نسخه
مفت

STL for TMU University Students

safeallah ramezanzadeh

این متن آموزشی برای همه علاقه مندان به برنامه نویسی به زبان C++ نوشته شده است، به ویژه بروبچ تربیت معلم.

ویرایش دوم

By Hich Kas
safecomp85@gmail.com
نسخه مفت

پشته Stack

بردار Vector

لیست List

صف دوطرفه Deque

صف Queue

صف اولویت Priority Queue

گروه Set

گروه چندگانه Multiset

گروه-بیت Bitset

نگاشت Map

نگاشت چند گانه Multimap

ساختمان داده هايي كه براساس الگوريتم هاي درهم سازي كار مي كنند

dev c++:

hash_set

hash_map

به سوی c++0x:

unordered_set

unordered_multiset

unordered_map

unordered_multimap

<complex>

نکته:تمام مثال های این متن آموزشی در Dev-C++ 4.9.9.2 نوشته شده اند.

پشته (Stack)

ساختمان داده ایست با ویژگی Last-In-First-Out.

برای استفاده از این ساختمان داده باید ابتدا کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم.

```
#include<stack>
```

متدهای مهم این ساختمان داده عبارتند از(نحوه استفاده از این متد ها در مثال نشان داده شده است):

push: این متد عنصری را به بالای پشته اضافه می کند.

pop: عنصر بالای پشته را حذف می کند.

top: عنصر بالای پشته را بر می گرداند ولی آن را از پشته حذف نمی کند.

empty: برای تشخیص خالی بودن یا نبودن پشته از آن استفاده می کنیم.

size: اندازه پشته را بر می گرداند.

مثال 1:پشته ای از اعداد

```
#include <iostream>
#include<stack>
using namespace std;
stack<int> st;
int main(){
    int hold;
    int x[]={2,4,6,8,10,12};
    cout<<"length of x :"<<sizeof(x)/sizeof(int)<<"\n";
    for(int i=0;i<sizeof(x)/sizeof(int);i++)
    {
            st.push(x[i]);
    }
    cout<<"Number Of Stack Elements :"<<st.size()<<"\n";
    cout<<"Top Of Stack :"<<st.top()<<"\n";
    while(!st.empty()){
           cout<<st.top()<<"\n";
           st.pop();
    }
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
length of x :6
Number Of Stack Elements :6
Top Of Stack :12
12
10
8
6
4
2
```

مثال 2:پشته ای از رشته ها

```
#include <iostream>
#include<stack>
#include <string>
using namespace std;
stack<string> st;
int main(){
    int hold;
    int num=0;
    cin>>num;
    string map[num];
    for(int i=0;i<num;i++)
    {
            cin>>map[i];
            st.push(map[i]);
    }
    for(int i=0;i<num;i++)
            cout<<map[i]<<"\n";
    cout<<"*********\n";
    cout<<"Number Of Stack Elements :"<<st.size()<<"\n";
    cout<<"Top Of Stack :"<<st.top()<<"\n";
    while(!st.empty()){
           cout<<st.top()<<"\n";
           st.pop();
    }
    cin>>hold;
    return 0;
}
```

نمونه ای از اجرای برنامه:

```
3
ali
reza
hasan
ali
reza
hasan
*********
Number Of Stack Elements :3
Top Of Stack :hasan
hasan
reza
ali
```

مثال 3:پشته ای از ساختار داده ای خودمان

گاهی اوقات نیاز داریم که عناصر پشته ساختار هایی باشند که خودمان ایجاد کرده ایم.

برای مثال ساختاری را در نظر بگیرید که زوج نام و نمره را برای هر فرد نگه می دارد.این ساختار را Pair می نامیم.از یک string برای نگه داری نام افراد و از یک int برای نگه داری نمره استفاده می کنیم.

*روش های جالبی برای تبدیل اعداد به رشته و برعکس در متد toString نشان داده شده است.

```
#include <iostream>
#include<stack>
#include <string>
#include <stdlib.h>
using namespace std;
typedef struct Pair{
        string name;
        int num;
};
string toString(Pair p){
       string r="";
       char num2str[33];
       itoa(p.num,num2str,10); //convert int to char array
       //parameters 1:number 2:char array 3:base of conversion
       //int x=atoi(num2str)+1;//convert char array to int
       //other usefule functions are atof //convert char array to float
       //atol //convert char array to long int
       //another way for converting numbers to string is using sprintf:
       ///double f=3.2534534;
       //sprintf(num2str,"%.4f",f);
       //cout<<num2str;
       r=p.name+" "+num2str;
       return r;
}
stack<Pair> st;
int main(){
    int hold;
    int num=0;
    cin>>num;
    for(int i=0;i<num;i++)
    {
            Pair temp;
            cin>>temp.name;
            cin>>temp.num;
            st.push(temp);
    }
    cout<<"*********\n";
    cout<<"Number Of Stack Elements :"<<st.size()<<"\n";
    cout<<"Top Of Stack :"<<toString(st.top())<<"\n";
    while(!st.empty()){
           cout<<toString(st.top())<<"\n";
           st.pop();
    }
    cin>>hold;
    return 0;
}
```

نمونه ای از اجرای برنامه

```
4
ali 10
reza 11
hasan 12
asghar 13
*********
Number Of Stack Elements :4
Top Of Stack :asghar 13
asghar 13
hasan 12
reza 11
ali 10
```

بردار(Vector)

بردار ها در حقیقت آرایه هایی با طول متغیر هستند. برای استفاده از این ساختمان داده باید ابتدا کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم.

```
#include <vector>
```

*بردار ها random Access را نیز پشتیبانی می کنند،یعنی با استفاده از اندیس یک عنصر می توان به آن دسترسی داشت.

متد های مهم این ساختمان داده عبارتند از:

back : عنصر آخر بردار را بر می گرداند.

front: عنصر ابتدای بردار را بر می گرداند.

begin: یک iterator که به ابتدای بردار اشاره می کند را بر می گرداند.

end: یک iterator که به انتهای بردار اشاره می کند را بر می گرداند.

clear: تمام عناصر بردار را از آن حذف می کند.(البته این عمل به صورت واقعی صورت نمی گیرد)

push_back : یک عنصر به انتهای بردار اضافه می کند.

pop_back: عنصر آخر بردار را حذف می کند.

size: اندازه بردار را بر می گرداند.

empty: برای تشخیص خالی بودن یا نبودن بردار از آن استفاده می کنیم.

insert : برای درج عنصر در مکان مشخصی از بردار از این متد استفاده می کنیم،البته این مکان با یک iterator مشخص می شود و نه اندیس عددی!.

erase: برای حذف یک یا چند عنصر از این متد استفاده می شود.

*iterator ها ابزاری برای پیمایش هستند و عملگر های ++ و -- و - و + و... برای آنها overload شده اند ،برای مثال هنگامی که یک iterator را یک واحد افزایش می دهیم ،به عنصر بعدی در مجموعه(مجوعه می تواند هر چیزی باشد،مثلاً set یا vector یا ...) اشاره می کنیم (محاسبه عنصر بعدی بر اساس الگوریتم خاصی صورت می گیرد) و این افزایش را نباید با افزایش صرفاً مقدار آدرس یک شی اشتباه گرفت.(هنگامی که با set آشنا می شویم این موضوع بیشتر مورد اهمیت قرار می گیرد)

مثال 1: آشنایی مقدماتی با vector و نحوه استفاده از iterator

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
vector<string> v;
int main(){
    int hold;
    string arr[]={"ali","reza","ali","hasan","taghi","naghi"};
    for(int i=0;i<6;i++)
            v.push_back(arr[i]);
    cout<<"Size Of Vector :"<<v.size()<<"\n";
    vector<string>::iterator first=v.begin();
    vector<string>::iterator last=v.end();
    cout<<"First Element Of Vector :"<<v.front()<<"\n";
    cout<<"Last Element Of Vector :"<<v.back()<<"\n";
    cout<<"---------------\n";
    while(first!=last){
                    string obj=(string)*first;//just cast!
                    cout<<obj<<"\n";
                    first++;//point to next element
    }
    cout<<"---------------\n";
    cin>>hold;
    return 0;
}
```

همانطور که مشاهده می کنید از push_back برای اضافه کردن عنصر جدید به انتهای بردار استفاده شده است،در ضمن این مثال نشان می دهد که بردار ساختمان داده ای است که عناصر تکراری را نیز نگه داری می کند.نحوه استفاده از iterator نیز در این مثال نشان داده شده است.

خروجی برنامه:

```
Size Of Vector :6
First Element Of Vector :ali
Last Element Of Vector :naghi
---------------
ali
reza
ali
hasan
taghi
naghi
---------------
```

مثال 2: (بی مقدمه!)

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
vector<string> v;
int main(){
    int hold;
    string arr[]={"ali","reza","ali","hasan","taghi","naghi"};
    for(int i=0;i<6;i++)
            v.push_back(arr[i]);
    cout<<"Size Of Vector :"<<v.size()<<"\n";

    cout<<"---------------\n";
    while(!v.empty()){
                    cout<<v.back()<<"\n";
                    v.pop_back();
    }
    cout<<"---------------\n";
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
Size Of Vector :6
---------------
naghi
taghi
hasan
ali
reza
ali
---------------
```

مثال 3:vector از random access پشتیبانی می کند (یعنی با اندیس عنصر می توان به آن دسترسی داشت)

