Obyektyönlü proqramlaşdırma

C++

ШАГ
компьютерная
академия

Dərs №16

C++ dili ilə obyektyönlü proqramlaşdırma

Mündəricat

vector sinfinin analizi və istifadəsi

vector kitabxanası

vector sinfi dinamik massivi və onda saxlanılan elementlər sayğacını dəstəkləyir. Onun şablon strukturu növbəti şəkildədir:

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

Burada **T -** *saxlanılan verilənlərin tipini*, **Allocator** *isə paylaşdırıcını ifadə edir*. ***vector*** sinfi növbəti konstruktorlar ehtiva edir:

```
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());
```

Konstruktorun birinci forması boş vektor yaradır. İkincisi val qiymətli num sayda element yaradan vektor yaradır. Üçüncüsü ob vektorunun malik olduğu elementləri ehtiva edir. Dördüncüsü start və end parametrləri ilə verilmiş diapazonda elementlər ehtiva edir.

vector sinfi üçün növbəti müqayisə operatorları təyin edilmmişdir:

■■==
■■<
■■<=
■■!=
■■>
■■>=

vector sinfi növbəti üzv-funksiyalar ehtiva edir:

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

start və end parametrləri ilə təyin edilən ardıcıllığı verktorda yerləşdirir.

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

val qiymətli num sayda elementləri vektorda yerləşdirir.

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

i parametri ilə verilmiş elementə istinadı qaytarır. Bu zaman, yüklənmiş [] operatorundan fərqli olaraq verilmiş funksiya massivin sərhədləri xaricinə çıxıldığında istisna hasil edir.

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

Vektorda sonuncu elementə istinadı qaytarır.

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

Vektorda birinci element üçün iterator qaytarır.


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

Əlavə yaddaşın ayrılması ehtiyacı duyulduqdan əvvəl, vektorda saxlanıla bilən elementlərin sayını ifadə edən vektorun cari ölçüsünü qaytarır.

```
void clear();
```

Bütün elemetləri vektordan silir.

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

İstifadə olunan vektor boşdursa, döğru qiymətini qaytarır, əks halda yalan qiymətini qaytarır.

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

Vektorun sonu üçün iterator qaytarır.

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

i iteratoru ilə ünvanlanan elementi silir, silinmiş elementdən sonra yerləşən element üçün iteratoru qaytarır.

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

start və end parametrləri ilə verilən diapazonda elementləri silir, sonuncu elementdən sonra yerləşən element üçün iteratoru qaytarır.

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

Vektorda birinci elementə istinadı qaytarır.

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

Vektorun paylaşdırıcısını qaytarır.

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

val qiymətini birbaşa i parametri ilə verilmiş elementdən əvvəl əlavə edir, bu element üçün iteratoru qaytarır.

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

val qiymətinin num sayda nüsxəsini i parametri ilə verilmiş elementdən əvvəl yerləşdirir.

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

star və end parametrləri ilə təyin edilimiş ardıcıllığı birbaşa i parametri ilə verilmiş elementdən əvvəl yerləşdirir.

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

Vektorun saxlaya bildiyi elementlərin maksimal sayını qaytarır.

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

i parametri ilə verilmiş elementə istinadı qaytarır.

```
void pop_back();
```

Vektorda sonuncu elementi qaytarır.


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

val parametri ilə verilmiş elementi vektorun sonuna əlavə edir.

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

Vektorun sonu üçün reversiv iteratoru qaytarır.

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

Vektorun əvvəli üçün reversiv iteratoru qaytarır.

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

Vektorun ölçüsünə verilmiş num qiymətindən az olmayan qiymət mənimsədir.

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

Vektorun ölçüsünə verilmiş num qiymətindən az olmayan qiymət mənimsədir, əgər bunun üçün vektorun uzunluğunu artırmaq lazım gələrsə, onda onun sonuna val parametri ilə verilmiş elementlər əlavə edilir.

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

Vektordakı elementlərin cari sayını qaytarır.

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

Verilmiş vektorun və ob vektorunun elementləri arasında mübadiləni icra edir.

```
void flip();
```

Vektorda bütün bitlərin elementlərini əks çevirir.

