Программирование на языке C++

Вводный курс

Александр Морозов
gelu.speculum@gmail.com

ИТМО, весенний семестр 2020

Содержание

Пространства имён

Позволяют использовать одинаковые имена для разных сущностей в разных частях программы.

```
 1  namespace A {
 2    void f(int);
 3    double g();
 4  }
 5  namespace B {
 6    bool f(const std::string &);
 7    void g();
 8  }
 9  int main()
10  {
11    int i = 101;
12    A::f(i);
13    double x = A::g();
14    B::g();
15  }
```

Пространства имён и определения

```
 1    namespace A {
 2      int f();
 3      void g();
 4      double sqrt(double x)
 5      {
 6        ...
 7      }
 8    }
 9
10    int A::f()
11    { ... }
12
13    void A::g()
14    { ... }
```

Вложенные пространства имён

```
 1    namespace A {
 2      namespace B {
 3        namespace C {
 4          class C {...};
 5        }
 6      }
 7    }
 8    namespace A::B::C {
 9      struct S {...};
10    }
11
12    int main()
13    {
14      A::B::C::C c;
15      A::B::C::S s;
16    }
```

Область видимости пространства имён

```
 1    namespace A {
 2      const int n = 1;
 3
 4      namespace B {
 5        void f() { std::cout << n << std::endl; }
 6      }
 7    }
 8
 9    const int n = 0;
10
11    namespace A {
12      namespace B {
13        const int n = 3;
14
15        void g() { f(); }
16
17        void h() { std::cout << n << std::endl; }
18      }
19    }
20
21    int main()
22    {
23      A::B::g();
24      A::B::h();
25    }
```

Область видимости пространства имён, продолжение

```
 1    namespace A {
 2      int f(int);
 3
 4      const int n = -13;
 5    }
 6
 7    int A::f(int i = n)
 8    {
 9        return i;
10    }
```

Анонимные пространства имён

```
 1    namespace {
 2      class C {...};
 3      C c;
 4      void f() {...}
 5    }
 6    namespace A {
 7      namespace {
 8        int g() {...};
 9      }
10    }
11
12    int main()
13    {
14      C cc = c;
15      f();
16      return A::g();
17    }
```

Полностью квалифицированные имена

```
 1    namespace U {
 2      namespace X {
 3        void f();
 4      }
 5
 6      namespace Y {
 7        namespace X {
 8          double f(int);
 9        }
10
11        void g(int n) {
12          double x = X::f(n);
13          ::U::X::f();
14        }
15      }
16    }
```

Псевдонимы пространств имён

```
1   namespace A::B::C::D::E {
2     class X {...};
3   }
4   namespace abcde = A::B::C::D::E;
5
6   int main()
7   {
8     abcde::X x;
9   }
```

using директива и объявление

```
1   namespace A {
2     using namespace std; // using directive
3
4     void say_hi()
5     {
6       cout << "Hi!" << endl;
7     }
8   }
9   namespace B {
10    using std::string; // using declaration
11
12    std::size_t strlen(const string & s)
13    {
14      return s.size();
15    }
16  }
```

Особенности using

```
1   class X {};
2
3   void swap(X &, X &) {}
4
5   namespace N {
6     void swap(X &, X &) {}
7   }
8
9   struct Y { X x; };
10
11  void swap(Y & l, Y & r) {
12    using std::swap;
13    using namespace N; // N::swap is hidden
14    swap(l.x, r.x); // ::swap is found by ADL
15  }
```

Добавление в пространство std

В общем случае добавление в пространство `std` запрещено и является UB.

Исключение: специализация шаблонных классов из пространства `std`, которая зависит хотя бы от одного пользовательского типа.

Ограничения добавления специализаций:

- полная специализация метода класса из `std` – UB
- полная или частичная специализация класса, являющегося вложенным в класс из `std` – UB

Пример правильного расширения std

```
1   struct S
2   {
3     std::string str;
4   };
5
6   namespace std {
7     template <>
8     struct hash<S> : hash<std::string> {
9       using argument_type = S;
10      using result_type = size_t;
11      size_t operator() (const S & s) const noexcept {
12        return hash<std::string>::operator() (s.str);
13      }
14    };
15  }
16
17  void f() {
18    std::unordered_set<S> set;
19    set.insert("foo");
20    set.insert("foo");
21    set.insert("bar");
22    assert(set.size() == 2);
23  }
```

Содержание

Виды поиска имён

- Квалифицированный
- Неквалифицированный
- ADL

Квалифицированный поиск имён

Квалифицированное имя – имя справа от оператора `::`.

