Лекция 13. Шаблоны, часть вторая

Александр Смаль

CS центр

5 декабря 2017

Санкт-Петербург

Полная специализация шаблонов: классы

```
template<class T>
struct Array {
    ...
    T *   data_;
};
template<>
struct Array<bool> {
    static unsigned const BITS = 8 * sizeof(unsigned);
    explicit Array(size_t size)
        : size_(size)
        , data_(new unsigned[size_ / BITS + 1])
    {}
    bool operator[](size_t i) const {
        return data_[i / BITS] & (1 << (i % BITS));
    }
private:
    size_t  size_;
    unsigned *  data_;
};
```

Полная специализация шаблонов: функции

```
template<class T>
void swap(T & a, T & b)
{
    T tmp(a);
    a = b;
    b = tmp;
}

template<>
void swap<Database>(Database & a, Database & b)
{
    a.swap(b);
}

template<class T>
void swap(Array<T> & a, Array<T> & b)
{
    a.swap(b);
}
```

Специализация шаблонов и перегрузка

```
template<class T>
void foo(T a, T b) { cout << "same types" << endl; }

template<class T, class V>
void foo(T a, V b) { cout << "different types" << endl; }

template<>
void foo<int, int>(int a, int b) {
    cout << "both parameters are int" << endl;
}

int main() {
    foo(3, 4);
    return 0;
}
```

Частичная специализация шаблонов

```
template<class T>
struct Array {
    T & operator[](size_t i) { return data_[i]; }
    ...
};

template<class T>
struct Array<T *> {
    explicit Array(size_t size)
        : size_(size)
        , data_(new T *[size_])
    {}

    T & operator[](size_t i) { return *data_[i]; }

private:
    size_t  size_;
    T **    data_;
};
```

Нетиповые шаблонные параметры

Параметрами шаблона могут быть типы, целочисленные значения, указатели/ссылки на значения с внешней линковкой и шаблоны.

```
template<class T, size_t N, size_t M>
struct Matrix {
    ...
    T & operator()(size_t i, size_t j)
    { return data_[M * j + i]; }
private:
    T data_[N * M];
};

template<class T, size_t N, size_t M, size_t K>
Matrix<T, N, K> operator*(Matrix<T, N, M> const& a,
                          Matrix<T, M, K> const& b);

// log - это глобальная переменная
template<ofstream & log>
struct FileLogger { ... };
```

Шаблонные параметры — шаблоны

```
// int -> string
string toString( int i );

// работает только с Array<>
Array<string> toStrings( Array<int> const& ar ) {
    Array<string> result(ar.size());
    for (size_t i = 0; i != ar.size(); ++i)
        result.get(i) = toString(ar.get(i));
    return result;
}

// от контейнера требуются: конструктор от size, методы size() и get()
template<template <class> class Container>
Container<string> toStrings(Container<int> const& c) {
    Container<string> result(c.size());
    for (size_t i = 0; i != c.size(); ++i)
        result.get(i) = toString(c.get(i));
    return result;
}
```

Использование зависимых имён

```
template<class T>
struct Array {
    typedef T value_type;
    ...
private:
    size_t  size_;
    T *     data_;
};

template<class Container>
bool contains(Container const& c,
              typename Container::value_type const& v);

int main()
{
    Array<int> a(10);
    contains(a, 5);
    return 0;
}
```

Компиляция шаблонов

- Шаблон независимо компилируется для каждого значения шаблонных параметров.
- Компиляция (*инстанциирование*) шаблона происходит в точке первого использования — *точке инстанциирования шаблона*.
- Компиляция шаблонов классов — ленивая, компилируются только те методы, которые используются.
- В точке инстанциирования шаблон должен быть полностью определён.
- Шаблоны следует определять в заголовочных файлах.
- Все шаблонные функции (свободные функции и методы) являются `inline`.
- В разных единицах трансляции инстанциирование происходит независимо.

Резюме про шаблоны

- Большие шаблонные классы следует разделять на два заголовочных файла: объявление (`array.hpp`) и определение (`array_impl.hpp`).
- Частичная специализация и шаблонные параметры по умолчанию есть только у шаблонов классов.
- Вывод шаблонных параметров есть только у шаблонов функций.
- Предпочтительно использовать перегрузку шаблонных функций вместо их полной специализации.
- Полная специализация функций — это обычные функции.
- Виртуальные методы, конструктор по умолчанию, конструктор копирования, оператор присваивания и деструктор не могут быть шаблонными.
- Используйте `typedef` для длинных шаблонных имён.