Пространство имён

Пространства имен предназначены для локализации имен идентификаторов и предотвращения их конфликтов.

Среда программирования C++ работает с большим количеством переменных, функций и классов. Раньше все их имена пребывали в глобальном пространстве и нередко конфликтовали между собой. Чаше всего конфликты имен возникали, когда программа использовала несколько сторонних библиотек одновременно. Особенно это касается имен классов.

Введение ключевого слова namespace позволило решить эти проблемы.

Поскольку пространство имен позволяет локализовать область видимости объектов, объявленных внутри него, одно и то же имя, упомянутое в разных контекстах, больше не вызывает конфликтов. Наибольшую пользу это нововведение принесло стандартной библиотеке языка C++. До сих пор вся стандартная библиотека языка C++ находилась в глобальном пространстве имен (которое, собственно говоря, было единственным). Теперь стандартная библиотека определена внутри своего собственного пространства имен std, что намного уменьшает вероятность конфликтов.

Программист может создавать свои собственные пространства имен и самостоятельно локализовать имена, которые могут вызывать конфликты. Это особенно важно при разработке классов или библиотек функций.

Пространство имён

Ключевое слово namespace позволяет разделить глобальное пространство имен на декларативные области (declarative region). Пространство имен – это область видимости.

Общий вид объявления пространства имен таков:

```
namespace <имя_пространства_имён>
  {
   // Объявления
     }
```

Все, что объявлено в разделе namespace, находится внутри области видимости этого пространства имен.

Пространство имен должно объявляться вне всех остальных областей видимости за исключением того, что одно пространство имен может быть вложено в другое.
То есть вложенным пространство имен может быть только в другое пространство имен, но не в какую бы то ни было иную область видимости.
Это означает, что нельзя объявлять пространства имен, например, внутри функции.

Пространство имён

Рассмотрим пример пространства имен CounterNameSpace:

В нём локализуются имена, использованные при создании простого класса, реализующего обратный счетчик В этом пространстве имен определен класс counter и переменные upperbound и lowerbound, содержащие верхнюю и нижнюю границу диапазона счетчика.

```
namespace CounterNameSpace
 {
  int upperbound;
  int lowerbound;
 class counter
  {   int count;
   public:
     counter(int n)
     {  if(n <= upperbound)
           count = n;
        else
           count = upperbound;
     }
     void reset(int n)
      { if(n <= upperbound) count = n; }
     int run()
           { if(count > lowerbound)
                 return count--;
              else
                 return lowerbound;
           }
  };
}
```

Переменные upperbound и lowerbound, а также класс counter находятся в области видимости, определенной пространством имен CounterNameSpace.

К идентификаторам, объявленным внутри пространства имен, можно обращаться непосредственно, не указывая квалификатор.

Например, внутри пространства имен CounterNameSpace функция run() может прямо ссылаться на переменную lowerbound:

```
if (count > lowerbound)  return count--;
```

Однако, для ссылок на объекты, находящиеся вне этого пространства имён, надо применять оператор разрешения области видимости.

Например, чтобы присвоить число 10 переменной upperbound в модуле, находящемся вне пространства имен CounterNameSpace, следует выполнить оператор:

```
CounterNameSpace::upperbound = 10;
```

Как правило, чтобы обратиться к элементу пространства имен извне, следует перед его именем указать имя пространства и оператор разрешения области видимости.

Пространство имён

Пример применения пространства имен CounterNameSpace:

```
#include <iostream>
  using namespace std;
namespace CounterNameSpace
 {
  int upperbound;
  int lowerbound;
 class counter
  {   int count;
   public:
     counter(int n)
     {  if(n <= upperbound)
           count = n;
         else
           count = upperbound;
      }
      void reset(int n)
       { if(n <= upperbound) count = n; }
      int run()
           { if(count > lowerbound)
                 return count--;
               else
                 return lowerbound;
            }
   };
}
```

см. продолжение

продолжение

```
int main()
 {
  CounterNameSpace::upperbound = 100;
  CounterNameSpace::lowerbound = 0;
  CounterNameSpace::counter ob1(10);
   int i;
do
   {  i = ob1.run();     cout << i << "   " ; }
while(i>CounterNameSpace::lowerbound);
cout << endl;
   CounterNameSpace::counter ob2(20);
do
   {  i = ob2.run();   cout << i <<  "    " ; }
while(i > CounterNameSpace::lowerbound);
cout << endl;
ob2.reset(100);
  CounterNameSpace::lowerbound = 90;
do
   {  i = ob2.run();  cout << i << "   "; }
while(i> CounterNameSpace::lowerbound);
return 0;
}
```

Объявлению объекта класса и ссылкам на переменные предшествует квалификатор CounterNameSpace. НО после объявления объекта класса его члены можно использовать без квалификатора. Т.о. функцию ob1.run() можно вызывать непосредственно, поскольку соответствующее пространство имен подразумевается.

