top
КОМПЬЮТЕРНАЯ АКАДЕМИЯ

ОБЪЕКТНО -ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА C++

Урок №15

Контейнеры в STL

Содержание

Анализ и использование класса vector

Библиотека vector

Класс vector поддерживает динамический массив и счетчик элементов, сохраненных в нем. Спецификация его шаблона имеет следующий вид:

```
template <class T, class Allocator = Allocator<T>>
class vector
```

Здесь **T** — ***тип сохраняемых данных***, а **Allocator** ***задает распределитель***. Класс vector имеет следующие конструкторы:

```
explicit vector(const Allocator &a = Allocator());
explicit vector(size_type num, const T &val = T(),
      const Allocator &a = Allocator());
vector(const vector <T,Allocator> &ob);
template < class InIter> vector(InIter start,
      InIter end, const Allocator &a = Allocator());
```

Первая форма конструктора создает пустой вектор. Вторая создает вектор, который создает num элементов со значением val. Третья создает вектор, который содержит те же элементы, что и вектор ob. Четвертая создает вектор, который содержит элементы в диапазоне, заданном параметрами start и end.

Для класса vector определены следующие операторы сравнения:

- ==
- <
- <=
- !=
- >
- >=

Класс vector содержит следующие функции-члены:

```
template <class InIter> void assign(InIter start,
                                    InIter end);
```

Помещает в вектор последовательность, определяемую параметрами start и end.

```
void assign(size_type num, const T &val);
```

Помещает в вектор num элементов со значением val.

```
reference at(size_type i);
const_reference at(size_type i) const;
```

Возвращает ссылку на элемент, заданный параметром i. При этом, в отличие от перегруженного оператора [] данная функция в случае выхода за пределы массива генерирует исключение.

```
reference back();
const_reference back() const;
```

Возвращает ссылку на последний элемент в векторе.

```
iterator begin();
const_iterator begin() const;
```

Возвращает итератор для первого элемента в векторе.

```
size_type capacity() const;
```

Возвращает текущую ёмкость вектора, которая представляет собой количество элементов, способное храниться в векторе до того, как возникнет необходимость в выделении дополнительной памяти.

```
void clear();
```

Удаляет все элементы из вектора.

```
bool empty() const;
```

Возвращает значение истины, если используемый вектор пуст, и значение лжи в противном случае.

```
const_iterator end() const;
iterator end();
```

Возвращает итератор для конца вектора.

```
iterator erase(iterator i);
```

Удаляет элемент, адресуемый итератором i, возвращает итератор для элемента, расположенного после удаленного.

```
iterator erase(iterator start, iterator end);
```

Удаляет элементы в диапазоне, задаваемом параметрами start и end, возвращает итератор для элемента, расположенного за последним удалённым элементом.

```
reference front();
const_reference front() const;
```

Возвращает ссылку на первый элемент в векторе.

```
allocator_type get_allocator() const;
```

Возвращает распределитель вектора.

```
iterator insert(iterator i, const T &val = T());
```

Вставляет значение val непосредственно перед элементом, заданным параметром i, возвращает итератор для этого элемента.

```
void insert(iterator i, size_type num, const T &val);
```

Вставляет num копий значения val непосредственно перед элементом, заданным параметром i.

```
template <class InIter>
  void insert(iterator i, InIter start, InIter end);
```

Вставляет в вектор последовательность, определяемую параметрами start и end, непосредственно перед элементом, заданным параметром i.

```
size_type max_size() const;
```

Возвращает максимальное число элементов, которое может содержать вектор.

```
reference operator[](size_type i) const;
const_reference operator[](size_type i) const;
```

Возвращает ссылку на элемент, заданный параметром i.

```
void pop_back();
```

Удаляет последний элемент в векторе.

```
void push_back(const T &val);
```

Добавляет в конец вектора элемент со значением, заданным параметром val.

```
reverse_iterator rbegin();
const_reverse_iterator rbegin() const;
```

Возвращает реверсивный итератор для конца вектора.

```
reverse_iterator rend();
const_reverse_iterator rend() const;
```

