Семинар 2 (по материалам лекции 1)

МГТУ им. Н.Э. Баумана

February 26, 2016

Пространства имен (namespace)

Namespace

Механизм логического группирования программных объектов.

C

```c
// static_frame.h
// global size
int g_frameSize;
```

```c
// dynamic_frame.h
// global size
int g_frameSize;
```

```c
#include "static_frame.h"
#include "dynamic_frame.h"
// COMPILE ERROR
// DUPLICATED g_frameSize
```

Пространства имен (namespace)

Namespace

Механизм логического группирования программных объектов.

C

```c
// static_frame.h
// global size
int g_frameSize;
```

```c
// dynamic_frame.h
// global size
int g_frameSize;
```

```c
#include "static_frame.h"
#include "dynamic_frame.h"
// COMPILE ERROR
// DUPLICATED g_frameSize
```

C – solution

```c
// static_frame.h
// global size
int g_sf_frameSize;
```

```c
// dynamic_frame.h
// global size
int g_df_frameSize;
```

```c
#include "static_frame.h"
#include "dynamic_frame.h"
// ...
g_sf_frameSize = 10;
g_df_frameSize = 20;
// ...
```

Пространства имен (namespace)

Namespace

Механизм логического группирования программных объектов.

C++

```
namespace namespace-name {
  // declarations, definitions
}
```

C++

```
// static_frame.h
namespace staticFrame {
  int g_frameSize;
}
```

```
// dynamic_frame.h
namespace dynamicFrame {
  int g_frameSize;
}
```

Полное квалифицированное имя

Оператор "::" – оператор расширения области видимости

```
#include "static_frame.h"
#include "dynamic_frame.h
// ...
staticFrame::frameSize = 10;
dynamicFrame::frameSize = 20;
// ...
```

Пространства имен (namespace)

Директива using

```
using namespace namespace-name;
```

Указание, что имена из указанного пространства имен могут использоваться так, будто бы объявлены вне пространства имен

```
// static_frame.h
namespace staticFrame {
  class Composition {
    // ...
  };
}
```

Пространства имен (namespace)

Директива using

```
using namespace namespace-name;
```

Указание, что имена из указанного пространства имен могут использоваться так, будто бы объявлены вне пространства имен

```
// static_frame.h
namespace staticFrame {
  class Composition {
    // ...
  };
}
```

```
// frame_user.cpp
#include "static_frame.h"

staticFrame::Composition g_Composition;
// ...
```

```
// frame_user.cpp
#include "static_frame.h"

using namespace staticFrame;
// ...
Composition g_Composition;
// ...
```

Пространства имен (namespace)

Объявление using

```
using namespace-name::entity-name;
```

Указание, что имя entity-name из пространства имен namespace-name может использоваться так, будто бы объявлено вне пространства имен.

```
// static_frame.h
namespace staticFrame {
  class Composition {
    // ...
  };
  class Disposition {
    // ...
  };
}
```

```
// frame_user.cpp
#include "static_frame.h"

using staticFrame::Composition;
Composition g_Composition;
staticFrame::Disposition g_Disposition;
// ...
```

Пространства имен (namespace)

Неименованные namespace

Ограничение области видимости программных объектов.

```
1 namespace {
2   int g_size = 0;
3   // ...
4 }
5
6 int g_thatSize = g_size; // ERROR: undefined g_size
```

Пространства имен (namespace)

Неименованные namespace

Ограничение области видимости программных объектов.

```
1 namespace {
2   int g_size = 0;
3   // ...
4 }
5
6 int g_thatSize = g_size; // ERROR: undefined g_size
```

Псевдонимы пространств имен

```
1 namespace American_Telegraph_and_Telephone {
2   int g_numbers = 0;
3   // ...
4 }
5 // Too long name for me
6 namespace ATT = American_Telegraph_and_Telephone;
7 int my_numbers = ATT::g_numbers;
```

Пространства имен (namespace)

Пространства имен открыты

```
1 namespace staticFrame {
2   int g_frameSize = 0;
3   // ...
4 }
5 // ...
6 namespace staticFrame {
7   int g_frameDepth = 0;
8   // ...
9 }
```

Пространства имен (namespace)

Пространства имен открыты

```
1 namespace staticFrame {
2   int g_frameSize = 0;
3   // ...
4 }
5 // ...
6 namespace staticFrame {
7   int g_frameDepth = 0;
8   // ...
9 }
```

