

Объектно-ориентированное программирование на алгоритмическом языке C++

МИРЭА, Институт Информационных
технологий, кафедра Вычислительной техники

Контейнеры

Контейнеры – это объекты, которые содержат другие объекты

Контейнеры – шаблонные классы для хранения, доступа и работе с данными

bitset	Множество битов	<bitset>
deque	Двусторонняя очередь	<deque>
list	Линейный список	<list>
map	Ассоциативный список для хранения пар ключ/значение, где с каждым ключом связано только одно значение	<map>
multimap	Ассоциативный список для хранения пар ключ/значение, где с каждым ключом связано два или более значений	<map>
multiset	Множество, в котором каждый элемент не обязательно уникален	<set>
priority_queue	Очередь с приоритетом	<queue>
queue	Очередь	<queue>
set	Множество, в котором каждый элемент уникален	<set>
stack	Стек	<stack>
vector	Динамический массив	<vector>

Итераторы

Итераторы – это объекты, которые действуют подобно указателям и реализуют инструменты доступа к элементам контейнеров

Итератор	Описание
Произвольного доступа (random access)	Используется для считывания и записи значений. Доступ к элементам произвольный
Двунаправленный (bidirectional)	Используется для считывания и записи значений. Может проходить контейнер в обоих направлениях
Однонаправленный (forward)	Используется для считывания и записи значений. Может проходить контейнер только в одном направлении
Ввода (input)	Используется только для считывания значений. Может проходить контейнер только в одном направлении
Вывода (output)	Используется только для записи значений. Может проходить контейнер только в одном направлении

Векторы

Класс **vector** (или просто «вектор») — это динамический массив, способный увеличиваться по мере необходимости для содержания всех своих элементов. Класс **vector** обеспечивает произвольный доступ к своим элементам через оператор индексирования [], а также поддерживает вставку и удаление элементов.

```
#include <vector>
```

```
vector < тип > «имя вектора»
```

```
vector < тип > :: iterator «имя итератора»
```



Некоторые методы вектора

Метод	Описание
<code>at (i)</code>	Возвращает ссылку на элемент, заданный параметром <code>i</code>
<code>back ()</code>	Возвращает ссылку на последний элемент вектора
<code>begin ()</code>	Возвращает итератор первого элемента вектора
<code>capacity ()</code>	Возвращает текущую емкость вектора
<code>clear ()</code>	Удаляет все элементы вектора
<code>empty ()</code>	Возвращает истинное значение если вектор пуст, иначе ложь
<code>end ()</code>	Возвращает итератор для конца вектора
<code>erase (iterator it)</code>	Удаляет элемент, на который указывает итератор <code>it</code> . Возвращает итератор элемента, который расположен следующим за удаленным
<code>erase (iterator start, iterator end)</code>	Удаляет элементы, заданные между итераторами <code>start</code> и <code>end</code> . Возвращает итератор элемента, который расположен следующим за последним удаленным
<code>front ()</code>	Возвращает ссылку на первый элемент вектора
<code>insert (iterator it, const T & val = T ())</code>	Вставляет параметр <code>val</code> перед элементом, заданным итератором <code>it</code> . Возвращает итератор элемента
<code>pop_back ()</code>	Удаляет последний элемент вектора
<code>push_back (const T & val)</code>	Добавляет в конец вектора элемент, значение которого равно параметру <code>val</code>

Пример использования класса vector

```
#include <iostream>
#include <vector>
using namespace std;
int main(){
    vector<int> vect;
    for (int count=0; count < 5; ++count)
        vect.push_back(10 - count); // вставляем числа в конец массива
    for (int index=0; index < vect.size(); ++index)
        cout << vect[index] << ' ';
    cout << '\n';
    return 0;
}
```

Результат выполнения программы

выше:

10 9 8 7 6

Пример использования вектора

```
#include <iostream>
#include <vector>
using namespace std;
int main(){
    vector<int> myVector;
    for (int count=0; count < 5; ++count)
        myVector.push_back(count);
    vector<int>::iterator it; // объявляем итератор
    it = myVector.begin(); // присваиваем ему начало вектора
    while (it != myVector.end()) { // пока итератор не достигнет конца
        cout << *it << " "; // выводим значение элемента, на который
указывает итератор
        it++; // и переходим к следующему элементу
    }
    cout << "\n";
    return 0;
}
```

Результат выполнения программы

выше:

0 1 2 3 4

Пример использования вектора

```
#include <iostream>
#include <vector>
using namespace std;
int main ( ) {
    vector < char > v, v2;
    unsigned int    i;

    for ( i = 0; i < 10; i++ ) v.push_back ( 'A' + i );
    cout << "Contents of v :\n";
    for ( i = 0; i < v.size (); i++ ) cout << v [ i ] << " ";
    cout << endl << endl;

    char str[] = "-STL Power-";          // инициализация второго вектора
    for ( i = 0; str [ i ]; i++ ) v2.push_back ( str [ i ] );

    vector < char > :: iterator p          = v.begin  () + 5;
    vector < char > :: iterator p2start = v2.begin  ();
    vector < char > :: iterator p2end   = v2.end    ();
    v.insert ( p, p2start, p2end );

    cout << "Contents of v :\n";
    for ( i = 0; i < v.size (); i++ ) cout << v [ i ] << " ";
    return 0;
}
```


Ответ примера

Contents of v :

A B C D E F G H I J

Contents of v :

A B C D E - S T L P o w e r - F G H I J

Класс **deque** (или просто «**дек**») — это двусторонняя очередь, реализованная в виде динамического массива, который может расти с обоих концов.

```
#include <iostream>
#include <deque>
using namespace std;
int main(){
    deque<int> deq;
    for (int count=0; count < 4; count++){
        deq.push_back(count); // вставляем числа в конец массива
        deq.push_front(10 - count); // вставляем числа в начало массива
    }
    for (int index=0; index < deq.size(); ++index)
        cout << deq[index] << ' ';
    cout << '\n';
return 0;
}
```

Результат выполнения
программы:

7 8 9 10 0 1 2 3

Стеки и очереди

```
#include <iostream>
#include <stack>
#include <queue>
using namespace std;

int main(){

    stack <int> st;
    queue <int> q;

    for(int i=0;i<10;i++){
        st.push(i);
        q.push(i);
    }
```

Стеки и очереди

```
    cout<<"очередь stack: ";  
    for(int i=0;i<10;i++){  
        cout<< st.top() <<" ";  
        st.pop();  
    }  
    cout<<endl<<"очередь queue: ";  
    for(int i=0;i<10;i++){  
        cout<< q.front() <<" ";  
        q.pop();  
    }  
    return 0;  
}
```

очередь stack: 9 8 7 6 5 4 3 2 1 0
очередь queue: 0 1 2 3 4 5 6 7 8 9

Вектор объектов

```
#include <iostream>
#include <vector>

using namespace std;
class compl{
    public:
        double Re, Im;
        compl(double r=0, double i=0){Re = r; Im = i;};
        void show();

};

void compl::show(){
    cout<<Re;
    if(Im>=0) cout<<" ";
    cout<<Im << " * i" << endl;
}
```

Вектор объектов

```
int main() {  
    vector <compl> c;  
    for(int i=0;i<5;i++)  
        c.push_back( compl(rand()%10,rand()%10) );  
  
    vector <compl> :: iterator i ;  
    for (i= c.begin();i != c.end(); i++){  
        i->show();  
        //c[i].show(); - не работает  
    }  
    return 0;  
}
```

7+1*
i
0+4*
i
4+9*
i
8+8*
i
4+2*