Возвращает реверсивный итератор для начала вектора.

```
void reverse(size_type num);
```

Устанавливает емкость вектора равной не менее заданного значения num.

```
void resize(size_type num, const T &val = T());
```

Устанавливает емкость вектора равной не менее заданного значения num, если вектор для этого нужно удлинить, то в его конец добавляются элементы со значением, заданным параметром val.

```
size_type size() const;
```

Возвращает текущее количество элементов в векторе.

```
void swap(deque<T,Allocator> &ob);
```

Выполняет обмен элементами данного вектора и вектора ob.

```
void flip();
```

Инвертирует значения всех битов в векторе.

```
static void swap(reference i, reference j);
```

Переставляет местами биты, заданные параметрами i и j.

Пример использования класса vector

```
//Пример: Данный пример показывает приемы работы
//с контейнером vector.
#include <iostream>
#include <vector>
using namespace std;

void main()
{
    //Создаем вектор
    vector<int> vect;

    cout << "\nNumber of elements that could be
         stored in the vector without "
         << "allocating more storage --> "
         << vect.capacity();

    cout << "\n-----------------------------------";
    //используя метод size() получаем текущее кол-во
    //элементов в векторе.
    //cout << "\nThe number of elements in the
           vector --> " << vect.size();

    cout << "\n-----------------------------------";
    vect.resize(4, 0); //изменяем размер, новые
                       //элементы заполнятся нулями
```

```
cout << "\nResizing...\n";
cout << "The number of elements in
       the vector --> " << vect.size() << endl;

cout << "\nvector  -->\t";
for (int i=0; i<vect.size(); i++)
{
      cout << vect[i] << '\t';
}
cout << "\n-----------------------------------";

//максимальный размер вектора.
//Метод max_size() возвращает кол-во байт.
cout << "\nThe maximum possible length of the
       vector --> "
       << vect.max_size()/4;
cout << "\n-----------------------------------";

vect.push_back(1); //вставляем единицу
                   //в конец вектора
cout << "\npush_back\nvector  -->\t";
for (int i=0; i<vect.size(); i++)
{
    cout << vect[i] << '\t';
}
cout << "\n-----------------------------------";
//создаем реверсный итератор и выставлем
//его на конец вектора
vector<int>::reverse_iterator i_riterator =
      vect.rbegin();
cout << "\nreverse_iterator\nvector  -->\t";

//выводим содержимое векртора на экран используя
//реверсный итератор
for (int i=0; i<vect.size(); i++)
{
    cout << *(i_riterator+i) << '\t';
}
```

```
    cout << "\n-----------------------------------";

    //создаем обычный итератор и выставлем его
    //на конец вектора
    vector<int>::iterator i_iterator = vect.end();

    //вставка элемента "-1" перед последним элементом
    vect.insert(i_iterator-1, -1);

    cout << "\ninsert\nvector -->\t";

    //выводим содержимое векртора на экран
    //используя обычный итератор
    for (i_iterator=vect.begin();
                i_iterator!=vect.end(); i_iterator++)
    {
        cout << *(i_iterator) << '\t';
    }
    cout << "\n-----------------------------------";

    i_iterator = vect.end(); //итератор конца вектора
    vect.insert(i_iterator-1, 2, 4); //вставка двух
                                     //четверок перед
                                     //последним
                                     //элементом
    cout << "\ninsert\nvector -->\t";
    for (int i=0; i<vect.size(); i++)
    {
        cout << vect[i] << '\t';
    }
    cout << "\n--------------------------------\n\n";
}

//Программа выводит следующий результат:

//Number of elements that the vector could contain
//without allocating more storage --> 0
```

```
//-------------------------------------------------
//The number of elements in the vector --> 0
//-------------------------------------------------
//Resizing...
//The number of elements in the vector --> 4
//
//vector  -->    0    0    0    0
//-------------------------------------------------
//The maximum possible length of the vector -->
268435455
//-------------------------------------------------
//push_back
//vector  -->    0    0    0    0    1
//-------------------------------------------------
//reverse_iterator
//vector  -->    1    0    0    0    0
//-------------------------------------------------
//insert
//vector -->    0    0    0    0    -1    1
//-------------------------------------------------
//insert
//vector -->    0    0    0    0    -1    4    4    1
//-------------------------------------------------
```