Nested namespaces

```
1  namespace spiritical {
2    int g_layers = 0;
3  }
4
5  namespace staticFrame {
6    namespace physical {
7      int g_size;
8    }
9    using namespace spiritical;
10 }
```

Пространства имен (namespace)

Поиск имен

```
1 namespace staticScene {
2   class Frame { /* ... */ };
3   int format(const Frame& f);
4 }
5
6 void f(staticScene::Frame f, double d) {
7   int a = format(f);
8   int b = format(d); // error: no format(double)
9 }
```

- Поиск функции в текущей области видимости
- Поиск функции в пространствах имен ее аргументов

Классы

Кратко

- пользовательский тип
- состоит из членов – полей и методов
- доступ к полям и методам ограничивается при объявлении класса
- обращение к членам класса посредством оператора "." и "->"
- специальные методы для инициализации экземпляров, копирования, перемещения и т.д.

Пример

```
1  class X {
2  private:
3    int m_x, m_y; // data
4  public:
5    X() : m_x(-1), m_y(0) {} // ctor
6    X(int x , int y) : m_x(x), m_y(y) {} // ctor
7
8    int f() { // member function
9      return m_x + m_y;
10   }
11 };
```

Методы класса

Методы: inline

```
1 class X {
2 // ...
3 public:
4   int f() {
5     return m_x + m_y;
6   }
7 };
```

Методы: вне определения класса

```
1 class X {
2 // ...
3 public:
4   int f();
5 };
6
7 int X::f() {
8   return m_x + m_y;
9 }
```

Управление доступом

Метки доступа

```
class X {
private: // is default for classes
  int m_x;
  int a();
protected:
  int m_y;
  int b();
public: // is default for struct
  inc c() {return m_x + m_y;}
}
```

```
// ...
X x;
x.m_x = 1; // error: is private
x.m_y = 1; // error: is protected (read: private)
x.a(); // error: is private
x.b(); // error: is protected
x.c(); // OK: is public!
```

Статические члены класса

static variable

Переменная, доступная для всех экземпляров класса

```
class X {
  static int m_x;
public:
  int c() {return m_x++;}
};

// initialisation is a must
int X::m_x = 0; // "static" keyword is not used
```

```
X x, x1;
std::cout << x.c(); // prints 0
std::cout << x1.c(); // prints 1
```

Статические члены класса

static method

Метод класса, для которого не требуется, чтобы он вызывался для конкретного объекта.

- Не имеет доступа к нестатическим членам класса.

```
class X {
  static int m_x;
  int m_y;
public:
  static int static_c() {return m_x++;}
  static int static_d() {return m_y++;} // ERROR: m_y is not static
};
int X::m_x = 0;
```

```
X x, x1;
std::cout << x.static_c(); // prints 0
std::cout << X::static_c(); // prints 1
```

Константные методы

const method

Метод класса, который не изменяет состояния объекта.

```
class Population {
  int m_MenCount;
  int m_WomenCount;
public:
  int totalPopulation() const {
    return (m_MenCount+m_WomenCount);
  }
  void onMensBirth() {
    ++m_MenCount;
  }
};
```

```
const Population p1;
Population p2;
p1.totalPopulation(); // OK
p1.onMensBirth(); // error, method is not const
p2.totalPopulation(); // OK
p2.onMensBirth(); // OK
```

Самостоятельное изучение

```
mutable
```

Указатель на себя

this

Ключевое слово, используемое в нестатических методах класса для обозначения указателя на текущий объект.

```
class Population {
  int m_MenCount;
  int m_WomenCount;
public:
  // ...
  Population& onMensBirth() {
    ++m_MenCount;
    return *this;
  }
  Population& onWomensBirth() {
    ++m_WomenCount;
    return *this;
  }
};
```

```
const Population p1;
p1.onMensBirth().onWomensBirth();
std::cout<<p1.totalPopulation();
```

Конструктор

Специальный метода класса, предназначеный для инициализации объектов в момент их создания.

```
class X { /* ... */ };
```

По умолчанию

```
X x;
X* px = new X();
```

С параметрами + преобразования

```
X x(1, 3);
X x1 = 4;
```

Копирования

```
X x;
X x1(x); // x1 is a copy
X x2 = x; // x1 is a copy
```

Перемещения

```
X f() { // yes, that's function
  X x;
  return x;
}
X x1 = f(); // here x is moved, not copied
```

Конструктор по умолчанию

Явное определение

```
class X {
  int m_x;
public:
  X();
};
X::X() { m_x = 0; }
```

```
class X { /* ... */
public:
  X(int x = 15) { m_x = x; };
};
```

Сгенерированный компилятором

```
class X {
  int m_x;
};
// ...
X x; // OK
```

Сгенерированный конструктор по умолчанию вызывает:

- конструкторы по умолчанию базовых классов
- конструкторы по умолчанию для объектов-членов

BEWARE!!! Объекты базовых типов не инициализируются!