Пространство имён

Директива using

Текст программы, в которой часто встречаются ссылки на элементы пространства имен станет малопонятным, поскольку будет переполнен квалификаторами и операторами области разрешения видимости.
Чтобы избежать этого, следует применять директиву using:

using namespace <имя_пространства_имён>;
using <имя_простр_имён> :: <член_простр_имён>;

В первом варианте параметр <name> задает название пространства имен. Все элементы этого пространства становятся частью текущей области видимости и могут использоваться без указания квалификатора.
Во втором варианте в область видимости включается только конкретный элемент пространства имен.

Если в программе, например, объявлено пространство имен CounterNameSpace, то можно применить следующие операторы:

```
using CounterNameSpace::lowerbound;   // Видимым является
                                      // только член lowerbound.
lowerbound =10;                       // Правильный оператор, поскольку переменная
                                      // lowerbound является видимой.
using namespace CounterNameSpace;     //Все члены видимы.
upperbound = 100;                     // Правильный оператор, поскольку все члены
                                      // являются видимыми.
```

Пространство имён

Директива using

Программа представляет собой переработанный вариант предыдущей программы, иллюстрирующий применение оператора using.

```
#include <iostream>
using namespace std;
 namespace CounterNameSpace
 {
  int upperbound;
  int lowerbound;
 class counter
  { int count;
 public:
  counter(int n)
   { if(n <= upperbound)  count = n;
        else count = upperbound;  }
  void reset(int n)
    { if(n <= upperbound)  count = n; }
  int run()
    { if(count > lowerbound)
          return count--;
        else return lowerbound; }
   };
 }
```

см. продолжение

Иллюстрируется еще одно полезное свойство: одно пространство имен не перекрывает другого. Просто добавляются новые члены к элементам текущей области видимости. Т.о., в конце программы, кроме глобального пространства имен, доступными оказываются также пространства имен std и CounterNameSpace.

```
int main()
 {   // Используется только член upperbound
     // из     пространства имен CounterNameSpace.
  using CounterNameSpace::upperbound;
     // Теперь квалификатор перед переменной
     // upperbound не нужен.
   upperbound = 100;
     // Перед переменной lowerbound квалификатор
     // нужен.
  CounterNameSpace::lowerbound = 0;
  CounterNameSpace::counter obi(10);
    int i ;
 do { i = obi.run(); cout << i >> "  "; }
 while(i>CounterNameSpace::lowerbound);
cout << endl;
   // Теперь доступно все пространство имен
   // CounterNameSpace.
   using namespace CounterNameSpace;
  counter" ob2 (20) ;
 do { i = ob2.run (); cout << i << "  "; }
 while(i > lowerbound);
cout << endl;
ob2.reset(100); lowerbound = 90;
  do { i = ob2.run(); cout << i <<  " "; }
  while(i > lowerbound);
return 0;
}
```

продолжение

Пространство имён

Неименованные пространства имен

Существует особая разновидность пространств имен – **неименованное пространство имен (unnamed namespace)**, позволяющее создавать уникальные идентификаторы, область видимости которых ограничена файлом. Неименованные пространства имен иногда называют **безымянными (anonymous namespaces)**.

Их объявление имеет следующий вид:

```
namespace
  {
   //  Объявления
     }
```

Неименованные пространства имен позволяют создавать уникальные идентификаторы, область видимости которых ограничена файлом. Иначе говоря, внутри файла, содержащего неименованное пространство имен, элементы этого пространства можно использовать без квалификаторов. Вне файла эти переменные считаются невидимыми.

Неименованные пространства имен позволяют избежать применения спецификатора static. Спецификатор static позволяет сузить область видимости глобального имени до размера файла.