Анализ и использование класса list

Библиотека list

Класс list поддерживает работу двунаправленного связанного списка. Спецификация его шаблона выглядит следующим образом:

```
template <class T, class Allocator = Allocator<T>>
class list
```

Здесь **T** — ***тип данных, сохраняемых в списке***. Класс list имеет следующие конструкторы:

```
explicit list(const Allocator &a = Allocator());
explicit list(size_type num, const T &val = T(),
              const Allocator &a = Allocator());
list(const list <T,Allocator> &ob);
template < class InIter> list(InIter start, InIter end,
             const Allocator &a = Allocator());
```

Первая форма конструктора создает пустой список. Вторая создает список, который содержит num элементов со значением val. Третья создает список, который содержит те же элементы, что и список ob. Четвертая создает список, который содержит элементы в диапазоне, заданном параметрами start и end.

Для класса list определены следующие операторы сравнения:

- ==
- <
- <=

- !=
- >
- >=

Класс list содержит следующие функции-члены:

```
template <class InIter> void assign(InIter start,
                                    InIter end);
```

Помещает в список последовательность, определяемую параметрами start и end.

```
void assign(size_type num, const T &val);
```

Помещает в список num элементов со значением val.

```
reference back();
const_reference back() const;
```

Возвращает ссылку на последний элемент в списке.

```
iterator begin();
const_iterator begin() const;
```

Возвращает итератор для первого элемента в списке.

```
void clear();
```

Удаляет все элементы из списка.

```
bool empty() const;
```

Возвращает значение истины, если используемый список пуст, и значение лжи в противном случае.

```
const_iterator end() const;
iterator end();
```

Возвращает итератор для конца списка.

```
iterator erase(iterator i);
```

Удаляет элемент, адресуемый итератором i, возвращает итератор для элемента, расположенного после удаленного.

```
iterator erase(iterator start, iterator end);
```

Удаляет элементы в диапазоне, задаваемом параметрами start и end, возвращает итератор для элемента, расположенного за последним удалённым элементом.

```
reference front();
const_reference front() const;
```

Возвращает ссылку на первый элемент в списке.

```
allocator_type get_allocator() const;
```

Возвращает распределитель списка.

```
iterator insert(iterator i, const T &val = T());
```

Вставляет значение val непосредственно перед элементом, заданным параметром i, возвращает итератор для этого элемента.

```
void insert(iterator i, size_type num, const T &val);
```

Вставляет num копий значения val непосредственно перед элементом, заданным параметром i.

```
template <class InIter> void insert(iterator i,
                          InIter start, InIter end);
```

Вставляет в список последовательность, определяемую параметрами start и end, непосредственно перед элементом, заданным параметром i.

```
size_type max_size() const;
```

Возвращает максимальное число элементов, которое может содержать список.

```
void merge(list<T,Allocator> &ob);
template <class Comp> void merge(list<T,Allocator>
                                 &ob, Comp cmpfn);
```

Объединяет упорядоченный список, содержащийся в объекте ob, с данным упорядоченным списком. Результат также упорядочивается. После объединения список, содержащийся в объекте ob, остается пустым. Во второй форме может быть задана функция сравнения, которая определяет, когда один элемент меньше другого.

```
void pop_back();
```

Удаляет последний элемент в списке.

```
void pop_front();
```

Удаляет первый элемент в списке.

```
void push_back(const T &val);
```

Добавляет в конец списка элемент со значением, заданным параметром val.

```
void push_front(const T &val);
```

Добавляет в начало списка элемент со значением, заданным параметром val.

```
reverse_iterator rbegin();
const_reverse_iterator rbegin() const;
```

Возвращает реверсивный итератор для конца списка.

```
reverse_iterator rend();
const_reverse_iterator rbegin() const;
```

