Перегрузка и переопределение

Александр Смаль

Академический университет
1 ноября 2013
Санкт-Петербург

Перегрузка функций (overloading)

```
double square(double d) { return d * d; }
int    square(int    i) { return i * i; }
```

Перегрузка имен методов

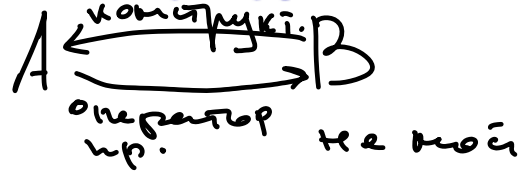
```
struct Point2D {
    explicit Point2D(double d) {}
    Point2D mult(double d) const {
        return Point(x * d, y * d);
    }

    double  mult(Point2D const& p) const {
        return x * p.x + y * p.y;
    }

    double x, y;
};
```

Point2D p;

p.mult(q);

p.mult(3);

Правила перегрузки



1. Если есть точное совпадение.
2. Нет функции, которая могла бы подойти с учётом преобразований.
3. Есть несколько подходящих функций.
   1. Расширение типов.
      `char, unsigned char, short, enum` → `int`
      `unsigned` → `int/unsigned int`
      `float` → `double`
   2. Стандартные преобразования (числа, указатели).
   3. Пользовательские преобразования.

   В случае нескольких параметров нужно, чтобы выбранная функция была *сильно лучше* остальных.
   Функции с переменным числом параметров и шаблоны имеют наименьший приоритет.

f(int, ...);

f(a, b, c)
(0, 2, 3) f(int, int, Vector)
(0, 1, 3) f(int, double, Vector

Правила перегрузки

1. Если есть точное совпадение.
2. Нет функции, которая могла бы подойти с учётом преобразований.
3. Есть несколько подходящих функций.
   1. Расширение типов.
      `char, unsigned char, short, enum` → `int`
      `unsigned` → `int/unsigned int`
      `float` → `double`
   2. Стандартные преобразования (числа, указатели).
   3. Пользовательские преобразования.

   В случае нескольких параметров нужно, чтобы выбранная функция была *сильно лучше* остальных.
   Функции с переменным числом параметров и шаблоны имеют наименьший приоритет.

Вывод

Не стоит злоупотреблять перегрузкой.

Перегрузка при наследовании

```
struct File {
    ...
    void write( char const * s );
    ...
};
struct FormattedFile : File {
    void write(string s);
    void write( int i );
    void write( double d );
    using File::write;
    ...
};
```

```
FormattedFile f(...),
char const * s =
    "Hello";
f.write(s),
File & p = f;
p.write(s);
f.File::write(s);
```

Переопределение методов (overriding)

```
struct Person {
    ...
    string name() const { return name_; }
    ...
};
struct Man : Person {
    ...
    string name() const {return "Mr. " + Person::name();}
    ...
};
struct Woman : Person {
    ...
    string name() const {return "Mrs. " + Person::name();}
    ...
};

int main() {
    Person * p = new Woman("Mary");
    cout << p->name(); // "Mary"
    ...
}
```

Person

Man Woman

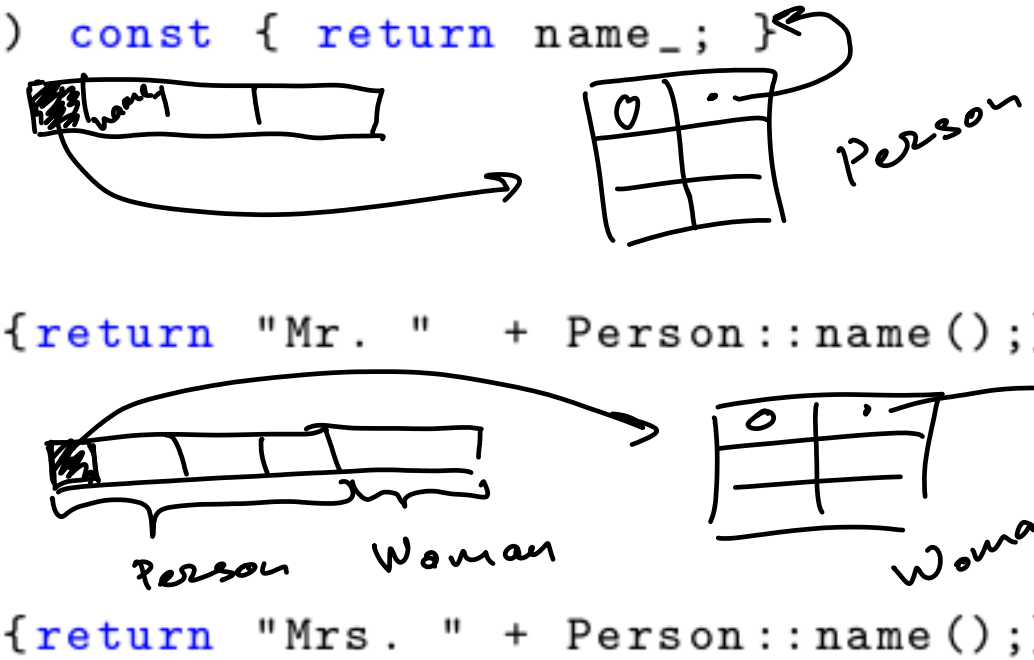
Woman * w = ...

cout << w -> name();

"Mrs. Mary"

Виртуальные методы

```
struct Person {
    ...
    virtual string name() const { return name_; }
    ...
};
struct Man : Person {
    ...
    string name() const {return "Mr. " + Person::name();}
    ...
};
struct Woman : Person {
    ...
    string name() const {return "Mrs. " + Person::name();}
    ...
};

int main() {
    Person * p = new Woman("Mary");
    cout << p->name(); // "Mrs. Mary"
    ...
}
```