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
```

```
vector<string> v;
int main(){
    int hold;
    string arr[]={"ali","reza","ali","hasan","taghi","naghi"};
    for(int i=0;i<6;i++)
            v.push_back(arr[i]);
    cout<<"Size Of Vector :"<<v.size()<<"\n";
    v[0]="safeallh";//random access to first element
    cout<<"---------------\n";
    for(int i=0;i<v.size();i++)
    {
            cout<<v[i]<<"\n";
            v[i]="*";//random access to i'th element
    }

    cout<<"---------------\n";
    for(int i=0;i<v.size();i++)
            cout<<v[i]<<"  ";
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
Size Of Vector :6
---------------
safeallh
reza
ali
hasan
taghi
naghi
---------------
*  *  *  *  *  *
```

مثال 4:هنگام استفاده از ویژگی random access باید بسیار مواظب باشید ،زیرا ممکن است به اندیس های تعریف نشده ای اشاره کنید.

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
vector<string> v;
int main(){
    int hold;
    v[0]="safeallh";//random access to undefined index
    cin>>hold;
    return 0;
}
```

مثال 5:درج یک عنصر ،درج یک عنصر به تعداد n

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
typedef vector<string> myvector;
myvector v;
```

```
void print(myvector m){
     cout<<"------------\n";
     for(int i=0;i<m.size();i++){
             cout<<i<<" "<<m[i]<<"\n";
     }
     cout<<"------------\n";
}
int main(){
    string arr[]={"ali","reza","ali","hasan"};
    int hold;
    for(int i=0;i<4;i++){
            v.push_back(arr[i]);
    }
    print(v);

    v.insert(v.begin(),"ahad");//insert to the first
    print(v);

    int i=2;
    myvector::iterator i_iterator=v.begin()+i;
    v.insert(i_iterator,"alireza");//insert to i'th position
    print(v);

    int n=3;
    v.insert(i_iterator,n,"safeallh");//insert n copy of new element to
                                      //i'th position
    print(v);

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
------------
0 ali
1 reza
2 ali
3 hasan
------------
------------
0 ahad
1 ali
2 reza
3 ali
4 hasan
------------
------------
0 ahad
1 ali
2 alireza
3 reza
4 ali
5 hasan
------------
------------
0 ahad
1 ali
2 safeallh
3 safeallh
4 safeallh
```

```
5 alireza
6 reza
7 ali
8 hasan
------------
```

مثال 6:حذف یک عنصر ،حذف یک بازه

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
typedef vector<string> myvector;
myvector v;
void print(myvector m){
     cout<<"------------\n";
     for(int i=0;i<m.size();i++){
             cout<<i<<" "<<m[i]<<"\n";
     }
     cout<<"------------\n";
}
int main(){

    string arr[]={"ali","reza","alireza","hasan",
                  "ahad","safeallah","mansoor","akbar"};
    int hold;
    for(int i=0;i<sizeof(arr)/sizeof(string);i++){
            v.push_back(arr[i]);
    }
    print(v);

    int i=2;
    myvector::iterator i_iterator=v.begin()+i;
    v.erase(i_iterator);//delete element in i'th position
    print(v);
    i_iterator=v.begin()+3;
    myvector::iterator j_iterator=v.begin()+6;
    v.erase(i_iterator,j_iterator);//delete an interval from [i to j-1]
    print(v);

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
------------
0 ali
1 reza
2 alireza
3 hasan
4 ahad
5 safeallah
6 mansoor
7 akbar
------------
------------
0 ali
1 reza
2 hasan
```

```
3 ahad
4 safeallah
5 mansoor
6 akbar
------------
------------
0 ali
1 reza
2 hasan
3 akbar
------------
```

لیست(List)

لیست نیز مانند بردار ساختاری است که طول آن به صورت پویا تغییر می کند،با این تفاوت که لیست ساختاری پیوندی دارد و این موضوع باعث می شود که درج یک عنصر جدید در آن در زمانی ثابت انجام شود (البته درج عنصر جدید به انتهای بردار نیز در زمانی ثابت انجام می شود).این ویژگی لیست به ویژه زمانی که درج عنصر باید در ابتدای مجموعه صورت گیرد خودنمایی می کند. برای استفاده از این ساختمان داده باید ابتدا کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم.

```
#include <list>
```

*لیست فقط به صورت ترتیبی پیمایش می شود و از random Access پشتیبانی نمی کند.

متدهای مهم این ساختمان داده عبارتند از:

back: عنصر انتهای مجموعه را بر می گرداند.

begin: یک iterator که ابتدای مجموعه اشاره می کند را بر می گرداند.

clear: لیست را پاک می کند.

empty: از این متد برای تشخیص خالی بودن یا نبودن مجموعه استفاده می کنیم.

end: یک iterator که به انتهای لیست اشاره می کند را بر می گرداند.

erase: برای حذف یک یا چند عنصر از لیست از این متد استفاده می کنیم.

front: عنصر ابتدای مجموعه را بر می گرداند.

insert: برای درج عنصر جدید به لیست از این متد استفاده می کنیم.

merge: دو لیست مرتب شده را با هم ترکیب می کند.نتیجه یک لیست مرتب شده است.

pop_back: عنصر انتهای لیست را حذف می کند.

pop_front: عنصر ابتدای لیست را حذف می کند.

push_back: عنصر جدید را به انتهای لیست اضافه می کند.

push_front: عنصر جدید را به ابتدای لیست اضافه می کند.

remove: عناصر با مقادیر خاص را از لیست حذف می کند.

reverse: لیست را معکوس می کند.

size: اندازه لیست را بر می گرداند.

sort: لیست را مرتب می کند.

splice: عناصری را از یک لیست برداشته و به لیستی دیگر اضافه می کند.

unique: تمام تکرار های متوالی را حذف می کند.

مثال 1:آشنایی مقدماتی با list

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

list<int> l1;
int main(){

    int hold;
    int x[]={1,3,4,6,7,9,13,16};
    for(int i=0;i<8;i++){
            l1.push_back(x[i]);
    }
    list<int>::iterator first=l1.begin();
    list<int>::iterator last=l1.end();
    cout<<"Size Of List :"<<l1.size()<<"\n";
    while(first!=last){
                        cout<<(int)*first<<"\n";//just cast!
                        first++;
    }

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
Size Of List :8
1
3
4
6
7
9
13
16
```

مثال 2:استفاده از push_front

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

list<int> l1;
int main(){

    int hold;
    int x[]={1,3,4,6,7,9,13,16};
    for(int i=0;i<8;i++){
            l1.push_front(x[i]);
    }
    list<int>::iterator first=l1.begin();
    list<int>::iterator last=l1.end();
    cout<<"Size Of List :"<<l1.size()<<"\n";
    while(first!=last){
                        cout<<(int)*first<<"\n";//just cast!
                        first++;
    }

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
Size Of List :8
16
13
9
7
6
4
3
1
```

مثال 3: چرا لیست؟؟؟؟؟

```
#include <iostream>
#include <vector>
#include <list>
#include <ctime>
using namespace std;

list<int> l1,l2;
vector<int> v1,v2;
int main(){

    int hold;
    int value=8;
    int c=clock();
    //adding to the end----------------------
    for(int i=0;i<500000;i++){
            v1.push_back(value);
    }
    cout<<"Time Of Adding Element to the end of vector:"
                                    <<(clock()-c)<<"\n";
    c=clock();
    for(int i=0;i<500000;i++){
            l1.push_back(value);
    }
    cout<<"Time Of Adding Element to the end of list:  "
                                    <<(clock()-c)<<"\n";

    //adding to the front--------------------
    c=clock();
    for(int i=0;i<100000;i++){
            v2.insert(v2.begin(),value);
    }
    cout<<"Time Of Adding Element to the front of vector:"
                                    <<(clock()-c)<<"\n";
    c=clock();
    for(int i=0;i<100000;i++){
            l2.push_front(value);
    }
    cout<<"Time Of Adding Element to the front of list:  "
                                    <<(clock()-c)<<"\n";

    cin>>hold;
    return 0;
}
```

نمونه ای از خروجی برنامه:

```
Time Of Adding Element to the end of vector:15
Time Of Adding Element to the end of list:  188
Time Of Adding Element to the front of vector:6734
Time Of Adding Element to the front of list:  47
```

مثال 4:pop_back و pop_front

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

list<string> l1,l2;
int main(){

    int hold;
    string x[]={"ali","reza","hasan","taghi","naghi","jamshid!"};
    for(int i=0;i<6;i++){
         l1.push_back(x[i]);
         l2.push_back(x[i]);
    }

    cout<<"l1:\n";
    while(!l1.empty()){
          cout<<l1.back()<<"\n";
          l1.pop_back();
    }
    cout<<"--------------------\n";
    cout<<"l2:\n";
    while(!l2.empty()){
             cout<<l2.front()<<"\n";
             l2.pop_front();
    }

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
l1:
jamshid!
naghi
taghi
hasan
reza
ali
--------------------
l2:
ali
reza
hasan
taghi
naghi
jamshid!
```

مثال 5:مرتب سازی،ترکیب،معکوس کردن لیست و...

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

list<int> l1,l2;
void print(list<int> l){
     list<int>::iterator first=l.begin();
     list<int>::iterator last=l.end();
     cout<<"\n-----------------\n";
     if(l.empty()){
                    cout<<"Empty";
     }else{
           while(first!=last){
               cout<<(int)*first<<"   ";
               first++;
           }
     }
     cout<<"\n-----------------\n";
}
int main(){

    int hold;
    int x1[]={3,2,8,1,15,14,17};
    int x2[]={4,6,5,9,18,7};
    for(int i=0;i<sizeof(x1)/sizeof(int);i++){
            l1.push_back(x1[i]);
    }
    print(l1);
    for(int i=0;i<sizeof(x2)/sizeof(int);i++){
            l2.push_back(x2[i]);
    }
    print(l2);
    l1.sort();
    l2.sort();
    print(l1);
    print(l2);
    l1.merge(l2);
    print(l1);
    print(l2);
    l1.reverse();
    print(l1);
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
--------------------------------
3  2  8  1  15  14  17
--------------------------------

--------------------------------
4  6  5  9  18  7
--------------------------------

--------------------------------
1  2  3  8  14  15  17
--------------------------------

--------------------------------
4  5  6  7  9  18
--------------------------------

--------------------------------
1  2  3  4  5  6  7  8  9  14  15  17  18
--------------------------------

--------------------------------
Empty
--------------------------------

--------------------------------
18  17  15  14  9  8  7  6  5  4  3  2  1
--------------------------------
```

همانطور که مشاهده می کنید،بعد از فراخوانی merge لیست دوم خالی می شود.