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

i və j parametrləri ilə verilmiş bitlərin yerlərini dəyişdirir.

vector sinfinin istifadəsinə aid nümunə

```
//Nümunə: Verilmiş nümunə vector konteyneri ilə
//iş üsullarını göstərir.
#include <iostream>
#include <vector>
using namespace std;

void main()
{
    //Vektor yaradır
    vector<int> vect;

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

    cout << "\n-----------------------------------";
    //size() metodundan istifadə etməklə vektordakı
    //elementərin cari sayını əldə edirik.
    //cout << "\nThe number of elements in the
           vector --> " << vect.size();

    cout << "\n-----------------------------------";
    vect.resize(4, 0); //ölçünü dəyişdiriirk,
    //elementlər sıfırlarla doldurulur
```



```
    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                                    ";
    //vektorun maksimal ölçüsü.
    //max_size() metodu baytların sayını qaytarır.
    cout << "\nThe maximum possible length of the
           vector --> "
           << vect.max_size()/4;

    cout << "\n                                    ";
    vect.push_back(1); //vektorun sonuna
                       //bir əlavə edirik
    cout << "\npush_back\nvector  -->\t";
    for (int i=0; i<vect.size(); i++)
    {
        cout << vect[i] << '\t';
    }
    cout << "\n                                    ";
    //revers iterator yaradırıq və onu vektorun
    //sonuna əlavə edirik

    vector<int>::reverse_iterator i_riterator =
          vect.rbegin();
    cout << "\nreverse_iterator\nvector   -->\t";
    //revers iteratordan istifadə etməklə vektorun
    //məzmununu ekrana veririk
    for (int i=0; i<vect.size(); i++)
    {
        cout << *(i_riterator+i) << '\t';
    }
```

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

    //adi iterator yaradırıq və onu vektorun
    //sonuna əlavə edirik
    vector<int>::iterator i_iterator = vect.end();
    //"-1" qiymətini sonuncu elementdən əvvəl

    //yerləşdiririk

    vect.insert(i_iterator-1, -1);

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

    //adi iteratordan istifadə edərək
    //vektorun məzmununu ekrana veririk
    for (i_iterator=vect.begin();
                i_iterator!=vect.end(); i_iterator++)
    {
        cout << *(i_iterator) << '\t';
    }
    cout << "\n----------------------------------- ";

    i_iterator = vect.end(); //vektorun sonu iteratoru
    vect.insert(i_iterator-1, 2, 4); //sonuncu
    //elementdən öncə 2 ədəd 4 yerləşdiririk

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

//Proqram növbəti nəticəni verir:

//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
//-----------------------------------
```

list sinfinin analizi və istifadəsi

list kitabxanası

list sinfi ikiistiqamətli əlaqəli siyahının işini dəstəkləyir. Onun şablonunun strukturu növbəti formadadır:

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

Burada **T** - ***siyahıda saxlanılan verilənlərin tipidir***. **list** sinfi növbəti konstruktorlar ehtiva edir:

```
    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());
```

Konstruktorun ilk forması boş siyahı yaradır. İkincisi val qiymətli num sayda element ehtiva edən siyahı yaradır. Üçüncüsü ob siyahısında olan elementləri ehtiva edən siyahı yaradır. Dördüncüsü verilmiş start və end parametrləri olan diapazonda elementləri ehtiva edən siyahı yaradır.

list sinfi üçün növbəti müqayisə operatorları təyin edilmişdir:

- ==
- <

- <=
- !=
- >
- >=

list sinfi növbəti üzv-funksiyalar ehtiva edir:

start və end parametrləri ilə təyin edilən ardıcıllığı siyahıya yerləşdirir.

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

val qiymətli num sayda elementləri siyahıda yerləşdirir.

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

Siyahının sonuncu elementinə istinadı qaytarır.

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

Siyahının birinci elementi üçün iteratoru qaytarır.