Возвращает реверсивный итератор для начала списка.

```
void remove(const T &val);
```

Удаляет из списка элементы со значением, заданным параметром val.

```
template <class UnPred> void remove_if(UnPred pr);
```

Удаляет элементы, для которых унарный предикат pr равен значению true.

```
void resize(size_type num, const T &val = T());
```

Устанавливает емкость списка равной не менее заданного значения num, если вектор для этого нужно удлинить, то в его конец добавляются элементы со значением, заданным параметром val.

```
void reverse();
```

Реверсирует список.

```
size_type size() const;
```

Возвращает текущее количество элементов в списке.

```
void sort();
template <class Comp> void sort(Comp cmpfn);
```

Сортирует список. Вторая форма сортирует список с помощью функции сравнения cmpfn, чтобы определять, когда один элемент меньше другого.

```
void splice(iterator i, list<T,Allocator> &ob);
```

Вставляет содержимое списка ob в данный список в позиции, указанной итератором i. После выполнения этой операции список ob остается пустым.

```
void splice(iterator i, list<T,Allocator> &ob,
            iterator el);
```

Удаляет из списка ob элемент, адресуемый итератором el, и сохраняет его в позиции, адресуемой итератором i.

```
void splice(iterator i, list<T,Allocator> &ob,
            iterator start, iterator end);
```

Удаляет из списка ob диапазон, определяемый параметрами start и end, и сохраняет его в данном списке, начиная с позиции, адресуемой итератором i.

```
void swap(list<T,Allocator> &ob);
```

Выполняет обмен элементами данного списка и списка ob.

```
   void unique();
   template <class BinPred> void unique(BinPred pr);
```

Удаляет из списка элементы-дубликаты. Вторая форма для определения уникальности использует предикат pr.

Пример использования класса list

```
//Пример: Принципы работы со списками:
//Создание, заполнение, сортировка,
//вывод списка на экран.
//Принципы работы с итераторами.

#include <iostream>
#include <list>
using namespace std;

typedef list<int> ourList;

void ShowLists (ourList& l1, ourList& l2)
{
    //Создаем итератор.
    ourList::iterator iter;

    cout << "list1: ";
    for (iter = l1.begin(); iter != l1.end(); iter++)
    {
        //выводим элемент на который указывает итератор
        cout << *iter << " ";
    }

    cout << endl << "list2: ";
    for (iter = l2.begin(); iter != l2.end(); iter++)
    {
        cout << *iter << " ";
    }
    cout << endl << endl;
}
```

```
void main()
{
    //Создание двух пустых списков
   ourList list1, list2;

    //Заполнение обоих списков элементами
    for (int i=0; i<6; ++i)
   {
        list1.push_back(i);
        list2.push_front(i);
    }
    //вывод списков на экран
    ShowLists(list1, list2);

    //Во втором списке перемещение
    //первого элемента в конец
    list2.splice(list2.end(),  //Позиция в приемник
        list2,                 //Источник
        list2.begin());        //Позиция в источнике

    //"переворачиваем" первый список
    list1.reverse();
    ShowLists(list1, list2);

    //Сортировка обоих списков
    list1.sort();
    list2.sort();
    ShowLists(list1, list2);

    //Сливаем два отсортированных списка
    //в первый списком
    list1.merge(list2);
    ShowLists(list1, list2);

    //удаляем дубликаты из первого списка
    list1.unique();
    ShowLists(list1, list2);
}
```

```
//Программа выводит следующий результат:

//list1:  0 1 2 3 4 5
//list2:  5 4 3 2 1 0

//list1:  5 4 3 2 1 0
//list2:  4 3 2 1 0 5

//list1:  0 1 2 3 4 5
//list2:  0 1 2 3 4 5

//list1:  0 0 1 1 2 2 3 3 4 4 5 5
//list2:

//list1:  0 1 2 3 4 5
//list2:
```

