

Технология ООП

Санкт-Петербургский государственный политехнический университет

11 октября 2011

Цель введения стандартных имен типов

Каждый контейнер:

- с помощью `typedef` дает стандартные имена используемым типам
- определяет эти типы своим способом, наиболее подходящим для реализации

Следовательно, все псевдонимы во внешний мир передаются одинаково.

Пример псевдонима в классе MyVector.

```
template<typename T> class MyVector {  
    typedef T value_type;  
};  
  
int main()  
{  
    int k1;  
    MyVector<int >::value_type k2;  
}
```

Псевдонимы контейнеров STL

<code>value_type</code>	<code>//тип элемента контейнера</code>
<code>allocator_type</code>	<code>//тип распределителя памяти</code>
<code>size_type</code>	<code>//тип индексов (эквивалент</code> <code>sizeof() элемента)</code>
<code>difference_type</code>	<code>//тип результата вычитания</code> <code>адресов двух элементов</code>
<code>iterator</code>	<code>//value_type*</code>
<code>const_iterator</code>	<code>//const value_type*</code>
<code>reverse_iterator</code>	<code>//value_type* в обратном</code> <code>порядке</code>
<code>const_reverse_iterator</code>	<code>//const value_type*</code>
<code>reference</code>	<code>//value_type&</code>
<code>const_reference</code>	<code>//const value_type&</code>

Пример использования псевдонима value_type

Пример псевдонима value_type для нахождения суммы элементов произвольного контейнера.

```
template<typename C> typename C::value_type sum( C&
    c)
// typename обязательно
{
    /*typename*/ C::value_type s = C::value_type();
    // typename необязательно
    /*typename*/ C::iterator it = c.begin();
    while(it != c.end() )
    {
        s += *it;
        ++it;
    }
    return s;
}
```

Пример использования псевдонима value_type

```
int main()
{
    MyVector<int> v(10,1);
    int res = sum(v); //
    MyVector<double> v1(10,1.1);
    double res1 = sum(v1); //
    MyVector<MyString> v2(10, MyString("A"));
    MyString res2 = sum(v2); //
}
```

Шаблонные методы класса

- Обычные методы (в частности, конструктор) могут быть шаблонными, базирующимися на другом типе параметра.
- Виртуальные методы не могут быть шаблонными.

Пример использования шаблонных методов класса

```
template <class T, int size> class MyArray
{
    T m_ar[size];
    public:
    void copy(T* p, size_t num)
    {
        int n = (num < size) ? num : size;
        for(int i = 0; i < n; i++) { m_ar[i] = p[i]; }
    }
};

int main() {
    MyArray<int, 5> a;
    a[0] = 5.5; //OK
    int iar[20] = {4, 7, 8};
    a.copy(iar, 20); //OK
    double dar[10] = {1.1, 2.2, 3.3, ...};
    a.copy(dar, 10); //ошибка
}
```

Пример использования шаблонных методов класса

```
template <class T, int size> class MyArray
{
    T m_ar[size];
    public:

    template<typename Other> void copy(Other* p,
        size_t num)
    {
        int n = (num<size) ? num : size;
        for(int i = 0; i<n; i++) { m_ar[i] = p[i];}
    }
};
```

Виртуальные методы шаблонных классов

- шаблоны могут участвовать в наследовании (при этом перемежаться с нешаблонными классами)
- методы шаблонных классов (не шаблонные) могут быть виртуальными

Виртуальные методы шаблонных классов

```
template<typename T> class A{
    T m_a;
    public: A(const T& a){m_a = a;}
    virtual void f(){m_a++;}
};

template<typename T> class B: public A<T>{
    T m_b;
    public: B(const T& a, const T& b):A<T>(a) {m_b
        = b;}
    virtual void f(){m_b++;}
};

int main()
{
    B<int>* pB = new B<int>(1,5);
    A<int>* pA = new B<int>(1,5);
    pA->f();
    pB->f();
}
```

```
class A{  
    public:  
    class B{  
  
    };  
};  
template<typename T> class C{  
    typename T::B* p;  
};  
C<A> ccc;
```

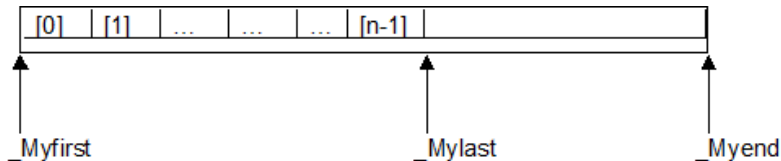
Специализация шаблонного класса

```
template<> class MyArray<int,10>;  
template class MyArray<int,10>;  
template int& MyArray<int,10>::operator [] ( int );
```

```
template<class _Ty,  
class _Ax = allocator<_Ty> >  
class vector  
: public _Vector_val<_Ty, _Ax>  
{...};
```

Распределитель памяти (allocator) — отвечает за выделение, освобождение, перераспределение памяти. Для специфических задач программисту предоставляется возможность реализовать собственный алокатор. Класс `_Vector_val` содержит встроенный объект распределителя памяти (алокатора).

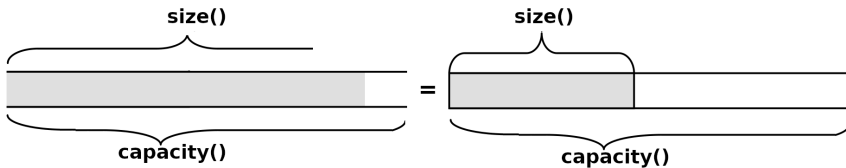
Внутреннее устройство vector



Размеры vector:

- `size()` //???
- `capacity()` //???

