Программирование на языке C++
Лекция 6

Ещё о шаблонах

Александр Смаль

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 { ... };
```

$A_{N \times M} * A_{M \times K} = A_{N \times K}$

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(ar.size());
    for (size_t i = 0; i != ar.size(); ++i)
        result.get(i) = toString(ar.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;
}
```

Array<int>::value_type

int

Array<int>

int

Использование функций для вывода параметров

```
template<class First, class Second>
struct Pair {
  Pair(First const& first, Second const& second)
      : first(first), second(second) {}
  First first;
  Second second;
};

template<class First, class Second>
Pair<First, Second> makePair(First const& f, Second const& s) {
    return Pair<First, Second>(f, s);
}

void foo(Pair<int, double> const& p);

void bar() {
  foo(Pair<int, double>(3, 4.5));
  foo(makePair(3, 4.5));
}
```

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.