Анализ и использование класса map

Библиотека map

Класс map поддерживает поддерживает ассоциативный контейнер, в котором уникальным ключам соответствуют определённые значения. Спецификация его шаблона имеет следующий вид:

```
template <class Key, class T, class Comp = less<key>,
class Allocator =Allocator<pair<const key, T>>>
class map
```

Здесь **key** — ***тип данных ключей***,**T** — ***тип сохраняемых (отображемых)значений***, а **Comp** — ***функция, которая сравнивает два ключа***. Класс map имеет следующие конструкторы:

```
explicit map(const Comp &cmpfn = Comp(),
             Allocator &a = Allocator());
map(map<Key, T, Comp,Allocator> &ob);
template < class InIter> map(InIter start,
             InIter end,const Comp &cmpfn = Comp(),
             const Allocator &a = Allocator());
```

Первая форма конструктора создает пустое отображение. Вторая создает отображение, которое содержит те же элементы, что и отображение ob. Третья создает отображение, которое содержит элементы в диапазоне, заданном параметрами start и end. Функция, заданная параметром cmpfn (и если она задана), определяет упорядочение отображения.

Для класса map определены следующие операторы сравнения:

- ==
- <
- <=
- !=
- >
- >=

Класс map содержит перечисленные ниже функции-члены. В приведенных описаниях элемент key_type представляет тип ключа, а элемент value_type — пару элементов pair<Key, T>.

```
iterator begin();
const_iterator begin() const;
```

Возвращает итератор для первого элемента в отображении.

```
void clear();
```

Удаляет все элементы из отображения.

```
size_type count(const key_type &k) const;
```

Возвращает число вхождений ключа k в отображении (1 или 0).

```
size_type count(const key_type &k) const;
```

Возвращает значение true, если данное отображение пустое, и false в противном случае.

```
const_iterator end() const;
iterator end();
```

Возвращает итератор, указывающий на конец отображения.

```
pair<iterator, iterator> equal_range(const key_type &k);
pair<const_iterator, const_iterator>
     equal_range(const key_type &k) const;
```

Возвращает пару итераторов, которые указывают на первый и последний элементы в отображении, содержащие заданный ключ.

```
void erase(iterator i);
```

Удаляет элемент, адресуемый итератором i.

```
void erase(iterator start, iterator end);
```

Удаляет элементы в диапазоне, задаваемом параметрами start и end.

```
size_type erase(const key_type &k);
```

Удаляет из отображения элементы, ключи которых имеют значение k.

```
iterator find(const key_type &k);
const_iterator find(const key_type &k) const;
```

Возвращает итератор для заданного ключа. Если ключ не обнаружен, возвращает итератор до конца отображения.

```
allocator_type get_allocator() const;
```

Возвращает распределитель отображения.

```
iterator insert(iterator i, const value_type &val);
```

Вставляет значение val после элемента, заданным итератором i, возвращает итератор для этого элемента.

```
template <class InIter> void insert(InIter start,
                                    InIter end);
```

Вставляет элементы заданного диапазона.

```
pair<iterator, bool> insert(const value_type &val);
```

Вставляет значение val в используемое отображение. Возвращает итератор для данного отображения. Элемент вставляет только в том случае, если его еще нет в отображении. Если элемент был вставлен возвращает пару pair<iterator, true>, в противном случае pair<iterator, false>.

```
key_compare key_comp() const;
```

Возвращает объект-функцию, которая сравнивает ключи.

```
iterator lower_bound(const key_type &k);
const_iterator lower_bound(const key_type &k) const;
```

Возвращает итератор для первого элемента в отображении, ключ которого равен значению k или больше этого значения.

```
size_type max_size() const;
```

Возвращает максимальное число элементов, которое может содержать отображение.

```
reference operator[](const key_type &i);
```

Возвращает ссылку на элемент, заданный параметром i. Если этого элемента не существует, вставляет его в отображение.

```
reverse_iterator rbegin();
const_reverse_iterator rbegin() const;
```

Возвращает реверсивный итератор для конца отображения.

```
reverse_iterator rend();
const_reverse_iterator rend() const;
```

Возвращает реверсивный итератор для начала отображения.

```
size_type size() const;
```

Возвращает текущее количество элементов в отображении.

```
void swap(map<Key, T, Comp, Allocator> &ob);
```

Выполняет обмен элементами данного отображения и отображения ob.

```
iterator upper_bound(const key_type &k);
const_iterator upper_bound(const key_type &k) const;
```

Возвращает итератор для первого элемента в отображении, ключ которого больше заданного значения k.

```
value_compare value_comp() const;
```

Возвращает объект-функцию, которая сравнивает значения.