```
void clear();
```

Siyahıdan bütün elementləri silir.

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

İstifadə edilən siyahı boş olarsa, doğru qiymətini qaytarır və əks halda yalan qiymətini qaytarır.

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

Siyahının sonu üçün iterator qaytarır.

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

i iteratoru ilə ünvanlaşdırılan elementi silir, silinən elementdən sonra yerləşən element üçün iterator qaytarır.

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

start və end parametrləri ilə verilən diapazonda elementləri silir, sonuncu silinən elementdən əvvəl yerləşən element üçün iterator qaytarır.

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

Siyahının birinci elementinə istinadı qaytarır.

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

Siyahı paylaşdırıcısını qaytarır.

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

i parametri ilə verilmiş elementdən əvvəl val qiymətini yerləşdirir, bu element üçün iterator qaytarır.

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

val qiymətli num sayda nüsxəni birbaşa i parametri ilə verilmiş elementdən əvvəl yerləşdirir.


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

start və end parametrləri ilə təyin edilmiş ardıcıllığı siyahıda i parametri ilə verilmiş elementdən əvvəl birbaşa yerləşdirir.

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

Siyahıda ola biləcək elementlərin maksimal sayını qaytarır.

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

ob obyektində verilmiş çeşidlənmiş siyahını verilmiş çeşidlənmiş siyahı ilə birləşdirir.

Nəticə, həmçinin çeşidlənmiş olur. Siyahılar birləşdirildikdən sonra ob obyektində saxlanılanlar boş qalır. İkinci formada bir elementin digərindən kiçik olmasını təyin edən müqayisə funksiyası verilə bilər.

```
void pop_back();
```

Siyahının sonuncu elementini silir.

```
void pop_front();
```

Siyahının birinci elementini silir.

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

val parametri ilə verilmiş qiymətli elementi siyahının sonuna əlavə edir.

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

val parametri ilə verilmiş qiymətli elementi siyahının əvvəlinə əlavə edir.

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

Siyahının sonu üçün reversiv iterator qaytarır.

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

Siyahının əvvəli üçün reversiv iterator qaytarır.

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

val parametri ilə verilmiş qiymətli elementi siyahıdan silir.

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

pr unar predikatı true qiymətinə bərabər olan elementləri silir.

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

Siyahının ölçüsünü verilmiş num qiymətindən az olmayaraq təyin edir, əgər bunun üçün vektorun olcüsünü artırmaq lazım gələrsə, onda onun sonuna val parametri ilə verilmiş qiymətli elementlər əlavə edilir.

```
void reverse();
```

Siyahını tərs çevirir.


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

Siyahının cari elementlərinin sayını qaytarır.

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

Siyahını çeşidləyir. İkinci forma bir elementin digərindən kiçik olmasını müəyyən etmək üçün siyahını cmpfn funksiyası ilə çeşidləyir.

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

ob siyahısının məzmununu verilmiş siyahıda i iteratoru ilə verilmiş mövqeyə yerləşdirir. Bu əməliyyat icra edildikdən sonra ob siyahısı boşalır.

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

el iteratoru ilə ünvanlanan elementi ob siyahısından silir və onu i iteratoru ilə ünvanlanan mövqedə saxlayır.

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

star və end parametrləri ilə təyin edilən diapazonu ob siyahısından silir və onu verilmiş siyahıda i iteratoru ilə ünvanlanan mövqedən başlayaraq yerləşdirir.

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

Verilmiş siyahının və ob siyahısının elementlərini mübadilə edir.

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

Dublikat-elementləri siyahıdan silir. İkinci forma nadirliliyin (unikallığın) təyin edilməsi üçün pr predikatını istifadə edir.

list sinfinin istifadəsinə aid nümunə

```
//Nümunə: siyahılarla iş prinsipləri
//Siyahının yaradılması, doldurulması,
//çeşidlənməsi, ekrana verilməsi.
//İteratorlarla iş prinsipləri

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

typedef list<int> ourList;

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

    cout << "list1: ";
    for (iter = l1.begin(); iter != l1.end(); iter++)
    {
        //iterator göstərən elementi ekrana veririk
         cout << *iter << " ";
    }

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



```
void main()
{
    //İki boş siyahının yaradılması
   ourList list1, list2;

    //Hər iki siyahının elementlərlə doldurulması
    for (int i=0; i<6; ++i)
   {
        list1.push_back(i);
        list2.push_front(i);
    }
    //Siyahıların ekrana verilməsi
    ShowLists(list1, list2);

    //İkinci siyahıda birinci elementin sona
    //yerləşdirilməsi
    list2.splice(list2.end(),   //Qəbuledicidə mövqe
        list2,                  //Mənbə
        list2.begin());         //Mənbədə mövqe

    //Birinci siyahını tərs çeviririk
    list1.reverse();
    ShowLists(list1, list2);

    //Hər iki siyahının çeşidlənməsi
    list1.sort();
    list2.sort();
    ShowLists(list1, list2);

    //İki çeşidlənmiş siyahını
    //birincidə birləşdiririk
    list1.merge(list2);
    ShowLists(list1, list2);

    //Dublikatları birinci siyahıdan silirik
    list1.unique();
    ShowLists(list1, list2);
}
```

```
//Proqram növbəti nəticəni verir:

//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:
```


map sinfinin analizi və istifadəsi

map kitabxanası

map sinfi unikal açarlara müəyyən qiymətlər uyğun gələn assosiativ konteyneri dəstəkləyir. Onun şablonunun strukturu növbəti formadadır:

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

Burada **key** - ***açarların verilənlər tipidir***, **T** - ***saxlanılan (əks etdirilən) qiymətin tipidir***, **Comp** ***isə - iki açarı müqayisə edən funksiyadır***. **map** sinfi növbəti kostruktorlar ehtiva edir:

```
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());
```

Konstruktorun birinci forması boş surət yaradır. İkincisi ob surətinin ehtiva etdiyi elementlər ehtiva edən surət yaradır. Üçüncüsü star və end parametrləri ilə verilmiş diapazonda elementləri ehtiva edən surət yaradır. cmpfn (əgər o verilibsə) parametrləri ilə verilmiş funksiya çeşidlənmiş surət təyin edir.

map sinfi üçün növbəti müqayisə operatorları təyin edilib:

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

map sinfi aşağıda verilmiş növbəti üzv-funksiyalar ethtiva edir. Verilmiş təsvirdə key_type elementi açarın tipini əks etdirir, value_type elementi isə - pair<Key, T> elementlər cütlüyünü əks etdirir.

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

Surətdəki birinci element üçün iterator qaytarır.

```
void clear();
```

Surətdən bütün elementləri silir.

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

k açarının surətdə rast kəlmə sayını qaytarır (1 və ya 0).

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

Əgər verilmiş surət boşdursa, true qiymətini qaytarır, əks halda false qiymətini qaytarır.


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

Surətin sonunu göstərən iteratoru qaytarır.

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

Verilmiş açarı ehtiva edən surətin birinci və sonuncu elementlərini göstərən iteratorlar cütünü qaytarır.

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

i iteratoru ilə ünvanlanan elementi silir.

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

star və end parametrləri ilə verilən diapazonda elementləri silir.

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

Açarları k qiyməti olan elementləri surətdən silir.

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

Verilmiş açar üçün iterator qaytarır. Əgər açar əks olunmayıbsa, surətin sonuna qədər iteratoru qaytarır.

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

Surət paylaşdırıcısını qaytarır.

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

val qiymətini i iteratoru ilə verilmiş elementdən sonra yerləşdirir və bu element üçün iterator qaytarır.

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

Verilmiş diapazonda elementləri yerləşdirir.

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

val qiymətini istifadə edilən surətdə yerləşdirir. Verilmiş surət üçün iterator qaytarır. Element yalnız o surətdə olmadıqda yerləşdirilir. Əgər element yerləşdirilmiş olarsa, pair<iterator, true> cütlüyünü qaytarır, əks halda pair<iterator, false> cütlüyünü qaytarır.

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

Açarları müqayisə edən obyekt-funksiya qaytarır.

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

Açarı k qiymətinə və ya bu qiymətdən böyük qiymətə bərabər olan surətin birinci elementi üçün iterator qaytarır.


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

Surətin saxlaya biləcəyi elementlərin maksimal sayını qaytarır.

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

i parametri ilə verilən elementə istinadı qaytarır. Əgər bu element mövcud deyilsə, onu surətə yerləşdirir.

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

Surət üçün reversiv iterator qaytarır.

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

Surətin əvvəli üçün reversiv iteratoru qaytarır.

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

Surətdə olan elementlərin cari sayını qaytarır.

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

Verilmiş surətdəki və ob surətindəki elementlərin mübadiləsini yerinə yetirir.

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

Surətdəki açarı verilmiş k qiymətindən böyük birinci element üçün iterator qaytarır.

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

Qiymətləri müqayisə edən obyekt-funksiyanı qaytarır.

Bu sinif qiyməti açara görə sürətli tapmaq üçün nəzərdə tutulmuşdur. Açar kimi hər şey istifadə edilə bilər, lakin bu zaman xatırlamaq lazımdır ki, açarın əsəs xüsusiyyəti ona müqayisə əməliyyatının tətbiq edilə bilməsidir. Qiymətin açara görə sürətli tapılması cütlüyün çeşidlənmiş şəkildə olmasına görə mümkündür.

