# Программирование на языке C++

## Вводный курс

Александр Морозов
gelu.speculum@gmail.com

ИТМО, весенний семестр 2020

# Содержание

# Последовательность неявного преобразования типов

1. 0 или 1 цепочка стандартных преобразований
2. 0 или 1 пользовательское преобразование
3. 0 или 1 цепочка стандартных преобразований

Цепочка стандартных преобразований:

1. 0 или 1 преобразование категории значения, преобразование массива или функции к указателю
2. 0 или 1 числовое расширение или числовое преобразование
3. 0 или 1 повышение квалификации

# Пользовательские преобразования

- не `explicit` конструктор
- не `explicit` оператор приведения типа

```
1   struct S
2   {
3     S(int);
4     operator double() const;
5   };
6
7   double f(const S & s)
8   { return s; }
9
10  double g(const int x)
11  { return f(x); }
```

# Содержание

# Перегрузка функций

Допустимо иметь несколько функций с одним именем, если в каждом случае вызова функции с таким именем компилятор может найти один наиболее подходящий вариант.

```
1  void f(int) { ... }
2  void f(double) { ... }
3  void f(const S &) { ... }
4
5  f(10);
```

# Этапы выбора функции

- поиск кандидатов – поиск сущностей по имени
- выбор подходящих к контексту вызова
- выбор единственного наилучшего кандидата

При этом в зависимости от контекста вызова ищутся:

- контекст вызова функции – функции с таким именем
- контекст функционального вызова объекта – члены-операторы функционального вызова соответствующего класса и его предков, а также операторы преобразования типа к типу подходящей функции
- контекст вызова оператора – члены-операторы (если первый или единственный операнд имеет сложный тип), свободные операторы, встроенные операторы

# Правила выбора подходящих к контексту вызова

Учитываются факторы:

- число аргументов
- типы аргументов
- допустимая ссылочная связываемость

# Ранжирование неявных преобразований типов

Каждому из 3 возможных этапов последовательности неявного преобразования назначается ранг:

1. точное совпадение
2. числовое расширение
3. преобразование

Ранг всей последовательности = наибольшему рангу этапов.

Связывание со ссылкой – либо точное совпадение, либо преобразование (к одному из базовых типов).

Отсутствие необходимости преобразования считается последовательностью нулевой длины с минимальным рангом.

# Ранжирование неявных преобразований типов, продолжение

Стандартное преобразование лучше пользовательского.

Чем короче необходимая цепочка стандартных преобразований – тем она лучше.

Связывание rvalue с rvalue ссылкой лучше связывания rvalue с const lvalue ссылкой.

Среди связываний lvalue с lvalue ссылкой лучше то, у которого меньше cv-квалификаторов.

Из двух последовательностей, содержащих пользовательское преобразование к одному типу, лучше та, у которой лучше третий этап.

# Выбор единственного наилучшего кандидата

Все подходящие кандидаты попарно сравниваются и выбирается наилучший из них. Несколько одинаково наилучших кандидатов – ошибка компиляции.

Один кандидат лучше другого, если требуемые неявные преобразования для всех его аргументов не хуже (имеют ранг не ниже) таковых для конкурента, а кроме того:

1. для хотя бы одного аргумента требуемое неявное преобразование лучше, чем у конкурента
2. либо этот кандидат – не шаблонный, а конкурент – шаблонный
3. либо этот кандидат более шаблонно-специализированный, чем конкурент

# Содержание

# Друзья классов

`friend` объявление: в теле класса можно объявить другой класс или функцию “другом”, что даст такому другу доступ ко всем членам класса (игнорируя спецификаторы доступа).

Отношение дружбы – не транзитивно, не наследуемо.

