Лекция 8

Область видимости
Время жизни

Область видимости

- **Область видимости** – характеристика именованного объекта

  Область видимости - часть текста программы, на протяжении которого к объекту можно обращаться по его имени.

- Глобальная область видимости

  Имя считается **глобальным**, если оно объявлено вне любой функции, класса или пространства имен. Область видимости – от объявления до конца файла (единицы трансляции).

- Локальная область видимости

  Имя считается **локальным**, если оно объявлено в теле функции или в пространстве имен. Область видимости – от объявления и до окончания блока (кода, ограниченного фигурными скобками {})

Вложенные области

- Объявление вводит имя в область видимости

  Область видимости имени начинается сразу после объявителя, но перед определением

- Области видимости могут быть вложенными

  Объявление имени во вложенном блоке скрывает объявление в охватывающем блоке. После выхода из блока имя восстанавливает прежний смысл

- Скрытые глобальные имена доступны всегда

  Доступ к глобальным переменным осуществляется с помощью оператора доступа ::

Примеры

```
int x ;                 // Глобальное имя x
float c = 7 ;           // Глобальное имя c
void * ptr = &ptr ;     // В инициализаторе имя ptr уже объявлено

int sum ( int x, int y )  // Аргументы – те же локальные имена
{                         // Внешнее x скрыто
    y = c ;               // y – аргумент, c - глобальное
    int c = x + y ;       // Внешнее c скрыто
    int y = c ;           // Ошибка – повторное определение
    x = 5 ;               // x - аргумент
    ::x = c ;             // x – глобальное имя, c уже локальное
    for ( int y = 7; y < 10 ; ++y )  // Скрыт аргумент y
    {
     int c = 1. / y ;  // Локальное c скрыто
     x += c ;
    }
    return c ;            // Локальное c восстановлено
}

void main ()
{
    int x = 9 ;           // Еще один локальный x, виден ТОЛЬКО в main
    int sum = ::sum( 12, x );
}
```

Пространства имен

Пространства имен (namespaces)

- Служат для группировки глобальных имен

```
namespace Math {
  const float PI =3.1415926 ;
  float sin( float );
}
```

переменных

-

```
Math::sin ( Math::PI / 2 );
```

лификатором ::

- Могут быть вложенными

```
namespace A { int x ; namespace B { int x ; }}
A::x = 2 ; A::B::x = 3 ;
```

вложение создает свою область видимости

- Существуют только вне кода

Объявления и определения в ПИ

- ПИ могут быть объявлены несколько раз

```
namespace A { int x ; }
namespace A { int y ; } // x и y из одного пространства имен
```

- Объявления всегда включаются в ПИ

Нельзя объявить новый объект из пространства имен вне определения этого пространства

```
namespace Math {}
float Math::sin ( float ); // Ошибка! Объявление вне ПИ
```

- Определения могут не включаться

Объекты, объявленные в пространстве имен, могут быть определены вне его с помощью квалификатора ::

```
namespace Math
{
    int sum ( int, int ); extern int x ;
}

int Math::sum ( int a, int b ) { return a + b ; } int Math::x ;
```

Ключевое слово **using**

- Раскрытие имени из пространства имен

  Директива **using** позволяет использовать имя из пространства имен внутри текущего и всех вложенных блоков без использования квалификатора

  Директива **using namespace** позволяет использовать все имена из пространства имен внутри текущего и всех вложенных блоков без использования квалификатора

```
void main ()
{
    float val1 = Math::sin ( Math::PI );

    using Math::PI ;

    float val2 = Math::sin (PI );

    using namespace Math ;

    float val3 = cos ( PI );
}
```

Безымянные ПИ и псевдонимы

- ## Безымянные пространства имен

  Служат для объявления объектов с внутренней компоновкой. Все имена безымянного пространства имен видны внутри текущей единицы трансляции

```
namespace {
  const float PI =3.1415926 ;
  float sin( float );
}
```

```
namespace __unused4732 {
  const float PI =3.1415926 ;
  float sin( float );
}
using namespace __unused4732 ;
```

- ## Псевдонимы пространств имен

  Служат для сокращения имен длинных и вложенных пространств имен

```
namespace Math { namespace Trigonometric { namespace Details { int x; }}}

Math::Trigonometric::Details::x = 9 ;

namespace MTD = Math::Trigonometric::Details ;

MTD::x = 10 ;
```