Bu sinfin çatışmayan cəhəti - yeni cütlüyün daxil edilməsi sürəti sinifdə saxlanılan elementlərin sayı ilə tərs mütənasibdir. Daha bir vacib məsələ - açar unikal olmalıdır.

map sinfinin istifadəsinə aid nümunə

```
//Nümunə: verilmiş nümunə surətlər üzərində

//işləmə metodlarını nümayiş etdirir

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

void main()
{
```



```
    //surət yaradırıq
    map <int, int> our_map;

    //vektor yaradırıq
    vector <int> our_vector;

    //vektorun maksimal ölçüsü
    cout << "\n\nmax size of vector --> "
         << our_vector.max_size() / sizeof(int);

    //surətin maksimal ölçüsü

    //(iki dəfə azdır, belə ki, hər bir element üçün
    //iki qiymət - cütlük saxlamaq lazımdır)

    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;
    //İki qiymət əsasında cütlük yaradırıq.
    pair<int, int> element(key, val);

    //Surətdə cütlüyü yerləşdiririk
    our_map.insert(element);

    //surətdəki elementlərin sayı
    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)
    {
        //elementi surətə əlavə etmək mümkün
        //olmadığında işə düşür
        //məsələn, əgər surətdə verilmiş
        //açarlı element artıq mövcuddursa.
        cout << "\nError !!!\n";
    }
    //Surətdəki elementlərin sayı
    cout << "\nCurrent element of map --> "
         << our_map.size() << endl;

    //bütün elementlərin ekrana verilməsi
    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";
}
```


multimap sinifinin analizi və istifadəsi

multimap kitabxanası

map-in dəyişdirilmiş variantıdır. Açarın unikallığı tələb olunmur - yəni, əgər açara görə axtarış aparılarsa, onda heç bir qiymət qaytarılmır, lakin bu açarla saxlanılan qiymətlər çoxluğu qaytarılır.

multimap sinfi ümumi halda unikal olmayan açara müəyyən qiymətin uyğun gəldiyi assosiativ konteyneri dəstəkləyir. Onun şablonunun strukturu növbəti formadadır:

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

Burada **key** - ***verilmiş açarların tipidir***, **T** - ***saxlanılan (əks olunan) qiymətlərin tipidir***, **Comp** ***isə - iki açarın müqayisə edildiyi funksiyadır***. **multimap** sinfi növbəti konstruksiyaya malikdir:

```
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());
```

Konstruktorun birinci forması boş multisurət yaradır. İkincisi ob multisurətinin ehtiva etdiyi elementlər ehtiva edən multisurət yaradır. Üçüncüsü start və end parametrləri ilə verilmiş diapazonda elementləri ehtiva edən multisurət yaradır. cmpfn (əgər o verilibsə) parametrləri ilə verilmiş funksiya çeşidlənmiş multisurət təyin edir.

multimap sinfi üçün növbəti müqayisə operatorları təyin edilib:

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

multimap sinfi aşağıda verilmiş növbəti üzv-funksiyalar ethtiva edir. Verilmiş təsvirdə key_type elementi açarın tipini əks etdirir, value_type elementi isə - pair<Key, T> elementlər cütlüyünü əks etdirir.

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

Multisurətdəki birinci element üçün iterator qaytarır.

```
void clear();
```

Multisurətdən bütün elementləri silir.


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

k açarının multisurətdə rast kəlmə sayını qaytarır (1 və ya 0).

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

Verilmiş multisurət boş olarsa, true qiymətini qaytarır, əgs halda false qiymətini qaytarır.

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

Multisurətin sonunu göstərən iteratoru qaytarır.

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

Verilmiş açarı ehtiva edən multisurətin birinci və sonuncu elementlərini göstərən iteraorlar cütünü qaytarır.

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

i iteratoru ilə ünvanlanan elementi silir.

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

star və end parametrləri ilə verilən diapazonda elementləri silir.

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

Açarları k qiyməti olan elementləri multisurətdən silir.

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

Verilmiş açar üçün iterator qaytarır. Əgər açar əks olunmayıbsa, multisurətin sonuna qədər iteratoru qaytarır.