مثال 6:erase،remove،insert،unique و ...

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

void print(list<int> p){
	cout<<"\n--------------------------------\n";
	if(p.empty()){
		cout<<"Empty";
	}else{
		list<int>::iterator start=p.begin();
		while(start!=p.end()){
			int p=(int)*start;
			cout<<p<<" ";
			start++;
		}
	}
	cout<<"\n--------------------------------\n";
}

list<int> l1,l2;
```

```
int main(){
    int hold;
    //0:
    int x[]={1,2,8,7,6,3,4,4,4,5,8,8,11,9,10};
    int value=3,n=5;
    l1=list<int>(n,value);
    l2=list<int>(x,x+sizeof(x)/sizeof(int));
    cout<<"\n0**********************\n";
    print(l1);
    print(l2);
    //1:erase i'th element
    int i=3,j;
    list<int>::iterator i_it=l2.begin();
    list<int>::iterator j_it=l2.begin();
    //caution!   i_it=i_it+3 is not applicable for list's iterator ,
    ///use "advance" instead
    advance(i_it,i);
    l2.erase(i_it);
    cout<<"\n1**********************\n";
    print(l2);
    //2:erase from [i to j-1]
    i=4;
    j=8;
    i_it=l2.begin();
    j_it=l2.begin();
    advance(i_it,i);
    advance(j_it,j);
    l2.erase(i_it,j_it);
    cout<<"\n2**********************\n";
    print(l2);

    //3:insert 18 ot 3'th position
    i_it=l2.begin();
    advance(i_it,3);
    l2.insert(i_it,18);
    cout<<"\n3**********************\n";
    print(l2);

    //4:insert 4 copy of 17 to the position of i_it
    l2.insert(i_it,4,17);
    cout<<"\n4**********************\n";
    print(l2);

    //5:insert l1 into l2
    l2.insert(i_it,l1.begin(),l1.end());
    cout<<"\n5**********************\n";
    print(l2);
    print(l1);

    //6:remove all 17s
    l2.remove(17);
    cout<<"\n6**********************\n";
    print(l2);

    //7:remove repetition
    l2.unique();
    cout<<"\n7**********************\n";
    print(l2);
    cin>>hold;
    return 0;
}
```

شاید این مثال به ظاهر طولانی باشد ولی با تقسیم آن به 8 مرحله (از صفر تا 7) و تحلیل هر قسمت می توان به سادگی آن پی برد.(مراحل در داخل کد برنامه شماره گذاری شده اند)

مرحله صفر:این مرحله فقط دو روش جدید برای ایجاد لیست را نشان می دهد.

مرحله یک:این مرحله نشان می دهد که چگونه باید i امین عنصر لیست را حذف کنیم و چگونه باید مقدار یک iterator مربوط به لیست را افزایش دهیم.

مرحله دو:چگونگی حذف یک بازه را نشان می دهد.

مرحله سه:مقدار 18 را در سومین مکان (با شروع از صفر) لیست درج می کنیم.

مرحله چهار:4 کپی از 17 را در مکان i_it درج می کند.نکته مهم این است که مقدار i_it در فراخوانی قبلی insert تغییر کرده است.

مرحله پنج:لیست l1 را در مکان i_it لیست l2 درج می کند.(به مکان فعلی i_it توجه کنید)،همچنین توجه کنید که این عمل تغییری در لیست l1 ایجاد نمی کند.

مرحله شش:تمام مقادیر 17 را از لیست حذف می کند.

مرحله هفت:تمام تکرار های متوالی را حذف می کند (به دو مقدار 8 باقی مانده در لیست توجه کنید)

خروجی برنامه:

```
0**********************

--------------------------------
3 3 3 3 3
--------------------------------

--------------------------------
1 2 8 7 6 3 4 4 4 5 8 8 11 9 10
--------------------------------

1**********************

--------------------------------
1 2 8 6 3 4 4 4 5 8 8 11 9 10
--------------------------------

2**********************

--------------------------------
1 2 8 6 5 8 8 11 9 10
--------------------------------
```

```
3**********

--------------------------------
1 2 8 18 6 5 8 8 11 9 10
--------------------------------

4**********

--------------------------------
1 2 8 18 17 17 17 17 6 5 8 8 11 9 10
--------------------------------

5**********

--------------------------------
1 2 8 18 17 17 17 17 3 3 3 3 3 6 5 8 8 11 9 10
--------------------------------

--------------------------------
3 3 3 3 3
--------------------------------

6**********

--------------------------------
1 2 8 18 3 3 3 3 3 6 5 8 8 11 9 10
--------------------------------

7**********

--------------------------------
1 2 8 18 3 6 5 8 11 9 10
--------------------------------
```

مثال 7:splice (از خروجی بفهمید!)

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

void print(list<int> p){
	cout<<"\n--------------------------------\n";
	if(p.empty()){
		cout<<"Empty";
	}else{
		list<int>::iterator start=p.begin();
		while(start!=p.end()){
			int p=(int)*start;
			cout<<p<<" ";
			start++;
		}
	}
	cout<<"\n--------------------------------\n";
}

list<int> l1,l2,l3;
```

```
int main(){

    int hold;
    //0:
    int x[]={1,2,8,7,6,3,4,4,4,5,8,8,11,9,10};
    int n=6;
    for(int i=0;i<n;i++){
            l1.push_back(23+i);
            l2.push_back(33+i);
    }
    l3=list<int>(x,x+sizeof(x)/sizeof(int));
    print(l1);
    print(l2);
    print(l3);
    l3.splice(l3.begin(),l1);
    print(l1);
    print(l3);

    list<int>::iterator i_it=l3.begin();
    list<int>::iterator j_it=l2.begin();
    list<int>::iterator k_it=l2.begin();
    advance(i_it,3),advance(j_it,1),advance(k_it,4);
    l3.splice(i_it,l2,j_it,k_it);
    print(l2);
    print(l3);

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
--------------------------------
23 24 25 26 27 28
--------------------------------

--------------------------------
33 34 35 36 37 38
--------------------------------

--------------------------------
1 2 8 7 6 3 4 4 4 5 8 8 11 9 10
--------------------------------

--------------------------------
Empty
--------------------------------

--------------------------------
23 24 25 26 27 28 1 2 8 7 6 3 4 4 4 5 8 8 11 9 10
--------------------------------

--------------------------------
33 37 38
--------------------------------

--------------------------------
23 24 25 34 35 36 26 27 28 1 2 8 7 6 3 4 4 4 5 8 8 11 9 10
--------------------------------
```

مثال 8:

در مثال های قبلی با sort و merge آشنا شدیم ،اما اگر عناصر لیست ساختارهایی باشند که خودمان آنها را تعریف کردیم آنگاه معیار مقایسه را نیز خودمان باید تعریف کنیم.

```
#include <iostream>
#include <list>
using namespace std;
typedef struct student{
        string name;
        int num;
};
void print(list<student> p){
     cout<<"\n--------------------------------\n";
     if(p.empty()){
           cout<<"Empty";
     }else{
           list<student>::iterator start=p.begin();
           while(start!=p.end()){
                student p=(student)*start;
                cout<<p.name<<"\t"<<p.num<<"\n";
                start++;
           }
     }
     cout<<"\n--------------------------------\n";
}
bool name_compare(student s1,student s2){
     return s1.name.compare(s2.name)>0?false:true;
}
bool num_compare(student s1,student s2){
     return s1.num>s2.num?false:true;
}
list<student> l1;
int main(){
    int hold;
    string names[]={"ali","reza","hasan","ali","alireza"};
    int nums[]=    {10   ,11    ,12     ,13   ,8   };
    for(int i=0;i<5;i++){
            student s;
            s.name=names[i];
            s.num=nums[i];
            l1.push_back(s);
    }
    l1.sort(name_compare);
    print(l1);
    l1.sort(num_compare);
    print(l1);

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
--------------------------------
ali      13
ali      10
alireza  8
hasan    12
reza     11

--------------------------------

--------------------------------
alireza  8
ali      10
reza     11
hasan    12
ali      13

--------------------------------
```

صف دو طرفه (Deque)

صف دو طرفه ساختمان داده ایست که ویژگی های list وvector را با هم ترکیب کرده است،یعنی هم می توان عناصر را به صورت کارایی به هر دو طرف آن اضافه کرد ،و هم می توان به عناصر با استفاده از اندیس آنها دسترسی داشت.

برای استفاده از این ساختمان داده باید ابتدا کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم.

```
#include <deque>
```

متد های مهم این ساختمان داده عبارتند از :

begin: یک iterator که به ابتدای صف اشاره می کند را بر می گرداند.

end: یک iterator که به انتهای صف اشاره می کند را بر می گرداند.

size: اندازه صف را بر می گرداند.

empty: برای تشخیص خالی بودن یا نبودن صف از این متد استفاده می کنیم.

at: برای دسترسی با استفاده از اندیس به یک عنصر از این متد استفاده کنیم ،تفاوت این متد با عمگر [] این است که این متد در صورت غیر مجاز بودن اندیس یک خطای out_of_range بر می گرداند.

front: عنصر ابتدای صف را بر می گرداند.

back: عنصر انتهای صف را بر می گرداند.

assign: محتویات یک container را به صف اختصاص می دهد.

push_back: برای اضافه کردن عنصر به انتهای صف از این متد استفاده می کنیم.

push_front: برای اضافه کردن عنصر به ابتدای صف از این متد استفاده می کنیم.

pop_back: عنصر انتهای صف را حذف می کند.

pop_front: عنصر ابتدای صف را حذف می کند.

insert: برای درج عنصر جدید از این متد استفاده می کنیم.

erase: برای حذف عناصر از این متد استفاده می کنیم.

swap: محتویات دو صف را با هم عوض می کند.

clear: محتویات صف را پاک می کند.

