

Программирование на языке C++

Лекция 6

Шаблоны функций

Александр Смаль

Шаблоны функций: возведение в квадрат

int float

```
// C
→ int squarei(int x) { return x * x; }
→ float squaref(float x) { return x * x; }

// C++
→ int square(int x) { return x * x; } | 1.
→ float square(float x) { return x * x; }

// C++ + OOP
→ struct INumber {
    → virtual INumber * multiply(INumber * x) const = 0;
    ...
};
→ struct Int : INumber { ... }; |
→ struct Float : INumber { ... }; |
→ INumber * square(INumber * x) { return x->multiply(x); } | 2.

// C++ + templates
→ template <typename Num>
→ Num square(Num x) { return x * x; } | 3.
```

Шаблоны функций: сортировка

```
// C
→ void qsort(void * base, size_t nitems, size_t size, /*function*/);
// C++
→ void sort(int * p, int * q);
→ void sort(double * p, double * q);
// C++ + OOP
struct IComparable {
    virtual int compare(IComparable * comp) const = 0;
    virtual ~IComparable() {}
};
→ void sort(IComparable ** p, IComparable ** q);
// C++ + templates
template <typename Type>
void sort(Type * p, Type * q);
```

→ **NB:** у шаблонных функций нет параметров по умолчанию.

Вывод аргументов (deduce)

```
→ template <typename Num>
   Num square(Num n) { return n * n; }
           int

template <typename Type>
→ void sort(Type * p, Type * q);
           float *   float *
                int *

template <typename Type>
→ void sort(Array<Type> & ar);
           Array<double>

void foo() {
    → int a = square<int>(3);
    → int b = square(a) + square(4); // square<int>(..)
        float * m = new float[10];
    → sort(m, m + 10); // sort<float>(m, m + 10)
    → sort(m, &a); // error: sort<float> vs. sort<int>
        Array<double> ad(100);
    → sort(ad); // sort<double>(ad)
}
```

Шаблоны методов

```
1. template <class Type>
→ struct Array {
    template<class Other>
    Array( Array<Other> const& other )
        : data_(new Type[other.size()])
        , size_(other.size()) {
        for(size_t i = 0; i != size_; ++i)
            data_[i] = other[i];
    }

    2. template<class Other>
    Array & operator=(Array<Other> const& other); ←
    ...
};

1. template<class Type>
2. template<class Other>
Array<Type> & Array<Type>::operator=(Array<Other> const& other)
{ ... return *this; }
```

Array<int> a;
Array<double> b(a);
a = b;

double ← int