Шаблоны, часть 1

Александр Смаль

Академический университет
15 ноября 2013
Санкт-Петербург

Проблема "одинаковых классов"

```
struct ArrayInt
{
explicit Array(size_t size)
    : data_(new int[size])
    , size_(size)
{}

~ArrayInt()
{ delete [] data_; }

size_t size() const
{ return size_; }

int operator[](size_t i) const
{ return data_[i]; }

int & operator[](size_t i)
{ return data_[i]; }
...

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

```
struct ArrayFlt
{
explicit ArrayFlt(size_t size)
    : data_(new float[size])
    , size_(size)
{}

~ArrayFlt()
{ delete [] data_; }

size_t size() const
{ return size_; }

float operator[](size_t i) const
{ return data_[i]; }

float & operator[](size_t i)
{ return data_[i]; }
...

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

Путь решения: макросы

```
#define DEFINE_ARRAY(Name, Type)\
struct Name                     \
{                               \
explicit Name(size_t size)      \
    : data_(new Type[size])     \
    , size_(size)               \
{}                              \
                                \
~Name()                         \
{ delete [] data_; }            \
                                \
size_t size() const             \
{ return size_; }               \
                                \
Type operator[](size_t i) const \
{ return data_[i]; }            \
                                \
Type & operator[](size_t i)     \
{ return data_[i]; }            \
...                             \
                                \
private:                        \
    Type *  data_;              \
    size_t  size_;              \
}
```

```
DEFINE_ARRAY(ArrayInt, int);
DEFINE_ARRAY(ArrayFlt, float);

int main()
{
    ArrayInt ai(10);
    ArrayFlt af(20);
    ...
    return 0;
}
```

Путь решения: шаблоны классов

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

~Array()
{ delete [] data_; }

size_t size() const
{ return size_; }

Type operator[](size_t i) const
{ return data_[i]; }

Type & operator[](size_t i)
{ return data_[i]; }
...

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

```
int main()
{
    Array<int> ai(10);
    Array<float> af(20);
    return 0;
}
```

Array<Array<int>>

<int>

Array<int>

instance

Array<int>(10),

Шаблоны классов

```
template <class Type = int
          class SizeT = size_t,
          class CRet = Type>
struct Array
{
explicit Array(SizeT size)
    : data_(new Type[size])
    , size_(size)
{}

~Array() { delete [] data_; }

SizeT size() const
{ return size_; }

CRet operator[](SizeT i) const
{ return data_[i]; }

Type & operator[](SizeT i)
{ return data_[i]; }
...

private:
    Type *  data_;
    SizeT   size_;
};
```

V = X<Type>

Array<> ai(10)

```
void foo()
{
    Array<int> ai(10);
    Array<float> af(20);
    Array<Array<int>,
     size_t,
     ArrayInt<int> const&>
     da(30);
    ...
}
```

da[3][4]

Array<int>

```
typedef Array<int> Ints;
typedef Array<Ints, size_t,
    Ints const &> IInts;

void bar()
{
  [ IInts da(30);
}
```

Array m(10);

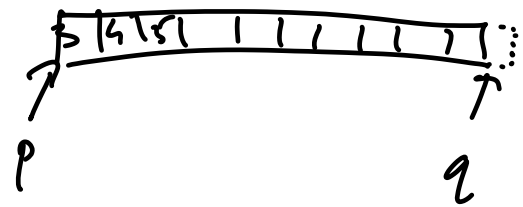
Шаблоны функций: пример 1

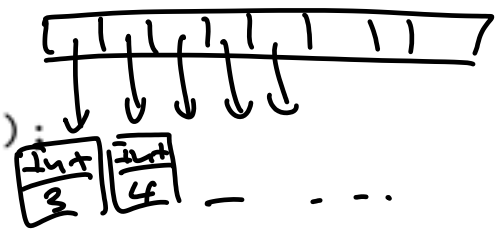
```
// C
int   squarei(int a)      { return a * a; }
float squaref(float f )   { return f * f; }

// C++
int   square(int a)       { return a * a; }
float square(float f )    { return f * f; }

// C++ + OOP
struct INumber {
    virtual  INumber * square() const = 0;
    virtual ~INumber(){}
};
struct Int   : INumber { ... };
struct Float : INumber { ... };

INumber * square(INumber * n) { return n->square(); }

// C++ + templates
template <typename Num>
Num square(Num n) { return n * n; }
```

int a = square<int>(3)

Шаблоны функций: пример 2

```
// 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);
```





(*p) < (*q)

Вывод аргументов (deduce)

NB: у шаблонных функций нет параметров по умолчанию.

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

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

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

void foo() {
    int a = square<int>(3);

    int b = square(a) + square(4); //square<int>(..)

    float m[10];
    sort(m, m + 10); //sort<float>(m, m + 10)

    // error: sort<float> vs. sort<int>
    sort(m, &a);
    Array<double> ad(100);
    sort(ad); // sort<double>(ad)
}
```

float * int *

int square(int);

struct A {};
struct B: A {};

B * → A *
A * a;
B * b;

sort(a, b);

sort<A>(a, b);

Шаблоны методов

```
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];
}

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

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

Array<int> ai(10)

Array<float> af(10)

af = ai;

Компиляция шаблонов

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. Следовательно шаблоны следует определять в заголовочных файлах.
5. В разных единицах трансляции инстанциирование происходит независимо.
6. Все шаблонные функции (свободные функции и методы) — `inline`.
7. Большие классы следует разделять на два заголовочных файла: описание (`array.hpp`) и реализацию (`arrayimpl.hpp`).

Компиляция шаблонов

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. Следовательно шаблоны следует определять в заголовочных файлах.
5. В разных единицах трансляции инстанциирование происходит независимо.
6. Все шаблонные функции (свободные функции и методы) — `inline`.
7. Большие классы следует разделять на два заголовочных файла: описание (`array.hpp`) и реализацию (`arrayimpl.hpp`).

NB: Не забывайте про typedef =).