Возможные формы:

- объявление функции (оператора)
- определение функции (оператора)
- предварительное объявление класса
- объявление видимого класса / `union`

# Примеры friend

```
1  struct A {
2      void f();
3      int h();
4      int g(double) const;
5  };
6  class B {};
7  class C {
8      int x;
9
10     friend std::ostream & operator << (std::ostream & strm
           ↪ , const C & c); // definition later
11     friend void A::f();
12     friend int A::f(), A::g(double) const;
13     friend class D; // definition later
14     friend B;
15 };
```

# Пример friend определения функции

```
1  class C {
2  public:
3      C(int i) : x(i) {}
4  private:
5      int x;
6
7      friend int get_x(const C & c)
8      { return c.x; }
9  };
10
11 C c(10);
12 int a = get_x(c); // OK (ADL)
13 int b = get_x(10); // error: 'get_x' is not declared
```

# Пример ограничения friend

```
1  class BigOne {
2      // lots of stuff
3      friend class Backdoor;
4  };
5
6  class Backdoor {
7      friend class UserA;
8      friend class UserB;
9      friend class UserC;
10     friend class UserD;
11
12     void access();
13 };
```

# Содержание

# Перегрузка операторов

Перегруженные операторы – функции со специальным именем.

- `operator` *op* – обычный оператор
- `operator` *type* – пользовательское приведение типа
- `operator new` – пользовательская аллокация
- `operator new []` – пользовательская аллокация
- `operator delete` – пользовательская деаллокация
- `operator delete []` – пользовательская деаллокация
- `operator ""` *suffix* – пользовательский литерал

Если в выражении один из операндов – сложный тип (класс, `enum`), то производится поиск перегруженной функции, реализующей данный оператор.

# Ограничения перегрузки операторов

- Нельзя перегружать некоторые операторы (например, `::`)
- Нельзя менять приоритет, ассоциативность операторов
- Нельзя менять число аргументов операторов
- Нельзя создавать новые операторы
- Перегруженный оператор `->` должен возвращать либо указатель, либо объект, для которого перегружен `->`
- Перегруженные операторы `&&` и `||` не имеют ленивого вычисления аргументов (но сохраняют порядок вычисления аргументов)

# Операторы – члены классов

Оператор может быть определён, как член класса:

```
1  struct SS {};
2  struct S {
3      S & operator += (const S & other) { ... }
4      const SS & operator * () const { ... }
5  };
6
7  S s1, s2;
8  s1 += s2;
9  const SS & ss = *s1;
```

При поиске имён, если первый операнд в выражении вызова оператора имеет сложный тип, то осуществляется как поиск свободных функций, так и членов классов.

# Отображение синтаксиса операторов

| Оператор | Член класса | Свободная функция |
|---|---|---|
| @a | (a).operator@ () | operator@ (a) |
| a@ | (a).operator@ (0) | operator@ (a, 0) |
| a@b | (a).operator@ (b) | operator@ (a, b) |
| a=b | (a).operator= (b) | – |
| a(b...) | (a).operator() (b...) | – |
| a[b] | (a).operator[] (b) | – |
| a-> | (a).operator-> () | – |

# Доступные для перегрузки операторы

- арифметические `+`, `-`, `*`, `/`, `%`, `+=`, `-=`, `*=`, `/=`, `%=`
- инкремента/декремента `++`, `--`
- битовые `^`, `&`, `|`, `~`, `^=`, `&=`, `|=`
- сдвига `<<`, `>>`, `<<=`, `>>=`
- логические `&&`, `||`, `!`
- сравнения `<`, `>`, `==`, `!=`, `<=`, `>=`
- присваивания `=`
- разыменования, доступа к члену через указатель и указатель на член `*`, `->`, `->*`
- индексации `[]`
- функционального вызова `()`
- запятой `,`

# Оператор пользовательского приведения типа

```
1  struct S {
2      operator X ();
3      explicit operator Y ();
4      operator auto () { return 1; }
5      template <class T>
6      operator T ();
7  };
8
9  S s;
10 X x = s;
11 Y y = static_cast<Y>(s);
12 Y yy(s);
```

Такой метод объявляется нестатическим, без параметров и без возвращаемого типа. Может иметь cv-квалификатор, `inline`, `virtual`, `constexpr` спецификаторы.