مثال 1:آشنایی با deque

```
#include <iostream>
#include <deque>
using namespace std;
deque<int> xdq;
deque<int> ydq;
void print(deque<int> dq){
	cout<<"\n--------------------\n";
	if(dq.size()==0){
		cout<<"Empty";
	}else{
		deque<int>::iterator first=dq.begin();
		while(first!=dq.end()){
			cout<<(int)*first<<" ";
			first++;
		}
	}
	cout<<"\n--------------------\n";
}
```

```
int main(){
    int x[]={2,4,6,8,10,12,14};
    int y[]={1,3,5,7,9,11,13};
    int hold;
    xdq=deque<int>(x,x+sizeof(x)/sizeof(int));
    ydq.assign(y,y+sizeof(y)/sizeof(int));
    print(xdq);
    print(ydq);
    int i=3;
    deque<int>::iterator i_it=xdq.begin()+i;
    xdq.insert(i_it,-2);
    i_it=xdq.begin()+i;//renew the i_it
    xdq.insert(i_it,3,-4);
    print(xdq);
    cout<<"y[3]: "<<ydq[3]<<"  "<<ydq.at(3);
    xdq.swap(ydq);
    print(xdq);
    print(ydq);
    cout<<"y[3]: "<<ydq[3]<<"  "<<ydq.at(3);
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
--------------------
2 4 6 8 10 12 14
--------------------

--------------------
1 3 5 7 9 11 13
--------------------

--------------------
2 4 6 -4 -4 -4 -2 8 10 12 14
--------------------
y[3]: 7  7
--------------------
1 3 5 7 9 11 13
--------------------

--------------------
2 4 6 -4 -4 -4 -2 8 10 12 14
--------------------
y[3]: -4  -4
```

مثال 2: تفاوت بین متد insert مربوط به deque و متد insert مربوط به list

هنگام استفاده از متد insert مربوط به deque بسیار مواظب iterator ها باشید!

```
#include <iostream>
#include <deque>
#include <list>
using namespace std;
deque<int> dq;
list<int> l;
```

```
void print(deque<int> dq){
     cout<<"\n--------------------\n";
     if(dq.size()==0){
            cout<<"Empty";
     }else{
           deque<int>::iterator first=dq.begin();
           while(first!=dq.end()){
                  cout<<(int)*first<<" ";
                  first++;
           }
     }
     cout<<"\n--------------------\n";
}

void print(list<int> l){
     cout<<"\n--------------------\n";
     if(l.size()==0){
            cout<<"Empty";
     }else{
           list<int>::iterator first=l.begin();
           while(first!=l.end()){
                  cout<<(int)*first<<" ";
                  first++;
           }
     }
     cout<<"\n--------------------\n";
}
int main(){
    int x[]={2,4,6,8,10,12,14};
    int hold;
    l=list<int>(x,x+sizeof(x)/sizeof(int));
    dq=deque<int>(x,x+sizeof(x)/sizeof(int));
    print(dq);
    print(l);
    int i=3;

    deque<int>::iterator i_it=dq.begin()+i;
    list<int>::iterator j_it=l.begin();
    advance(j_it,i);

    dq.insert(i_it,20);
    dq.insert(i_it,40);
    dq.insert(i_it,18);
    dq.insert(i_it,19);
    dq.insert(i_it,21);

    l.insert(j_it,20);
    l.insert(j_it,40);
    l.insert(j_it,18);
    l.insert(j_it,19);
    l.insert(j_it,21);


    print(dq);
    print(l);
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
--------------------
2 4 6 8 10 12 14
--------------------

--------------------
2 4 6 8 10 12 14
--------------------

--------------------
2 4 6 40 19 21 18 20 8 10 12 14
--------------------

--------------------
2 4 6 20 40 18 19 21 8 10 12 14
--------------------
```

مثال 3:تفاوت بین at و []

هنگامی که از [] برای دسترسی به عناصر استفاده کنیم ،اگر اندیس مورد نظر غیر معتبر باشد تغییری در روند اجرای برنامه ایجاد نمی شود ولی اگر از at استفاده کنیم خطای out_of_range برگردانده می شود.

```
#include <iostream>
#include <deque>
#include <stdexcept>
#include <stdlib.h> //or <cstdlib>
using namespace std;
deque<int> dq;

int main(){
    int hold;
    for(int i=0;i<5;i++){
            dq.push_back(rand());
    }
    int x;
    try{
        x=dq[7];//invalid access
        cout<<"HA?\n";
    }catch(out_of_range oor){
          cout<<"ONE !\n";
    }
    try{
        x=dq.at(7);//invalid access
        cout<<"What?\n";
    }catch(out_of_range oor){
          cout<<"TWO !\n";
    }

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
HA?
TWO !
```

مثال 4:تفاوت درج در انتها بین list وvector و deque

```
#include <iostream>
#include <deque>
#include <vector>
#include <list>
#include <ctime>
using namespace std;

deque<int> dq;
list<int> l;
vector<int> v;
int main(){
    int hold;
    int x=88888;
    int c=clock();
    for(int i=0;i<800000;i++){
            v.push_back(x);
    }
    cout<<"vecotr :"<<(clock()-c)<<"\n";
    c=clock();
    for(int i=0;i<800000;i++){
            l.push_back(x);
    }
    cout<<"list :"<<(clock()-c)<<"\n";
    c=clock();
    for(int i=0;i<800000;i++){
            dq.push_back(x);
    }
    cout<<"deque :"<<(clock()-c)<<"\n";

    cin>>hold;
    return 0;
}
```

نمونه ای از خروجی برنامه:

```
vecotr :16
list :297
deque :31
```

مثال 5:تفاوت درج در ابتدا بین list و deque

```cpp
#include <iostream>
#include <deque>
#include <list>
#include <ctime>
using namespace std;

deque<int> dq;
list<int> l;

int main(){
    int hold;
    int x=88888;
    int c=clock();

    for(int i=0;i<800000;i++){
            l.push_front(x);
    }
    cout<<"list :"<<(clock()-c)<<"\n";
    c=clock();
    for(int i=0;i<800000;i++){
            dq.push_front(x);
    }
    cout<<"deque :"<<(clock()-c)<<"\n";

    cin>>hold;
    return 0;
}
```

نمونه ای از خروجی برنامه:

```
list :281
deque :32
```

مثال 6:تفاوت دسترسی تصادفی به عناصر بین vector وdeque

```cpp
#include <iostream>
#include <deque>
#include <vector>
#include <ctime>
#include <new>
#include <stdlib.h>
using namespace std;

deque<int> dq;
vector<int> v;
```

```
int main(){
    int hold;
    int x;
    int c;
    int* rands;
    for(int i=0;i<50000;i++){
            v.push_back(1);
            dq.push_back(1);
    }
    cout<<RAND_MAX<<endl;//The maximum value
                         //that can be returned by the rand()
    try{
        rands=new int[400000];
    }catch(bad_alloc ba){
        cout << "Allocation Failure\n";
        cin>>hold;
        return 0;
    }
    for(int i=0;i<400000;i++){
             rands[i]=rand();
    }
    c=clock();
    for(int i=0;i<400000;i++){
           x=v[rands[i]];
    }
    cout<<"vector :"<<(clock()-c)<<endl;
    c=clock();
    for(int i=0;i<400000;i++){
           x=dq[rands[i]];
    }
    cout<<"deque :"<<(clock()-c)<<endl;
    cin>>hold;
    return 0;
}
```

نمونه ای از خروجی برنامه:

```
32767
vector :16
deque :62
```

مثال 7:تفاوت درج در مکان های تصادفی بین list و vector و deque

```
#include <iostream>
#include <deque>
#include <vector>
#include <ctime>
#include <list>
#include <stdlib.h>
using namespace std;

deque<int> dq;
vector<int> v;
list<int> l;
```

```
int main(){
    int hold;
    int x=2;
    int c;
    int rands[10000];
    for(int i=0;i<50000;i++){
            v.push_back(1);
            dq.push_back(1);
            l.push_back(1);
    }
    for(int i=0;i<10000;i++)
            rands[i]=rand();

    vector<int>::iterator v_it;
    list<int>::iterator l_it;
    deque<int>::iterator dq_it;
    c=clock();
    for(int i=0;i<10000;i++){
            v_it=v.begin()+rands[i];
            v.insert(v_it,x);
    }
    cout<<"vector :"<<(clock()-c)<<"\n";
    c=clock();
    for(int i=0;i<10000;i++){
            l_it=l.begin();
            advance(l_it,rands[i]);
            l.insert(l_it,x);
    }
    cout<<"list :"<<(clock()-c)<<"\n";
    c=clock();
    for(int i=0;i<10000;i++){
            dq_it=dq.begin()+rands[i];
            dq.insert(dq_it,x);
    }
    cout<<"deque :"<<(clock()-c)<<"\n";


    cin>>hold;
    return 0;
}
```

نمونه ای از خروجی برنامه:

```
vector :531
list :1922
deque :5094
```

صف (Queue)

صف یک طرفه یا به اختصار همان صف، ساختمان داده است با ویژگی First-In-First-Out.

برای استفاده از این ساختمان داده باید ابتدا کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم.

```
#include <queue>
```

این ساختمان داده یک ساختمان داده مستقل نیست بلکه از ساختمان داده های دیگر به عنوان container استفاده می کند (list یا deque)،اگر برنامه نویس container ای برای آن مشخص نکند این کلاس به صورت پیش فرض از deque استفاده می کند.
متدهای مهم این ساختمان داده عبارتند از:

empty:	برای تشخیص خالی بودن یا نبودن صف از این متد استفاده می کنیم.
size :	اندازه صف را بر می گرداند.
front:	عنصر ابتدای صف را بر می گرداند.
back:	عنصر انتهای صف را بر می گرداند.
push:	عنصر جدید را به ابتدای صف اضافه می کند.
pop:	عنصر ابتدای صف را حذف می کند.