Поиск Кёнига

- Поиск имени вызываемой функции

  Если имя вызываемой функции отсутствует в текущей области видимости, ее поиск осуществляется в областях видимости ее аргументов, причем настолько широко, насколько это возможно.

```
namespace Error {
  enum Type {
    ET_DEBUG,
    ET_WARNING,
    ET_ERROR,
    ET_FATAL,
  };
  void print ( Type t, const char * );
}

void main ()
{
  print ( Error::ET_FATAL, "Abnormal termination" );  // Ок
  print ( 0, "Debug info" );                          // Ошибка
  Error::print ( Error::Type(1), "Trust no one!" );   // Ок
}
```

Операции с пространствами имен

```
namespace ATL {
  int x ;
  int z ;
}

namespace WTL {
  int x ;
  int y ;
}

// Объединенное
namespace Common {
  using namespace ATL ;
  using namespace WTL ;
  using ATL::x ;
}
```

Размещение и время жизни

Размещение в памяти

- Глобальное (статическое)

  Время жизни объекта совпадает с временем жизни программы
- Динамическое

  Время жизни объекта управляется пользователем
- Локальное (Стековое, автоматическое)

  Время жизни объекта ограничено областью видимости
- Временное

  Время жизни объекта ограничено точкой применения

Глобальные переменные

- **Размещены вне функций и классов**

  Пространства имен не влияют на вид размещения.

- **Всегда инициализируются**

  Если специально не указан инициализатор, глобальные объекты простых и адресных типов всегда инициализируются нулем, для объектов пользовательских типов вызывается конструктор по-умолчанию.

- **Создаются до старта программы**

  Порядок создания внутри единицы трансляции задается порядком определений. Порядок создания для объектов в разных единицах трансляции не определен

- **Удаляются после окончания программы**

  Глобальные объекты удаляются из памяти автоматически, в порядке, обратном порядку создания.

Динамические переменные

- Позволяет использовать всю доступную память процесса

  Локальные переменные оперируют стеком, размер которого ограничен. Динамические объекты могут использовать все свободное адресное пространство процесса
- Создаются при помощи оператора **new**

  Команда выделения и освобождения памяти является оператором. Таким образом, память может быть выделена в любой момент в теле программы
- Удаляются при помощи оператора **delete**

  После окончания программы вся(!) динамическая память должна быть освобождена, в противном случае это считается ошибкой.
- Время жизни – от создания до удаления

Операторы **new** и **delete**

- Оператор new

  Выделение памяти под переменную

```
int * p = new int ; int * q = new int(7);
```

  Выделение памяти под массив

```
int * r = new int[12];
```

  Размещение в уже выделенной памяти (**placement new**)

```
int * s = new (buf) s ;
```

- Оператор delete

  Освобождение выделенной памяти

```
delete p ; delete q ;
```

  Освобождение выделенной памяти под массив

```
delete[] s ;
```

  Применение delete к нулю не вызывает никаких действий

  Двойное удаление вызывает ошибку

Локальные переменные

- Выделяются на стеке
- Время жизни совпадает с областью видимости
- Память освобождается в обратном порядке выделения

```
int & f ( int a )
{
  int c = a ;
  int d  = 7 ;
  { double d ; }
  return d ;
}
```

Статические локальные переменные

- Объявляются локально

  Имеют локальную область видимости

- Создаются статически

  Время жизни как у глобальных объектов

- Инициализируются при первом обращении

  Инициализатор статической локальной переменной выполняется только один раз, при выполнении инструкции определения. При последующих обращениях к тому же коду ся»

```
void f ()
{
  static int s = 7 ;
  s++ ;
}
```

```
int& f ()
{
  static int f_ = 0 ;
  return f_ ;
}
```

Размещение литералов

- Литералы базовых типов

  Не имеют размещения и транслируются непосредственно в машинный код

- Литералы строковых типов

  Размещаются статически и существуют до окончания программы

Временные переменные

- Временные аргументы

  Создаются в момент вызова функции. Время жизни – до окончания функции

- Временные возвращаемые значения

  Создаются на стэке вызывающей функции. Время жизни – до окончания инструкции вызова.

- Константные ссылки на временные значения

  Создаются на стэке вызывающей функции. Время жизни продлевается до окончания области видимости