Этот класс предназначен для быстрого поиска значения по ключу. В качестве ключа может быть использовано все, что угодно, но при этом необходимо помнить, что главной особенностью ключа является возможность применить к нему операцию сравнения. Быстрый поиск значения по ключу осуществляется благодаря тому, что пары хранятся в отсортированном виде. Этот класс имеет недостаток — скорость вставки новой пары обратно пропорциональна количеству элементов, сохраненных в классе. Еще один важный момент — ключ должен быть уникальным.

Пример использования класса map

```
//Пример: данный пример демонстрирует методы
//работы со отображениями

#include <iostream>
#include <map>
#include <vector>
using namespace std;

void main()
{
```

```
    //создаем отображение
    map <int, int> our_map;

    //создаем вектор
    vector <int> our_vector;

    //максимальный размер вектора
    cout << "\n\nmax size of vector --> "
         << our_vector.max_size() / sizeof(int);

    //максимальный размер отображения
    //(в два раза меньше т.к. для каждого элемента
    //нужно хранить два значения - пару).
    cout << "\n\nmax size of map --> "
         << our_map.max_size() / sizeof(int);

    cout << "\n\n-------------------------------\n";

    int val;
    int key;

    cout << "\nInput value : ";
    cin >> val;

    cout << "\nInput key : ";
    cin >> key;

    //создаем пару на основании двух значений.
    pair<int, int> element(key, val);

    //вставляем пару в отображение
    our_map.insert(element);

    //кол-во элементов в отображении
    cout << "\nCurrent element of map --> "
         << our_map.size() << endl;

   cout << "\n\n-------------------------------\n";
```

```
    cout << "\nInput value : ";
    cin >> val;

    cout << "\nInput key : ";
    cin >> key;

    pair<map<int, int>::iterator, bool>
         err = our_map.insert(make_pair(key, val));

    if (err.second == false)
    {
        //отработает в случае если в отображение
        //не получилось добавить элемент
        //например если в отображении уже был
        //элемент с данным ключом.
        cout << "\nError !!!\n";
    }
    //кол-во элементов в отображении
    cout << "\nCurrent element of map --> "
         << our_map.size() << endl;

    //вывод всех элементов на экран
    map<int, int>::iterator iter = our_map.begin();
    for (; iter != our_map.end(); iter++)
    {
        cout << "\nKey --> " << iter->first
             << "\t\tValue --> " << iter->second;
    }
    cout << "\n---------------------------------\n";
}
```

Анализ и использование класса multimap

Библиотека multimap

Модифицированный вариант map, в котором отсутствует требования уникальности ключа — то есть, если произвести поиск по ключу, то вернется не одно значение, а набор значений, сохраненных с данным ключом.

Класс multimap поддерживает поддерживает ассоциативный контейнер, в котором неуникальным (в общем случае) ключам соответствуют определённые значения. Спецификация его шаблона имеет следующий вид:

```
template <class Key, class T, class Comp = less<key>,
class Allocator =Alloctor<pair<const key, T>>>
class multimap
```

Здесь **key** — ***тип данных ключей***, **T** — ***тип сохраняемых (отображемых)значений***, а **Comp** — ***функция, которая сравнивает два ключа***. Класс **multimap** имеет следующие конструкторы:

```
explicit multimap(const Comp &cmpfn =
          Comp(),Allocator &a = Allocator());

multimap(multimap<Key, T, Comp,Allocator> &ob);

template < class InIter> multimap(InIter start,
          InIter end,const Comp &cmpfn = Comp(),
          const Allocator &a = Allocator());
```

Первая форма конструктора создает пустое мультиотображение. Вторая создает мультиотображение, которое содержит те же элементы, что и мультиотображение ob. Третья создает мультиотображение, которое содержит элементы в диапазоне, заданном параметрами start и end. Функция, заданная параметром cmpfn (и если она задана), определяет упорядочение мультиотображения.