مثال 1:آشنایی با صف

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

queue<int> q;

int main()
{
    int hold;
    for(int i=0;i<5;i++)
            q.push(i);
    cout<<"Size of Queue :"<<q.size()<<"\n";
    cout<<"Front of Queue:"<<q.front()<<"\n";
    cout<<"Back of Queue:"<<q.back()<<"\n";
    while(!q.empty()){
             cout<<q.front()<<"\n";
             q.pop();

    }

    cin>>hold;

      return 0;
}
```

خروجی برنامه:

```
Size of Queue :5
Front of Queue:0
Back of Queue:4
0
1
2
3
4
```

مثال 2:استفاده از کلاس list به عنوان container

```
#include <iostream>
#include <queue>
#include <list>
#include <ctime>
using namespace std;

queue<int> q1;
queue<int,list<int> > q2;//CAUTION! write queue<int,list<int> > not
                                        //queue<int,list<int>>
int main()
{
    int hold;
    cout<<"q1:\n";
    int c=clock();
    for(int i=0;i<800000;i++)
            q1.push(i);
    cout<<(clock()-c)<<"\n";
    c=clock();
    while(!q1.empty()){
            q1.pop();
    }
    cout<<(clock()-c)<<"\n";
    cout<<"q2:\n";
    c=clock();
    for(int i=0;i<800000;i++)
            q2.push(i);
    cout<<(clock()-c)<<"\n";
    c=clock();
    while(!q2.empty()){
            q2.pop();
    }
    cout<<(clock()-c)<<"\n";

    cin>>hold;
      return 0;
}
```

نمونه ای از خروجی برنامه:

```
q1:
31
32
q2:
328
281
```

صف اولویت (Priority Queue)

گاهی اوقات نیاز داریم که در یک مجموعه بتوانیم به عنصر با اولویت بیشتر به صورتی کارا دسترسی داشته باشیم (برای مثال در پیاده سازی الگوریتم prime یا dijkestra یا....)،صف اولویت ساختمان داده ای است که قابلیت فوق را در اختیار ما قرار می دهد.(این ساختمان داده به صورت max heap پیاده سازی شده است). برای استفاده از این ساختمان داده باید ابتدا کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم.

```
#include <queue>
```

این ساختمان داده یک ساختمان داده مستقل نیست بلکه از ساختمان داده های دیگر به عنوان container استفاده می کند (vector یا deque)،اگر برنامه نویس container ای برای آن مشخص نکند این کلاس به صورت پیش فرض از vector استفاده می کند.(هر ساختمان داده ای که بخواهید از آن به عنوان container برای صف اولویت استفاده کنید باید از دسترسی تصادفی پشتیبانی کند و متد های front()، push_back() و pop_back() را داشته باشد.)

متد های مهم این ساختمان داده عبارتند از :

empty: برای تشخیص خالی بودن یا نبودن صف اولویت از این متد استفاده می کنیم.

size: اندازه صف اولویت را بر می گرداند.

top: عنصر با اولویت بیشتر را بر می گرداند.

push: عنصر جدید را به مجموعه اضافه می کند.

pop: عنصر با اولویت بیشتر را از مجموعه حذف می کند.

مثال 1:

```
#include <iostream>
#include <queue>
#define int_size(x) sizeof(x)/sizeof(int)
using namespace std;

priority_queue<int> pq;
int main()
{
    int hold;
    int x[]={1,6,2,4,7,3,5,12,9};
    for(int i=0;i<int_size(x);i++){
            pq.push(x[i]);
    }

    cout<<"Size of Priority Queue: "<<pq.size()<<"\n";
    while(!pq.empty()){
           cout<<pq.top()<<" ";
           pq.pop();
    }
    cin>>hold;
      return 0;
}
```

خروجی برنامه:

```
Size of Priority Queue: 9
12 9 7 6 5 4 3 2 1
```

مثال 2:

اگر عناصر صف اولویت ساختار داده هایی باشند که خودمان تعریف کردیم آنگاه باید هم container و هم معیار مقایسه را صریحاً مشخص کنیم.(معیار مقایسه یک class یا struct است که () را overload کرده است)

```
#include <iostream>
#include <queue>
#include <deque>
using namespace std;
typedef struct student{
        string name;
        int num;
};
class NameCompare {
public:
    bool operator()(student s1,student s2)
    {
         if(s1.name.compare(s2.name)>0){
              return true;
         }
         return false;
    }
};
struct NumCompare {
    bool operator()(student s1,student s2)
    {
         if(s1.num>s2.num){
              return true;
         }
         return false;
    }
};
priority_queue<student,vector<student>,NameCompare> pq1;
priority_queue<student,deque<student>,NumCompare> pq2;
int main()
{
    int hold;
    string name[]={"a","b","c","d","e"};
    int num[]=    {2  ,4  ,1  ,5  ,3};
    for(int i=0;i<5;i++){
            student p;
            p.name=name[i];
            p.num=num[i];
            pq1.push(p);  pq2.push(p);
    }
    while(!pq1.empty()){
          student p=pq1.top();
          cout<<p.name<<" "<<p.num<<"\n";
          pq1.pop();
    }
    cout<<"-----\n";
```

```
    while(!pq2.empty()){
          student p=pq2.top();
          cout<<p.name<<" "<<p.num<<"\n";
          pq2.pop();
    }
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
a 2
b 4
c 1
d 5
e 3
-----
c 1
a 2
e 3
b 4
d 5
```

گروه(Set)

گروه ساختمان داده ایست با ویژگی های زیر:

1) عناصر بر اساس ترتیب خاصی نگه داری می شوند.
2) عناصر به صورت منحصر به فرد در آن قرار می گیرند(از هر عنصر فقط یکی).
3) پیدا کردن یک عنصر و پیدا کردن عنصر بعدی به صورتی کارا در آن انجام می شود(چون گروه معمولاً به صورت درخت جست و جوی دودویی پیاده سازی می شود ،عملیات ذکر شده در زمان لگاریتمی انجام می شوند.)

معمولاً از گروه در جاهایی که نیاز به هرس کردن داریم (زیرا عناصر قبل و بعد از یک عنصر در زمانی کارا مشخص می شوند) یا مواقعی که نیاز داریم عضویت یا عدم عضویت یک کلید در یک مجموعه را مشخص کنیم استفاده می کنیم.

برای استفاده از این ساختمان داده باید ابتدا کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم.

```
#include <set>
```

متد های مهم این ساختمان داده عبارتند از:

begin: یک iterator که به ابتدای گروه اشاره می کند را بر می گرداند.

end: یک iterator که به انتهای گروه اشاره می کند را بر می گرداند.

empty: برای تشخیص خالی بودن یا نبودن گروه از این متد استفاده می کنیم.

size: اندازه گروه را بر می گرداند.

insert: برای درج عنصر جدید به کار می رود.

erase: یک عنصر را از گروه حذف می کند.

clear: گروه را پاک می کند.

swap: محتویات گروه را با گروهی دیگر جا به جا می کند.

find: برای پیدا کردن یک عنصر خاص از این متد استفاده می کنیم.

count: تعداد تکرار های یک عنصر را بر می گرداند.(از آنجایی که در گروه از هر عنصر فقط یکی نگه داری می شود از این متد می توانیم برای تشخیص عضویت یا عدم عضویت یک عنصر استفاده کنیم)

lower_bound: یکiterator که به عنصر بزرگتر یا مساوی یک کلید اشاره می کند را بر می گرداند.

upper_bound: یکiterator که به عنصر منحصراً بزرگتر از یک کلید اشاره می کند را بر می گرداند.

مثال 1:آشنایی با گروه

مثال زیر نشان می دهد که گروه عناصر تکراری را نگه داری نمی کند و عناصر بر اساس ترتیب خاصی در آن نگه داری می شوند.

```
#include <iostream>
#include <set>
#define int_size(x) sizeof(x)/sizeof(int)
using namespace std;
set<int> s;
int main()
{
    int hold;
    int x[]={1,3,5,2,7,4,9,8,2,1,5,5};
    for(int i=0;i<int_size(x);i++){
            s.insert(x[i]);
    }
    cout<<"Size of Set :"<<s.size()<<"\n";
    set<int>::iterator start=s.begin();
    while(start!=s.end()){
          cout<<(int)*start<<" ";
          start++;
    }
    cin>>hold;
      return 0;
}
```

خروجی برنامه:

```
Size of Set :8
1 2 3 4 5 7 8 9
```

مثال 2:

از متد find برای پیدا کردن یک عنصر خاص استفاده می کنیم،اگر عنصر مورد نظر در گروه وجود داشته باشد این متد یک iterator که به آن عنصر اشاره می کند را بر می گرداند در غیر اینصورت یک iterator که به انتهای گروه اشاره می کند را بر می گرداند.همچنین مثال زیر نشان می دهد که چگونه می توانیم از count برای تشخیص عضویت یا عدم عضویت یک عنصر استفاده کنیم.