Может относиться к:

- члену класса
- сущности из пространства имён
- элементу `enum`

Слева от оператора `::` – либо ничего, либо уже найденное имя (класса, пространства имён или `enum`).

Примеры квалифицированного поиска имён

```
 1  namespace N {
 2  struct A {
 3    int f() const { return 10; }
 4  };
 5  struct B : A {
 6    int f() const { return 15; }
 7
 8    enum class E { A, B, C };
 9  };
10  }
11
12  void g() {
13    N::B b;
14    std::cout << b.f() << std::endl;
15    std::cout << b.A::f() << std::endl;
16    std::cout << static_cast<int>(N::B::E::A);
17  }
```

Приоритет поиска в пространствах имён

```
1   namespace A {
2     void f();
3     void g();
4   }
5   namespace B {
6     using namespace A;
7     void f();
8   }
9
10  int main() {
11    B::f(); // 'f' from 'B'
12    B::g(); // 'g' from 'A'
13  }
```

Elaborated type specifier

```
 1    class X {
 2    public:
 3      struct Y {};
 4    private:
 5      int Y;
 6    };
 7
 8    void f() {
 9      char X;
10      X x; // error, 'X' refers to variable X
11      class X xx; // OK
12      X::Y y; // error, 'X::Y' refers to private member 'Y'
13      class X::Y yy; // OK
14    }
```

Неквалифицированный поиск

Если имя не стоит справа от `::` применяется неквалифицированный поиск.

При таком поиске будут анализироваться только подходящие сущности (например, для имени слева от `::` – только пространства имён, классы и `enum`).

Такой поиск обрабатывает `using` директивы, как если бы содержимое соответствующего пространства имён находилось в ближайшем окружающем пространстве имён.

Поиск будет идти вплоть до обнаружения хотя бы одной сущности с таким именем или до исчерпания вариантов. Результатом будут все подходящие сущности с искомым именем, объявленные на первом уровне, на котором обнаружилось имя.

ADL: пример

```
1    std::cout << x << std::endl;
2    operator << (std::cout, x);
3    operator << (std::cout, std::endl);
4
5    std::cout << endl; // error: 'endl' identifier not found
6    endl(std::cout); // OK
```

ADL: пример посложнее

```
 1    namespace adl {
 2      struct X {
 3        int n = 10;
 4        X & operator + (const X & other) {
 5          n += other.n;
 6          return *this;
 7        }
 8      };
 9      X operator + (const X & lhs, const int rhs)
10      {
11        X x = lhs;
12        x.n += rhs;
13        return x;
14      }
15      enum class E { A, B, C };
16      std::ostream & operator << (std::ostream & strm, const E & e)
17      { return strm << static_cast<int>(e); }
18    }
19
20    int main() {
21      adl::X x1, x2 = adl::X{} + 10;
22      std::cout << (x1 + x2).n << std::endl;
23      adl::E e = adl::E::B;
24      std::cout << e << std::endl;
25    }
```

ADL: правила

Применяется для имени из выражения вызова функции.

Не производится, если неквалифицированный поиск дал:

- член класса
- объявление функции на уровне блока (не считая `using` объявления)
- объявление сущности, не являющееся функцией

Иначе каждый аргумент функционального вызова добавляет пространства имён и/или классы во множество, в котором затем будет производиться поиск (помимо обычного неквалифицированного поиска).

В процессе поиска отбираются только объявления функций.

ADL: правила для различных типов аргументов

В зависимости от типа аргумента к поиска добавляется:

- базовый тип – ничего
- класс – сам класс, его предки, его область определения (окружающий класс или пространство имён)
- указатель на функцию – анализ применяется для типа возвращаемого значения и типов всех параметров
- шаблон – в дополнение к обычным правилам, анализ применяется для каждого шаблонного параметра-типа
- `enum` – область определения `enum`
- `T *` – анализ применяется для `T`
- указатель на метод – анализ применяется для класса, типа возвращаемого значения и всех параметров
- указатель на поле – анализ применяется для типа поля и класса