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

Multisurət paylaşdırıcısını qaytarır.

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

val qiymətini i iteratoru ilə verilmiş elementdən sonra yerləşdirir və bu element üçün iterator qaytarır.

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

Verilmiş diapazonda elementləri yerləşdirir.

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

val qiymətini istifadə edilən multisurətdə yerləşdirir. Verilmiş multisurət üçün iterator qaytarır. Element, yalnız o multisurətdə olmadıqda yerləşdirilir. Əgər element yerləşdirilmiş olarsa, pair<iterator, true> cütlüyünü qaytarır, əks halda pair<iterator, false> cütlüyünü qaytarır.

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

Açarları müqayisə edən obyekt-funksiya qaytarır.


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

Açarı k qiymətinə və ya bu qiymətdən böyük qiymətə bərabər olan multisurətin birinci elementi üçün iterator qaytarır.

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

Multisurətin saxlaya biləcəyi elementlərin maksimal sayını qaytarır.

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

i parametri ilə verilən elementə istinadı qaytarır. Əgər bu element mövcud deyilsə, onu multisurətə yerləşdirir.

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

Multisurət üçün reversiv iterator qaytarır.

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

Multisurətin əvvəli üçün reversiv iteratoru qaytarır.

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

Multisurətdə olan elementlərin cari sayını qaytarır.

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

Verilmiş multisurətdəki və ob multisurətindəki elementlərin mübadiləsini yerinə yetirir.

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

Multisurətdəki açarı verilmiş k qiymətindən böyük birinci element üçün iterator qaytarır.

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

Qiymətləri müqayisə edən obyekt-funksiyanı qaytarır.

multimap sinfinin istifadəsinə aid nümunə

```
//Bu nümunədə surət və multisurətlə işin fərqi
//müqayisəli şəkildə verilir
#include <iostream>
                  //map, multimap  - surət,
#include <map>    //multisurət (təkrar olan surət)

#include <string>
using namespace std;

//Surətin və ya multisurətin məzmununu ekrana verən
//şablon funksiya
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";
    //Boş konteyner (surət) yaradırıq
    map<string,int> cont;

    //Boş konteyner (multisurət) yaradırıq
    multimap<string,int> multicont;

    //İki cütlüyü surətə əlavə edirik
    cont.insert(pair<string,int>("Ivanov",10));
    cont.insert(pair<string,int>("Petrov",20));

    // "Sidorov, 30" cütlüyü əlavə edilir
    cont["Sidorov"] = 30;
    show(cont);

    //"Ivanov" açarlı cütlükdə qiymət dəyişdirilir.
    cont["Ivanov"] = 50;
    show(cont);

    //Element əlavə edilmir, belə ki,
    //"Ivanov" açarı artıq mövcuddur
    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) );
```

```
    //Multisurət üçün "[]" operatoru təyin edilməmişdir
    //multicont["Sidorov"] = 30;    //Error
    show( multicont );

    // ("Ivanov",100) cütlüyünü əlavə edirik
    multicont.insert( pair<string,int>("Ivanov",100) );
    show( multicont );

    //"Petrov" açarlı elementin ilk rast
    //gəlinməsini axtarırıq
    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;

    //verilmiş açarı və ya surətin sonunu (olmadığı
    halda)göstərən iteratoru qaytarır
    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;
}

//Proqram növbəti nəticəni verir:
//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
```

Ev tapşırığı

1. Uzunluğu 10 olan vektoru tam ədədlərin kvadratı ilə doldurun və onu çıxış axınına verin.
2. İkiölçülü vektoru vurma cədvəli ilə doldurun və onu çıxış axınına verin.
3. “Tələbə” sinfini növbəti sahələrlə verməli: ad, soyad, kurs. Bu sinifdə axına çıxış operatornu təyin etməli. “Tələbə” sinfinin vektorunun ixtiyari verilənlərlə doldurulması funksiyasını yazmalı. Vektorun məzmununu ekrana verən funksiya təyin etməli. Vektoru tələbələrin adına görə çeşidləməli. Vektoru tələbələrin soyadına görə stabil çeşidləməli. Vektorun ik üç elementinə ən aşağı kurs tələbələrini artan sıra ilə yerləşdirməli. Hər əməliyyatdan sonra tələbələrin siyahısını çıxış axınına verməli.