Оптимизации операций



```
vector( );  
explicit vector( size_type _Count );  
vector( size_type _Count, const Type& _Val );  
    vector( const vector& _Right );  
template<class InputIterator> vector( InputIterator  
    _First, InputIterator _Last );  
vector( vector&& _Right );
```

Методы, связанные с размером

```
size_type size() const;  
size_type capacity () const;  
void reserve(size_type n);  
void resize(size_type n);  
void resize(size_type n, T& x);  
size_type max_size() const;  
bool empty() const;
```

Примеры использования методов управления размером

```
vector<int> v;  
v.resize(10);  
size_t n = v.size(); //
```

```
vector<int> v;  
v.reserve(10);  
size_t n = v.size(); //
```

Получение/модификация значений элементов

```
reference at(size_type pos);  
reference operator [] (size_type pos);  
reference front();  
reference back();
```

Получение/модификация значений элементов

```
vector<int> v(10,1);  
//Распечатать значения элементов  
//???
```

```
//Получить/изменить значение i-ого элемента  
//???
```

Вставка/удаление последнего элемента

```
void push_back(const T& x);  
void pop_back();
```

```
vector<int> v;  
v.resize(10);  
v.push_back(1);  
size_t n = v.size(); //
```

```
vector<int> v;  
v.reserve(10);  
v.push_back(1);  
size_t n = v.size(); //
```

Нет операций с началом последовательности: `push_front()`,
`pop_front()`

Использование итераторов

```
iterator begin();  
const_iterator begin() const;  
iterator end();  
iterator end() const;  
reverse_iterator rbegin();  
const_reverse_iterator rbegin() const;  
reverse_iterator rend();  
const_reverse_iterator rend() const;
```

Использование итераторов

Пример: прохождение последовательности в обратном порядке с помощью “прямого” итератора

```
char ar[] = "QWERTY";
```

```
//создать вектор, элементы которого должны быть  
копиями символов массива
```

```
//создать итератор для вектора
```

```
//вывести значения элементов вектора с помощью  
итератора
```

Использование итераторов

```
char ar[] = "QWERTY";  
vector<char> v(ar, ar+sizeof(ar) -1);  
vector<char>::iterator it=v.end();  
while(it != v.begin())  
{  
    it--;  
    cout<<*it<<"_";  
}  
cout<<endl;
```

Использование итераторов

Пример: прохождение последовательности в обратном порядке с помощью “реверсивного” итератора

```
char ar[] = "QWERTY";  
vector<char> v(ar, ar+sizeof(ar) -1);  
vector<char>::reverse_iterator rit=v.rbegin();  
while(rit!=v.rend())  
{  
    cout<<*rit<<" ";  
    rit++; //!!!!  
}  
cout<<endl;
```

```
class A{ int m_a;  
public: A(int a=0){m_a = a;}  
int GetA(){return m_a;}  
};  
int main()  
{  
    vector<A> v(10, A(1));  
    vector<A>::iterator it = v.begin();  
    while(it != v.end())  
    {  
        std::cout<< (*it).GetA();  
        std::cout<< it->GetA();  
        ++it;  
    }  
}
```

```
void assign(const_iterator first , const_iterator  
           last);
```

```
void assign(size_type n, const T& x);
```

```
vector<int> v1(10,1);
```

```
vector<int> v2(11,12);
```

```
int ar[] = {5,7,1,-6, ...};
```

```
//заменить элементы v1 на элементы v2
```

```
//заменить элементы v1 на элементы ar
```

```
iterator insert(iterator it, const T& x);  
void insert(iterator it, size_type n, const T& x);  
void insert(iterator it, const_iterator first,  
            const_iterator last);
```

Пример использования оператора вставки

```
vector<int> v;  
//формируем значения элементов таким образом, чтобы  
    содержимое вектора стало 1,2,3,4  
  
//требуется вставить между элементами значение 33 –  
    1,33,2,33,3,33,4
```

Пример использования оператора вставки

```
vector<int>::iterator it =v.begin();
int size = v.size();
for(int i=0; i<size; i++ )
{
    ++it;
    it = v.insert(it , 33);
    ++it;
}
```

```
iterator erase(iterator it);  
iterator erase(iterator first , iterator last);  
void clear();
```

Требуется удалить все 33 из полученного вектора v.

Пример использования оператора удаления

```
vector<int>::iterator it =v.begin();
while(it != v.end())
{
    if(*it==33) {
        it = v.erase(it);
    }
    else {
        ++it;
    }
}
```

Уменьшение емкости вектора

```
void shrink_to_fit( );
```

Обмен двух векторов местами:

```
void swap(vector<Type, Allocator>& x);
```

Пример создание двумерных массивов

```
vector< vector<int> > vv;
```

```
typedef vector<int> VECT_INT;  
vector<VECT_INT> vv;
```

В отличие от обычного массива строки такого вектора могут быть разной длины.

Пример использования класса vector

```
vector< vector<int> > vv1(10, vector<int>(10,1));  
//  
vector< vector<int> > vv2; //  
vv2[2][5] = 1; //  
vv2.at(2).at(5) = 1; //  
//  
int ar[10][3] = {{1,2,3},{4,5,6}},};  
//Создать вектор векторов таким образом, чтобы  
элементы каждого вектора стали копиями  
элементов соответствующей строки массива
```

Если вектор содержит указатели

```
{  
vector<MyString*> v;  
v.push_back(new MyString("AAA"));  
v.push_back(new MyString("BBB"));  
  
//Печать  
  
}///???
```
