаблоны, директивы препроцессора, пространства имен

Шаблоны, директивы препроцессора, пространства имен

Шаблоны, директивы препроцессора, пространства имен

Шаблоны, директивы препроцессора, пространства имен

Шаблоны, директивы препроцессора, пространства имен

Шаблоны, директивы препроцессора, пространства имен

Шаблоны, директивы препроцессора, пространства имен

Шаблоны, директивы препроцессора, пространства имен

Шаблоны, директивы препроцессора, пространства имен