Для класса multimap определены следующие операторы сравнения:

- ==
- <
- <=
- !=
- >
- >=

Класс multimap содержит перечисленные ниже функции-члены. В приведенных описаниях элемент key_type представляет тип ключа, а элемент value_type — пару элементов pair<Key, T>.

```
iterator begin();
const_iterator begin() const;
```

Возвращает итератор для первого элемента в мультиотображении.

```
void clear();
```

Удаляет все элементы из мультиотображения.

```
size_type count(const key_type &k) const;
```

Возвращает число вхождений ключа k в мультиотображении (1 или 0).

```
bool empty() const;
```

Возвращает значение true, если данное мультиотображение пустое, и false в противном случае.

```
const_iterator end() const;
iterator end();
```

Возвращает итератор, указывающий на конец мультиотображения.

```
pair<iterator, iterator> equal_range(const key_type &k);
pair<const_iterator, const_iterator>
                equal_range(const key_type &k) const;
```

Возвращает пару итераторов, которые указывают на первый и последний элементы в мультиотображении, содержащие заданный ключ.

```
void erase(iterator i);
```

Удаляет элемент, адресуемый итератором i.

```
void erase(iterator start, iterator end);
```

Удаляет элементы в диапазоне, задаваемом параметрами start и end.

```
size_type erase(const key_type &k);
```

Удаляет из мультиотображения элементы, ключи которых имеют значение k.

```
iterator find(const key_type &k);
const_iterator find(const key_type &k) const;
```

Возвращает итератор для заданного ключа. Если ключ не обнаружен, возвращает итератор до конца мультиотображения.

```
allocator_type get_allocator() const;
```

Возвращает распределитель мультиотображения.

```
iterator insert(iterator i, const value_type &val);
```

Вставляет значение val после элемента, заданным итератором i, возвращает итератор для этого элемента.

```
template <class InIter> void insert(InIter start,
                                    InIter end);
```

Вставляет элементы заданного диапазона.

```
pair<iterator, bool> insert(const value_type &val);
```

Вставляет значение val в используемое мультиотображение. Возвращает итератор для данного мультиотображения. Элемент вставляет только в том случае, если его еще нет в мультиотображении. Если элемент был вставлен возвращает пару pair<iterator, true>, в противном случае pair<iterator, false>

```
key_compare key_comp() const;
```

Возвращает объект-функцию, которая сравнивает ключи.

```
iterator lower_bound(const key_type &k);
const_iterator lower_bound(const key_type &k) const;
```

Возвращает итератор для первого элемента в мультиотображении, ключ которого равен значению k или больше этого значения.

```
size_type max_size() const;
```

Возвращает максимальное число элементов, которое может содержать мультиотображение.

```
reference operator[](const key_type &i);
```

Возвращает ссылку на элемент, заданный параметром i. Если этого элемента не существует, вставляет его в мультиотображение.

```
reverse_iterator rbegin();
const_reverse_iterator rbegin() const;
```

Возвращает реверсивный итератор для конца мультиотображения.

```
reverse_iterator rbegin();
const_reverse_iterator rbegin() const;
```

Возвращает реверсивный итератор для начала мультиотображения.

```
size_type size() const;
```

Возвращает текущее количество элементов в мультиотображении.

```
void swap(multimap<Key, T, Comp, Allocator> &ob);
```

Выполняет обмен элементами данного мультиотображения и мультиотображения ob.

```
iterator upper_bound(const key_type &k);
const_iterator upper_bound(const key_type &k) const;
```

Возвращает итератор для первого элемента в мультиотображении, ключ которого больше заданного значения k.

```
value_compare value_comp() const;
```

Возвращает объект-функцию, которая сравнивает значения.