Конструктор с параметрами

Выполняется при инициализации с параметрами.

```
X x(1, 2); // uses X::X(int, int)
X x2 = 1; // uses X::X(int)
X x3("hello"); // uses X::X(const char*)
```

```
class X {
  int m_x;
public:
  X(int x, int y) { m_x = x + y; };
  X(int x) : m_x(x) { }; // LOOK: initializer list
  explicit X(const char* c) {/*...*/};
};
```

explicit

```
X x = 1; // ok, ctor is implicit
X x1 = "hello"; // error, need explicit call
X x2("hello"); // ok, ctor called explicitly
X x3 = X("hello"); // ok, ctor called explicitly
```

Конструктор преобразования

Implicit конструктор с одним параметром. Если определен, поддерживается преобразование типа static_cast к данному классу.

```
void f(X x) {/*...*/};
f((X)1); // ok
```

Конструктор копирования

Вызывается при инициализации другим объектом.

```
X x; // default ctor here
X x1(x); // direct initialization
X x2 = x; // copy initialization
```

```
class X {
  int m_x;
public:
  X(const X& src) { m_x = src.m_x; }; // copy ctor
};
```

Сгенерированный компилятором

```
class X {
  int m_x;
};
// ...
X x(x1); // OK
```

Поэлементное копирование объектов-членов.

Конструктор перемещения

Конструктор, принимающий rvalue-ссылку на объект, и "крадущий" ресурсы объекта. Объект-аргумент конструктора остается валидным, но переходит в неопределенное состояние.

```
X x(std::move(x1)); // initialization
```

```
f(std::move(x1)); // argument passing
```

```
return x; // returning from function (if has move ctor)
```

```
class X {
  int* m_px;
public:
  X(X&& src) {
    m_px = src.m_px;
    src.m_px = nullptr;
  }; // move ctor
};
```

X&&

Rvalue-ссылка. Специальный тип для поддержки move-семантики. Преобразование к rvalue-ссылке:

```
std::move(x);
```

Деструктор

Специальный метод класса, вызывающийся при разрушении объекта (когда закончилось его время жизни).
Вызывается:

- Завершение программы (static storage objects)
- Разрушение объекта при помощи delete
- Раскручивание стека (удаление локальных объектов)

```
{
  X x;
} // X::~X()
```

```
X* px = new X();
delete px; // X::~X()
```

```
class X {
public:
  ~X() {/* release resources here */}
};
```

Объекты-члены класса, содержащиеся в классе по значению, уничтожаются автоматически после выполнения деструктора в порядке, обратном инициализации.

friend specifier

friend function

[Внешняя] функция, имеющая доступ ко всем членам класса независимо от класса доступа.

```
1 class X {
2   int m_x;
3   friend void printX(X); // access specs do not matter
4 };
5 void printX(X x) {
6   std::cout<<x.m_x; // yes, I access private member
7 }
```

friend class

Класс, имеющий доступ ко всем членам данного класса независимо от класса доступа.

```
1 class X {
2   int m_x;
3   friend class Printer; // access specs do not matter
4 };
5 class Printer {
6 public:
7   void print(X x) {
8     std::cout<<x.m_x; // yes, I access private member
9   }
```

Не освещено

- Список инициализаторов, порядок инициализации членов
  самостоятельно
- mutable specifier
  самостоятельно
- Что делать, если в конструкторе произошла ошибка?
  узнаем позже, сейчас std::terminate();
- = default; = delete для конструкторов/деструкторов
  самостоятельно
- Могут ли ctor/dtor быть не public?
- Инициализация членов внутри объявления класса (C++11)
  самостоятельно
- Операторы присваивания: copy, move
  узнаем позже
- Ctors, dtors & иерархии
  узнаем позже