```cpp
#include <iostream>
#include <set>
#define int_size(x) sizeof(x)/sizeof(int)
using namespace std;
set<int> s;
void print(set<int>::iterator a,set<int>::iterator last){
    if(a!=last){
           cout<<(int)*a<<" is a member of set\n";
    }else{
          cout<<"Not Found!\n";
    }
}
void yes_no(int x){
     x==0 ? cout<<"NO\n" : cout<<"YES\n";
}
int main()
{
    int hold;
    int x[]={1,3,5,2,7,4,9,8,2,1,5,5};
    for(int i=0;i<int_size(x);i++){
            s.insert(x[i]);
    }
    cout<<"Size of Set :"<<s.size()<<"\n";
    set<int>::iterator f_it=s.find(7);
    print(f_it,s.end());
    f_it=s.find(6);
    print(f_it,s.end());
    yes_no(s.count(7));
    yes_no(s.count(6));

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
Size of Set :8
7 is a member of set
Not Found!
YES
NO
```

مثال 3:تفاوت بین lower_bound و upper_bound

```
#include <iostream>
#include <set>
#define int_size(x) sizeof(x)/sizeof(int)
using namespace std;
set<int> s;

int main()
{
    int hold;
    int x[]={1,3,5,2,7,4,9,8,2,1,5,5};
    for(int i=0;i<int_size(x);i++){
            s.insert(s.begin(),x[i]);
    }

    cout<<"Size of Set :"<<s.size()<<"\n";
    set<int>::iterator start=s.begin();
    while(start!=s.end()){
          cout<<(int)*start<<" ";
          start++;
    }
    cout<<"\n";
    set<int>::iterator lb_it=s.lower_bound(6);
    set<int>::iterator ub_it=s.upper_bound(6);
    cout<<"for 6\n";
    cout<<(int)*lb_it<<"\n";
    cout<<(int)*ub_it<<"\n";
    cout<<"--\n";
    cout<<"for 7\n";
    lb_it=s.lower_bound(7);
    ub_it=s.upper_bound(7);
    cout<<(int)*lb_it<<"\n";
    cout<<(int)*ub_it<<"\n";

    cin>>hold;
      return 0;
}
```

خروجی برنامه:

```
Size of Set :8
1 2 3 4 5 7 8 9
for 6
7
7
--
for 7
7
8
```

مثال 4:باز هم upper_bound وlower_bound

```
#include <iostream>
#include <set>
#define int_size(x) sizeof(x)/sizeof(int)
using namespace std;
set<int> s;
int main()
{
    int hold;
    int x[]={1,3,5,2,7,4,9,8,2,1,5,5};
    for(int i=0;i<int_size(x);i++){
            s.insert(s.begin(),x[i]);
    }

    cout<<"Size of Set :"<<s.size()<<"\n";
    set<int>::iterator start=s.begin();
    set<int>::iterator lb_it,ub_it;

    while(start!=s.end()){
          cout<<(int)*start<<" ";
          start++;
    }
    cout<<"\n";
    lb_it=s.lower_bound(7);
    ub_it=s.upper_bound(7);

    while(ub_it!=s.end()){
           cout<<(int)*ub_it<<" ";
           ub_it++;
    }
    cout<<"\n";
    start=s.begin();
    while(start!=lb_it){
          cout<<(int)*start<<" ";
          start++;
    }
    cin>>hold;
      return 0;
}
```

خروجی برنامه:

```
Size of Set :8
1 2 3 4 5 7 8 9
8 9
1 2 3 4 5
```

مثال 5:erase

از erase برای پاک کردن یک عنصر یا یک بازه استفاده می کنیم،اگر فقط بخواهیم یک عنصر را حذف کنیم هم می توانیم مستقیماً از خود عنصر به عنوان پارامتر این متد استفاده کنیم و هم اینکه از iterator استفاده کنیم ولی اگر هدف حذف یک بازه باشد حتماً باید از iterator استفاده کنیم.

```
#include <iostream>
#include <set>
#define int_size(x) sizeof(x)/sizeof(int)
using namespace std;
set<int> s;
void print(set<int> s){
    set<int>::iterator start=s.begin();
    while(start!=s.end()){
          cout<<(int)*start<<" ";
          start++;
    }
    cout<<"\n";
}
int main()
{
    int hold;
    int x[]={1,3,5,2,7,4,9,8,2,1,5,5};
    for(int i=0;i<int_size(x);i++){
            s.insert(s.begin(),x[i]);
    }
    print(s);
    s.erase(5);
    print(s);
    set<int>::iterator it=s.find(7);
    s.erase(it);
    print(s);
    set<int>::iterator a=s.find(3);
    set<int>::iterator b=s.find(8);
    s.erase(a,b);
    print(s);

    cin>>hold;
      return 0;
}
```

خروجی برنامه:

```
1 2 3 4 5 7 8 9
1 2 3 4 7 8 9
1 2 3 4 8 9
1 2 8 9
```

مثال 6:هشدار!

اگر قبل از استفاده از erase از find برای به دست آوردن iterator استفاده می کنید ،حتماً مطمئن شوید که عنصر مورد نظر پیدا شده است و iterator به دست آمده معتبر است ،در غیر این صورت استفاده از erase منجر به نتایج نا خواسته ای می شود.

```
#include <iostream>
#include <set>
#define int_size(x) sizeof(x)/sizeof(int)
using namespace std;
set<int> s;
void print(set<int> s){
    set<int>::iterator start=s.begin();
    while(start!=s.end()){
          cout<<(int)*start<<" ";
          start++;
    }
    cout<<"\n";
}
int main()
{
    int hold;
    int x[]={1,3,5,2,7,4,9,8,2,1,5,5};
    for(int i=0;i<int_size(x);i++){
            s.insert(s.begin(),x[i]);
    }
    print(s);
    s.erase(6);
    print(s);
    set<int>::iterator it=s.find(6);
    s.erase(it);
    print(s);
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
1 2 3 4 5 7 8 9
1 2 3 4 5 7 8 9
1 2 3 1 2 9
```

اصلاح برنامه قبلی:

به شرطی که قبل از فراخوانی erase اضافه شده توجه کنید.

```
#include <iostream>
#include <set>
#define int_size(x) sizeof(x)/sizeof(int)
using namespace std;
set<int> s;
void print(set<int> s){
    set<int>::iterator start=s.begin();
    while(start!=s.end()){
          cout<<(int)*start<<" ";
          start++;
    }
    cout<<"\n";
}
int main()
{
    int hold;
    int x[]={1,3,5,2,7,4,9,8,2,1,5,5};
    for(int i=0;i<int_size(x);i++){
            s.insert(s.begin(),x[i]);
    }
    print(s);
    s.erase(6);
```

```
    print(s);
    set<int>::iterator it=s.find(6);
    if(it!=s.end())
        s.erase(it);
    print(s);
    cin>>hold;
    return 0;
}
```

خروجی برنامه اصلاح شده:

```
1 2 3 4 5 7 8 9
1 2 3 4 5 7 8 9
1 2 3 4 5 7 8 9
```

مثال 7:

اگر عناصر گروه ساختار داده هایی باشند که خودمان تعریف کردیم آنگاه باید معیار مقایسه را صریحاً مشخص کنیم.(معیار مقایسه یک class یا struct است که () را overload کرده است)

توجه کنید که در این حالت نیز گروه عناصر تکراری را نگه داری نمی کند.

```
#include <iostream>
#include <set>
#include <string>
using namespace std;
typedef struct student{
        string name;
        int num;
};
struct student_compare{
       bool operator() (student s1,student s2){
            if(s1.num<s2.num){
                return true;
            }else if(s1.num==s2.num){
                  if(s1.name.compare(s2.name)<0){
                     return true;
                  }else{
                     return false;
                  }
            }else{
                  return false;
            }
       }
};
set<student,student_compare> s;
void print(set<student,student_compare> s){
    set<student,student_compare>::iterator start=s.begin();
    while(start!=s.end()){
          student p=(student)*start;
          cout<<p.num<<" "<<p.name<<"\n";
          start++;
    }
    cout<<"\n";
}
```

```
int main()
{
    int hold;
    string names[]=
            {"ali","reza","ali","ahmad","hasan","ali","taghi","jamshid!"};
    int nums[]={10   ,11    ,12   , 12    ,13     ,10   ,9      ,0  };
    for(int i=0;i<8;i++){
             student p;
             p.name=names[i];
             p.num=nums[i];
             s.insert(p);
    }
    print(s);
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
0 jamshid!
9 taghi
10 ali
11 reza
12 ahmad
12 ali
13 hasan
```

گروه چندگانه(Multiset)

گروه چند گانه نیز مانند گروه است با این تفاوت که عناصر تکراری را نیز نگه داری می کند.

برای استفاده از این ساختمان داده باید ابتدا کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم.

```
#include <set>
```

متد های مهم این ساختمان داده عبارتند از:

begin: یک iterator که به ابتدای گروه چندگانه اشاره می کند را بر می گرداند.

end: یک iterator که به انتهای گروه چندگانه اشاره می کند را بر می گرداند.

empty: برای تشخیص خالی بودن یا نبودن گروه چندگانه از این متد استفاده می کنیم.

size: اندازه گروه چندگانه را بر می گرداند.

insert: برای درج عنصر جدید به کار می رود.

erase: یک عنصر را از گروه چندگانه حذف می کند.

clear: گروه چندگانه را پاک می کند.

swap: محتویات گروه چندگانه را با گروه چندگانه دیگری جا به جا می کند.

find: برای پیدا کردن یک عنصر خاص از این متد استفاده می کنیم.

count: تعداد تکرار های یک عنصر را بر می گرداند.

lower_bound: یکiterator که به عنصر بزرگتر یا مساوی یک کلید اشاره می کند را بر می گرداند.

upper_bound: یکiterator که به عنصر منحصراً بزرگتر از یک کلید اشاره می کند را بر می گرداند.

equal_range: یک جفت (pair) بر می گرداند که این جفت ابتدا و انتهای یک بازه را که عناصر آن مساوی عنصر مورد نظر هستند ،مشخص می کند.

*جفت (pair) ساختار داده ایست که برای نگه داری یک مجموعه دو عنصری از آن استفاده می شود.عنصر اول این مجموعه با first و عنصر دوم آن با second مشخص می شود.

مثال 1:استفاده از pair

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int hold;
    pair<int,int> int_p;
    pair<string,string> str_p;
    int_p.first=10;
    int_p.second=12;

    str_p.first="ali";
    str_p.second="reza";

    cout<<int_p.second<<endl;
    cout<<int_p.first<<endl;

    cout<<str_p.second<<endl;
    cout<<str_p.first<<endl;

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
12
10
reza
ali
```