Пространство имён

Особенности пространства имен

Одно и то же пространство имен можно объявлять несколько раз. Это позволяет распределить одно пространство имен среди нескольких файлов и даже разделить его внутри одного файла.

```
#include <iostream>
using namespace std;
  namespace NS
    { int i ; }
//  . . .
   namespace NS
     { int j ;  }
int main()
 {
   NS::i = NS::j = 10;
//  Конкретная ссылка на пространство имен NS.
 cout << NS::i * NS::j <<  "\n";
// Применение пространства NS.
using namespace NS;
 cout << i * j;
 return 0;
}
```

Результаты работы этой программы:
100
100

В данном примере пространство имен NS разделено на две части. Однако содержимое каждой части по-прежнему находится в одном и том же пространстве имен NS.

Пространство имен должно быть объявлено вне какой бы то ни было области видимости. Это значит, что пространство имен нельзя объявлять, например, внутри функции.

Пространство имён

Особенности пространства имен

Несмотря на то, что пространство имен нельзя объявлять внутри функции, существует однако одно исключение: пространства имен могут быть вложенными. Рассмотрим следующую программу:

```
#include <iostream>
using namespace std;
namespace NS1
    { int i ;
      namespace NS2
        { // Вложенное пространство имен
            int j ;  }
    }
int main()
 { NS1::i = 19;
    // NS2::j  = 10; Ошибка,   пространство имен NS2
    // находится вне области видимости.
  NSl::NS2::j  = 10;   // Это правильно!
  cout << NSl::i <<  "   " <<    NSl::NS2::j << "\n";
   // Использование пространства имен NS1
using namespace NS1;
      /* Теперь,  поскольку пространство имен NS1 находится в
области видимости, для ссылки на переменную j можно
использовать пространство имен NS2. */
  cout <<  i * NS2::j;
  return 0;
 }
```

Результаты работы этой программы:
19 10
190

В данном примере пространство имен NS2 вложено в пространство имен NS1. Следовательно, в начале программы при ссылке на переменную j нужно указывать названия обоих пространств имен, которым она принадлежит. Одного имени NS2 недостаточно.

После выполнения директивы:

using namespace NS1;

можно указывать только имя NS2, поскольку пространство имен NS1 уже находится в области видимости.

Пространство имён

Пространства имен std

Стандартная библиотека языка C++ пребывает в собственном пространстве имен std. По этой причине большинство программ, приведенных в лекциях, содержат директиву

```
using namespace std;
```

Этот оператор открывает прямой доступ к функциям и классам, определенным внутри библиотеки, поэтому квалификатор std:: не нужен.

При желании можно явно указывать квалификатор std::
Например, в следующей программе стандартная библиотека не включается в глобальное пространство имен:

```
// Применение явного квалификатора std::
#include <iostream>
int main()
{   int val;
    std::cout << "Введите номер: ";
    std::cin >> val;
    std::cout <<  "Это ваш номер: ";
    std::cout << std::hex << val;
  return 0;
}
```

Здесь потоки cout, cin и манипулятор hex сопровождаются указанием квалификатора std::. Т.е., при записи в стандартный поток вывода нужно задавать имя std: :cout, при чтении из стандартного потока ввода следует задавать имя std: :cin, а манипулятор формата необходимо вызывать по имени std: :hex.

Пространство имён

Пространства имен std

Если программа не очень широко использует средства стандартной библиотеки, ее можно не включать в глобальное пространство имен. Но, если программа содержит сотни обращений к стандартным функциям и классам, будет слишком утомительно каждый раз указывать квалификатор std::.

Если в программе используется лишь несколько членов стандартной библиотеки, имеет смысл применить директиву using к каждому из них отдельно. Эти имена по-прежнему можно использовать, не указывая спецификатор std::, однако при этом вся стандартная библиотека не погружается в глобальное пространство имен. Рассмотрим пример:

```
// В глобальное пространство имен включается
// лишь несколько идентификаторов
 #include <iostream>
// Предоставляем доступ к именам cout, cin и hex
using std::cout;
using std::cin;
 using std::hex;
int main()
{
   int val;
   cout << "Введите номер:  ";
   cin >> val;
   cout << "Это ваш номер: ";
   cout << hex << val;
  return 0;
}
```

В этой программе имена cin, cout и hex можно использовать непосредственно, а остальная часть библиотеки std остается вне области видимости.

Первоначально библиотека языка C++ находилась в глобальном пространстве имен. При работе со старыми программами на языке C++, в них необходимо включить оператор using namespace std; или указывать перед каждым членом библиотеки квалификатор std::.

Надо помнить, что старые заголовочные файлы (.h) помещают свое содержимое в глобальное пространство имен, а новые – в пространство имен std.