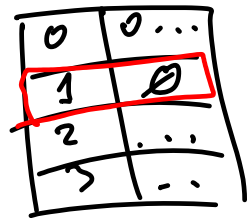
p->f() → call 0x....

p->name() → call p->vtbl[0]

Чисто-виртуальные методы

абстрактный

```
struct Person {
    ...
    virtual string ocupation() const = 0;
    ...
};
struct Student : Person {
    ...
    string ocupation() const { return "student"; }
    ...
};
struct Teacher : Person {
    ...
    string ocupation() const { return "teacher"; }
    ...
};

int main() {
    Person * p = new Teacher("Mary");
    cout << p->ocupation(); // "teacher"
    ...
}
```



string Person::ocupation() const {}

Person::ocupation() +

...

Person pers;

Таблица виртуальных функций

```
struct Person {
    ...
    virtual string name() const { return name_; }
    virtual string ocupation() const = 0;
    ...
};
struct Student : Person {
    ...
    /*virtual*/ string ocupation() const { return "student"; }
    virtual int group() const { return group_; }
    ...
};
```

Person

0	name	0xabcd
1	ocupation	0x0000

Student

0	name	0xabcd
1	ocupation	0xbcde
2	group	0xcdef

Снова о статической и динамической типизации

C: squared
square i

C++:

square(k)
double int

Полиморфизм

Возможность единообразно обрабатывать объекты разных типов.

Перегрузка функций

Механизм статической типизации, *статический полиморфизм*.

Виртуальные функции

Механизм динамической типизации, *динамический полиморфизм*.

Person * p = ...
p->name();

Виртуальный деструктор

К чему приведёт такой код?

```
struct Person {
    ...
};
struct Teacher : Person {
    ...

private:
    string course;
};

int main() {
    Person * p = new Teacher("Mary");
    ...
    delete p;   // ~Person()
}
```

Виртуальный деструктор

Правильная реализация.

```
struct Person {
    ...
    virtual ~Person() = 0 {};
};

struct Teacher : Person {
    ...

private:
    string course;
};

int main() {
    Person * p = new Teacher("Mary");
    ...
    delete p;   // ~Teacher()
}
```

Person::~Person() {}

Наследование ⇒ вирт. деструктор.

Порядок построения таблицы виртуальных функций

```
struct Person {
    ...
};

struct Teacher : Person {
    ...
};

struct Professor : Teacher {
    ...
};
```

Person

0	name	0xabcd
1	ocupation	0x0000

Teacher

0	name	0xabcd
1	ocupation	0xbcde
2	course	0xcdef

Professor

0	name	0xd1ab
1	ocupation	0xba2e
2	course	0xcdef
3	thesis	0x5efa

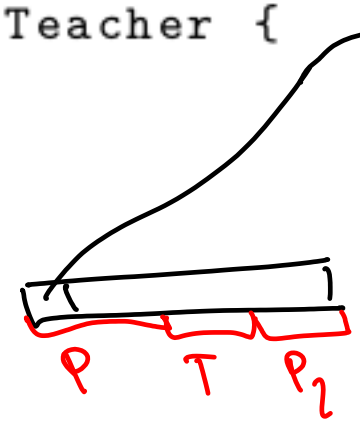
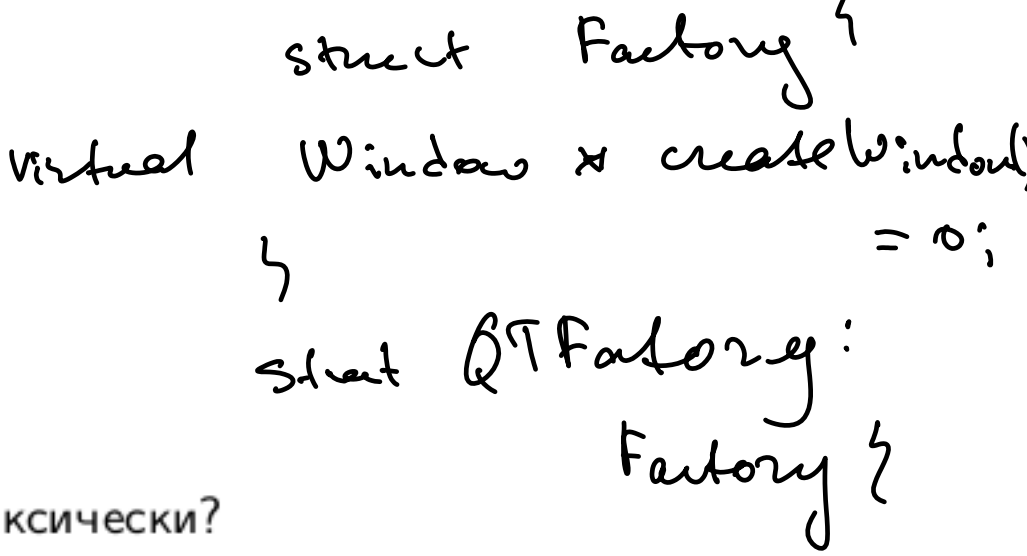


Таблица виртуальных функций в конструкторе и деструкторе

```
struct Person {
    virtual string name() const { return name_; }
    ...
};

struct Teacher : Person {
    Teacher(string const& name) : Person(name) {
        cout << name();
    }
    ...
};

struct Professor : Teacher {
    string name() const { return "Prof. " + name(); }
    ...
};
```

Вопросы для проверки

1. Опасность вызова виртуальных методов из конструктора.



2. Почему это не запрещено синтаксически?