Пример использования класс multimap

```
//в данном примере в сравнительном виде показаны отличия
//в работе с отображением и мультиотображением

#include <iostream>
#include <map>     //map,multimap  -- отбражение,
                   //мультиотображение (отображение
                   //с повторениями)
#include <string>
using namespace std;

//шаблонная функция для вывода содержимого отбражения
//или мультиотображения на экран
template<class container> void show(container col)
{
    for(container::const_iterator i = col.begin();
                                   i != col.end(); ++i)
    {
        cout << i->first << '\t' << i->second << endl;
```

```
    }
    cout << endl << endl;
}

void main()
{
    cout << "map\n\n";
    //Создаем пустой контейнер (отображение)
    map<string,int> cont;

    //Создаем пустой контейнер (мультиотображение)
    multimap<string,int> multicont;

    //добавляем две пары в отображение
    cont.insert(pair<string,int>("Ivanov",10));
    cont.insert(pair<string,int>("Petrov",20));

    //добавится пара "Sidorov, 30"
    cont["Sidorov"] = 30;
    show(cont);

    //заменится значение в паре с ключем "Ivanov"
    cont["Ivanov"] = 50;
    show(cont);

    //Элемент не добавится, т.к. пара
    //с ключем "Ivanov" уже существует
    cont.insert( pair<string,int>("Ivanov",100) );
    show(cont);

    ////////////////////////////////////////////////
    cout << "-----------------------\nmultimap\n\n";

    multicont.insert( pair<string,int>("Ivanov",10) );
    multicont.insert( pair<string,int>("Petrov",20) );
    multicont.insert( pair<string,int>("Sidorov",20) );
```

```
     //Для мульти отображения не определен оператор "[]"
    //multicont["Sidorov"] = 30;    //Error
    show( multicont );

    //Добавляем пару ("Ivanov",100)
    multicont.insert( pair<string,int>("Ivanov",100) );
    show( multicont );

    //Ишем первое вхождение элемента с ключем "Petrov"
    multimap<string,int>::iterator iter =
                        multicont.find("Petrov");
    cout << iter->first << '\t' << iter->second
                        << endl << endl;

    cout << "Count of key \"Ivanov\" in multimap = "
         << multicont.count("Ivanov") << endl;

    //возвращает итератор, указывающий на первое включение
    //данного ключа или на конец отображения
    //в случае отсутствия
    iter = multicont.lower_bound("Ivanov");

    for(; iter != multicont.upper_bound("Ivanov")
             && iter != multicont.end(); iter++)
    {
        cout << iter->first << '\t' << iter->second
                                    << endl;
    }

    cout << endl << endl;
}

//Программа выводит следующий результат:
//map
//
//Ivanov  10
//Petrov  20
```

```
//Sidorov 30
//
//
//Ivanov  50
//Petrov  20
//Sidorov 30
//
//
//Ivanov  50
//Petrov  20
//Sidorov 30
//
//
//-------------------------------------------------
//multimap
//
//Ivanov  10
//Petrov  20
//Sidorov 20
//
//
//Ivanov  10
//Ivanov  100
//Petrov  20
//Sidorov 20
//
//
//Petrov  20
//
//Count of key "Ivanov" in multicont = 2
//Ivanov  10
//Ivanov  100
```

Домашнее задание

1. Заполнить вектор длинной 10 квадратами целых чисел и вывести его в выходной поток.
2. Заполнить двухмерный вектор таблицей умножения и выввести его в выходной поток.
3. Описать класс «студент» с полями: имя, фамилия, курс. Переопределить у этого класса оператор вывода в поток. Написать функцию заполнения вектора из класса «студент» произволными данными. Написать функцию печати содержимого вектора. Отсортировать вектор по именам студентов по возрастанию. Отсортировать стабильно вектор по фамилиям студентов. Поставить в первые три элемента вектора студентов самых младших курсов по возрастанию. После каждой операции выводить список студентов в выходной поток.