مثال 2:آشنایی با multiset

```
#include <iostream>
#include <set>
#define int_size(y) sizeof(y)/sizeof(int)
using namespace std;
multiset<int> ms;
typedef multiset<int>::iterator ms_iter;
```

```
int main()
{
    int hold;
    int x[]={1,4,5,2,12,7,9,12,14,13,7,12,15,12};
    for(int i=0;i<int_size(x);i++){
            ms.insert(x[i]);
    }
    ms_iter start=ms.begin();
    while(start!=ms.end()){
           cout<<(int)*start<<"\t"<<ms.count((int)*start)<<"\n";
           start++;
    }

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
1       1
2       1
4       1
5       1
7       2
7       2
9       1
12      4
12      4
12      4
12      4
13      1
14      1
15      1
```

مثال 3:equal_range

همانطور که اشاره شد equal_range یک جفت (pair) بر می گرداند که این جفت ابتدا و انتهای یک بازه را که عناصر آن مساوی عنصر مورد نظر هستند ،مشخص می کند.توجه کنید که عنصر دوم این جفت یا به انتهای گروه چندگانه اشاره می کند (مانند p2 در مثال زیر) یا به ابتدای بازه ای که عناصر آن منحصراً از عنصر مورد نظر بزرگترند(مانند p1 و p3).اگر بازه ای برای عنصر مورد نظر وجود نداشته باشد عنصر اول به همانجایی که عنصر دوم اشاره می کند، اشاره می کند.

```
#include <iostream>
#include <set>
#define int_size(y) sizeof(y)/sizeof(int)

using namespace std;
multiset<int> ms;
typedef multiset<int>::iterator ms_iter;
```

```
void print(pair<ms_iter,ms_iter> p,ms_iter last){
    cout<<"\n-------------\n";
    if(p.first==p.second)
        cout<<"No!!!!!!";
    else {
         while(p.first!=p.second){
             cout<<(int)*p.first<<" ";
             p.first++;
         }
    }
    cout<<"\n";

    if(p.second!=last)
          cout<<(int)*p.second;
    cout<<"\n-------------\n";
}
int main()
{
    int hold;
    int x[]={1,4,5,2,12,7,9,12,14,13,7,12,15,12};
    for(int i=0;i<int_size(x);i++){
            ms.insert(x[i]);
    }
    pair<ms_iter,ms_iter> p0;
    p0.first=ms.begin();
    p0.second=ms.end();
    print(p0,ms.end());
    pair<ms_iter,ms_iter> p1=ms.equal_range(8);
    pair<ms_iter,ms_iter> p2=ms.equal_range(15);
    pair<ms_iter,ms_iter> p3=ms.equal_range(12);
    print(p1,ms.end());
    print(p2,ms.end());
    print(p3,ms.end());
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
-------------
1 2 4 5 7 7 9 12 12 12 12 13 14 15

-------------

-------------
No!!!!!!
9
-------------

-------------
15

-------------

-------------
12 12 12 12
13
-------------
```

گروه-بیت (Bitset)

ساختمان داده ایست برای نگه داری بیت!

برای استفاده از این ساختمان داده باید ابتدا کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم.

```
#include <bitset>
```

این ساختمان داده از دستیابی تصادفی پشتیبانی می کند.

متد های مهم این ساختمان داده عبارتند از:

set: از این متد برای یک کردن تمام مجموعه،یک کردن یک بیت یا مقدار دهی یک بیت استفاده می شود.

reset: از این متد برای صفر کردن تمام مجموعه یا صفر کردن یک بیت استفاده می شود.

flip: از این متد برای معکوس کردن کل مجموعه یا فقط یک بیت استفاده می شود.(صفر را به یک و یک را به صفر تبدیل می کند)

to_ulong: یک unsigned long که مقدار آن معادل مقدار مجموعه باشد بر می گرداند،اگر مقدار مجموعه طوری باشد که در یک unsigned long جای نگیرد این متد یک خطای overflow_error بر می گرداند.

to_string: برای تبدیل مجموعه به رشته ای متوالی از صفر و یک ها استفاده می شود.

count: تعداد یک های مجموعه را بر می گرداند.

size: اندازه مجموعه را بر می گرداند.

test: برای چک کردن مقدار یک موقعیت به کار می رود ،در صورتی که موقعیت مورد نظر خارج از محدوده باشد این متد بر خلاف عملگر [] یک خطای out_of_range بر می گرداند.

any: تشخیص می دهد که آیا حداقل یک عضو از مجوعه دارای مقدار یک است یا نه؟

none: تشخیص می دهد که آیا تمام عضو های مجموعه صفر هستند؟

مثال 1:روش های ایجاد bitset ،دستیابی تصادفی

مثال زیر سه روش استفاده از گروه-بیت را نشان می دهد.

توجه کنید که معادل دودویی عدد 49 مقدار 110001 و مقدار 111010 معادل دودویی 58 است.همچنین توجه کنید که در متد print عناصر از آخر به اول چاپ می شوند(یعنی آخرین بیت معادل بیت پر ارزش و اولین بیت معادل بیت کم ارزش است).

```
#include <iostream>
#include <bitset>
using namespace std;
bitset<12> bs1;
bitset<12> bs2(49);
bitset<12> bs3(string("0111010"));
```

```
void print(bitset<12> bs){
     for(int i=bs.size()-1;i>=0;i--){
             cout<<bs[i];
     }
     cout<<"\n";
}

int main()
{
    int hold;
    print(bs1);
    print(bs2);
    print(bs3);
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
000000000000
000000110001
000000111010
```

مثال 2:

```
#include <iostream>
#include <bitset>
#include <stdexcept>
using namespace std;

bitset<70> bs1(string("0111010"));
bitset<70>
bs2(string("1111111111111111111111111111111101111101010101010101"));
void print(bitset<70> bs){
     for(int i=bs.size()-1;i>=0;i--){
             cout<<bs[i];
     }
     cout<<"\n";
}
int main()
{
    int hold;
    print(bs1);
    print(bs2);
    try{
        unsigned long x=bs1.to_ulong();
        cout<<x<<endl;
    }catch(overflow_error){
          cout<<"OverFlow1";
    }
    try{
        unsigned long x=bs2.to_ulong();
        cout<<x<<endl;
    }catch(overflow_error){
          cout<<"OverFlow2";
    }
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
00000000000000000000000000000000000000000000000000000000000111010
00000000000000000001111111111111111111111111111111101111101010101010101
58
OverFlow2
```

مثال 3:استفاده از reset، set

به سه شیوه می توان از متد set استفاده کرد:

1) بدون پارامتر:در این حالت کل مجموعه یک می شود.
2) فقط با یک پارامتر:در این حالت پارامتر ورودی موقعیتی از مجموعه است که باید یک شود.
3) با دو پارامتر:در این حالت پارامتر اول موقعیت مورد نظر و پارامتر دوم مقداری که باید به آن موقعیت نسبت داده شود را مشخص می کند.

به دو شیوه می توان از reset استفاده کرد:

1) بدون پارامتر:در این حالت کل مجموعه صفر می شود.
2) فقط با یک پارامتر: در این حالت پارامتر ورودی موقعیتی از مجموعه است که باید صفر شود.

```cpp
#include <iostream>
#include <bitset>

using namespace std;

bitset<10> bs1(string("0111010"));
void print(bitset<10> bs){
     for(int i=bs.size()-1;i>=0;i--){
             cout<<bs[i];
     }
     cout<<"\n";
}
int main()
{
    int hold;
    print(bs1);
    bs1.set(0);
    print(bs1);
    bs1.set(3,0);
    print(bs1);
    bs1.set();
    print(bs1);
    bs1.reset(3);
    print(bs1);
    bs1.reset();
    print(bs1);
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
0000111010
0000111011
0000110011
1111111111
1111110111
0000000000
```

مثال 4:test،count

```
#include <iostream>
#include <bitset>
#include <stdexcept>
using namespace std;

bitset<10> bs1(string("0111010"));
void print(bitset<10> bs){
     for(int i=bs.size()-1;i>=0;i--){
             cout<<bs[i];
     }
     cout<<"\n";
}
int main()
{
    int hold;
    print(bs1);
    cout<<"Size of Bitset: "<<bs1.size()<<"\n";
    cout<<"Number of Ones in Bitset: "<<bs1.count()<<"\n";
    cout<<"Number of Zeros in Bitset: "<<bs1.size()-bs1.count()<<"\n";


    try{
        cout<<bs1.test(1)<<"\n";
    }catch(out_of_range){cout<<"Out of Range1";}
    try{
        cout<<bs1.test(0)<<"\n";
    }catch(out_of_range){cout<<"Out of Range2";}
    try{
        cout<<bs1.test(12)<<"\n";
    }catch(out_of_range){cout<<"Out of Range3";}

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
0000111010
Size of Bitset: 10
Number of Ones in Bitset: 4
Number of Zeros in Bitset: 6
1
0
Out of Range3
```

مثال 5:flip،none،any

```
#include <iostream>
#include <bitset>
#include <stdexcept>
using namespace std;

bitset<7> bs1(string("0000100"));
void print(bitset<7> bs){
     for(int i=6;i>=0;i--){
             cout<<bs[i];
     }
     cout<<"\n";
}
```

```
int main()
{
    int hold;
    print(bs1);
    cout<<bs1.any()<<"\n";
    cout<<bs1.none()<<"\n";
    bs1.flip(2);
    print(bs1);
    cout<<bs1.any()<<"\n";
    cout<<bs1.none()<<"\n";
    bs1.flip();
    print(bs1);
    cout<<bs1.any()<<"\n";
    cout<<bs1.none()<<"\n";
    bs1.flip(2);
    print(bs1);
    cout<<bs1.any()<<"\n";
    cout<<bs1.none()<<"\n";

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
0000100
1
0
0000000
0
1
1111111
1
0
1111011
1
0
```

نگاشت (Map)

نگاشت ها ساختمان داده هایی هستند که برای نگه داری جفت (کلید،مقدار) استفاده می شوند،این ساختمان داده دارای سه ویژگی زیر است:

1) کلید ها منحصر به فرد هستند.
2) کلید ها بر اساس ترتیب خاصی در مجموعه قرار می گیرند.
3) این ساختمان داده طوری طراحی شده است که پیدا کلید در آن به صورتی کارا انجام گیرد.

