Шаблоны, часть 2

Александр Смаль

Академический университет
22 ноября 2013
Санкт-Петербург

Полная специализация шаблонов: классы

```
template<class T>
struct Array {
    explicit Array(size_t size)
        : size_(size)
        , data_(new T[size_])
    {}
    T operator[](size_t i) const { return data_[i]; }
private:
    size_t  size_;
    T   *   data_;
};

template<>
struct Array<bool> {
    explicit Array(size_t size)
        : size_(size / sizeof(int) / 8 + 1)
        , data_(new int[size_])
    {}
    bool operator[](size_t i) const
    { return data_[i/sizeof(int)/8] & (1<<(i%(sizeof(int)*8))); }
private:
    size_t  size_;
    int *   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 функций и перегрузка

```
template<class T>
void foo(T a, T b)
{
    printf("1\n");
}

template<class T, class V>
void foo(T a, V b)
{
    printf("2\n");
}

template<>
void foo<int, int>(int a, int b)
{
    printf("3\n");
}

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

Частичная специализация шаблонов

```
template<class T>
struct Array {
    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_;
};

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

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

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

template<class T, template <class> class Container>
void print(Container<T> & c);

template<FILE * f>
struct Logger { ... };
```

Использование зависимых имён

```
template<class T>
struct Array {
    explicit Array(size_t size)
        : size_(size)
        , data_(new T [size_])
    {}
    typedef T value_type;
    T & operator[](size_t i) { return data_[i]; }
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;
}
```

Использование функций для вывода параметров

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

int main()
{
    Pair<int, double> p;
    p = makePair(3, 4.5);
    return 0;
}
```

Резюме про шаблоны

1. Компиляция происходит в точке первого использования — *инстанциирование шаблона*.
2. Компиляция шаблонов ленивая — компилируются только те методы, которые используются.
3. В точке инстанциирования шаблон должен быть полностью известен.
4. Шаблоны следует определять в заголовочных файлах. Большие классы следует разделять на два заголовочных файла: описание (`array.hpp`) и реализацию (`arrayimpl.hpp`).
5. Частичная специализация есть только у классов.
6. Специализация шаблонных функций не участвует в перегрузке.
7. Все шаблонные функции (свободные функции и методы) — `inline`.
8. Специализации функций — обычные функции.

NB: Не забывайте про typedef и typename =).

Вопрос для проверки

Вопрос: может ли шаблонный метод быть виртуальным?

Использование шаблонных параметров — шаблонов

```
// int --> string
string toString( int i ) {
    char buff[128];
    itoa(i, buff, 10);
    return string(buff);
}

// works only with 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));
}

//works with any Container having .size() and .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));
}
```