همچنین این ساختمان از دستیابی تصادفی پشتیبانی می کند(بدون نیاز به iterator)،البته زمان دستیابی لگاریتمی است.

برای استفاده از این ساختمان داده باید ابتدا کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم.

```
#include <map>
```

متدهای مهم این ساختمان داده عبارتند از:

begin: یک iterator که به ابتدای مجموعه اشاره می کند را بر می گرداند.

end: یک iterator که به انتهای مجموعه اشاره می کند را بر می گرداند.

size: اندازه مجوعه (تعداد کلید ها) را بر می گرداند.

empty: برای تشخیص خالی بودن یا نبودن مجموعه از این متد استفاده می کنیم.

insert: برای درج عنصر جدید به کار می رود.

erase: برای پاک کردن عنصر به کار می رود.

swap: محتویات نگاشت را با محتویات نگاشت دیگری جا به جا می کند.

clear: مجموعه را پاک می کند.

find: برای پیدا کردن یک عنصر خاص به کار می رود.

count: تعداد عناصر با کلیدی خاص را می شمرد.

lower_bound: یکiterator که به عنصر بزرگتر یا مساوی یک کلید اشاره می کند را بر می گرداند.

upper_bound: یکiterator که به عنصر منحصراً بزرگتر از یک کلید اشاره می کند را بر می گرداند.

equal_range: یک جفت (pair) بر می گرداند که این جفت ابتدا و انتهای یک بازه را که عناصر آن مساوی عنصر مورد نظر هستند ،مشخص می کند.

مثال 1:آشنایی با نگاشت

همانطور که در مثال زیر مشاهده می کنید ،کلید ها در نگاشت منحصر به فرد هستند و عناصر بر اساس ترتیب کلید ها نگه داری می شوند،همچنین توجه کنید که اگر یک جفت (کلید،مقدار) در مجموعه وجود داشته باشد از درج مقادیر جدید با کلید مشابه صرف نظر می شود(به جفت (ali,12) توجه کنید).

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

map<string,int> mp;
void print(map<string,int> m){
	cout<<"\n---------------\n";
	if(m.empty()){
		cout<<"Empty\n";
	}else{
		map<string,int>::iterator start=m.begin();
		while(start!=m.end()){
			pair<string,int> p=(pair<string,int>)*start;
			cout<<p.first<<"\t"<<p.second<<"\n";
			start++;
		}
	}
	cout<<"\n---------------\n";
}
```

```
int main()
{
    int hold;
    string key[]={"ali","reza","hasan","ali","taghi"};
    int  value[]={ 12  ,10    ,12     ,13   ,11 };
    for(int i=0;i<5;i++)
            mp.insert(pair<string,int>(key[i],value[i]));
    print(mp);
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
---------------
ali     12
hasan   12
reza    10
taghi   11

---------------
```

مثال 2:نگاشت ها از دستیابی تصادفی پشتیبانی می کنند.

هنگام استفاده از عمگر [] برای درج یا دستیابی به عناصر باید مواظب موارد زیر باشید:

1) هر بار که از [] به همراه عملگر = برای درج عنصر جدید استفاده می کنیم،در صورت وجود کلید،مقدار جدید جایگزین مقدار قبلی می شود(بر خلاف متد insert).
2) اگر از [] برای دسترسی به کلیدی که در مجموعه وجود ندارد استفاده کنید،این کلید به همراه یک مقدار پیش فرض به مجموعه اضافه می شود(برای جلوگیری از این مشکل باید از find استفاده کنید).

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

map<string,int> mp;

int main()
{
    int hold;
    mp["ali"]=12;
    mp["reza"]=10;
    mp["ali"]=13;
    mp.insert(pair<string,int>("ali",14));
    cout<<"ali "<<mp["ali"]<<"\n";
    cout<<"reza "<<mp["reza"]<<"\n";
    cout<<"hasan "<<mp["hasan"]<<"\n";
    cout<<mp.size();

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
ali 13
reza 10
hasan 0
3
```

مثال 3:find،count،erase

از count برای شمارش تعداد تکرار های یک کلید خاص استفاده می شود،ولی از آنجایی که در نگاشت کلید ها منحصر به فرد هستند این متد در صورت وجود کلید، 1 و در صورت عدم وجود آن 0 بر می گرداند.

از find برای به دست آوردن یک iterator که به کلید مورد نظر اشاره می کند،استفاده می شود در صورتی که کلید مورد نظر در مجموعه وجود نداشته باشد یک iterator که به انتهای مجموعه اشاره می کند بر گردانده می شود.

از erase برای پاک کردن یک عنصر یا یک بازه استفاده می شود،اگر هدف پاک کردن تنها یک عنصر باشد از دو شیوه می توان از این متد استفاده کرد:

1) با استفاده از کلید
2) با استفاده از iterator

ولی اگر هدف حذف یک بازه باشد باید حتماً از iterator استفاده کرد.

اگر از iterator برای حذف یک عنصر استفاده می کنید حتماً مطمئن شوید که iterator به دست آمده معتبر است ،در غیر این صورت استفاده از erase منجر به نتایج نا خواسته ای می شود.

```
#include <iostream>
#include <map>
#include <string>
using namespace std;


map<string,int> mp;

void print(map<string,int> m){
	cout<<"\n---------------\n";
	if(m.empty()){
		cout<<"Empty\n";
	}else{
		map<string,int>::iterator start=m.begin();
		while(start!=m.end()){
			pair<string,int> p=(pair<string,int>)*start;
			cout<<p.first<<"\t"<<p.second<<"\n";
			start++;
		}
	}
	cout<<"\n---------------\n";
}
```

```
int main()
{
    int hold;
    string key[]={"ali","reza","hasan","ali","taghi","naghi"};
    int  value[]={ 12  ,10    ,12     ,13   ,11     ,9};
    for(int i=0;i<6;i++)
            mp.insert(pair<string,int>(key[i],value[i]));
    print(mp);
    cout<<mp.count("ali")<<"\n";
    cout<<mp.count("safeallah")<<"\n";
    mp.erase("ali");
    mp.erase("safeallah");
    map<string,int>::iterator f_it=mp.find("hosein");
    if(f_it!=mp.end())//check the iterator !
         mp.erase(f_it);
    print(mp);
    mp.erase(++mp.begin(),--mp.end());
    print(mp);
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
---------------
ali     12
hasan   12
naghi   9
reza    10
taghi   11

---------------
1
0

---------------
hasan   12
naghi   9
reza    10
taghi   11

---------------

---------------
hasan   12
taghi   11

---------------
```

مثال 4:lower_bound،upper_bound

```
#include <iostream>
#include <map>
#include <string>
using namespace std;
typedef map<string,string>::iterator my_it;
map<string,string> mp;
```

```
void print(map<string,string> m){
	cout<<"\n---------------\n";
	if(m.empty()){
		cout<<"Empty\n";
	}else{
		map<string,string>::iterator start=m.begin();
		while(start!=m.end()){
			pair<string,string> p=(pair<string,string>)*start;
			cout<<p.first<<"\t"<<p.second<<"\n";
			start++;
		}
	}
	cout<<"\n---------------\n";
}

int main()
{
    int hold;
    string key[]={"ali","reza","hasan","ahad","taghi","naghi"};
    string value[]={"0012","0010","0102","0103","0011","0009"};
    for(int i=0;i<6;i++)
        mp.insert(pair<string,string>(key[i],value[i]));
    print(mp);
    pair<string,string> temp;
    my_it m_i;
    //lower bound:
    cout<<"lower_bound:\n";
    m_i=mp.lower_bound("hasan");
    temp=(pair<string,string>)*m_i;
    cout<<temp.first<<" "<<temp.second<<"\n";
    m_i=mp.lower_bound("jamshid");
    temp=(pair<string,string>)*m_i;
    cout<<temp.first<<" "<<temp.second<<"\n";
    //upper_bound:
    cout<<"upper_bound:\n";
    m_i=mp.upper_bound("hasan");
    temp=(pair<string,string>)*m_i;
    cout<<temp.first<<" "<<temp.second<<"\n";
    m_i=mp.lower_bound("jamshid");
    temp=(pair<string,string>)*m_i;
    cout<<temp.first<<" "<<temp.second<<"\n";

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
---------------
ahad    0103
ali     0012
hasan   0102
naghi   0009
reza    0010
taghi   0011

---------------
```

```
lower_bound:
hasan 0102
naghi 0009
upper_bound:
naghi 0009
naghi 0009
```

مثال 5:

اگر عناصر نگاشت ساختار داده هایی باشند که خودمان تعریف کردیم آنگاه باید معیار مقایسه را صریحاً مشخص کنیم.(معیار مقایسه یک class یا struct است که () را overload کرده است)

توجه کنید که در این حالت نیز نگاشت کلید تکراری را نگه داری نمی کند.(به (reza,reza_e توجه کنید)

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

typedef struct student{
        string name;
        string lastName;
};

struct student_compare{
       bool operator() (student s1,student s2){
            if(s1.name.compare(s2.name)<0){
                return true;
            }else if(s1.name.compare(s2.name)==0){
                  if(s1.lastName.compare(s2.lastName)<0){
                        return true;
                  }else{
                        return false;
                  }
            }else{
                  return false;
            }
       }
};
typedef map<student,int,student_compare> my_map;
my_map sts;


void print(my_map m){
     my_map::iterator start=m.begin();
     while(start!=m.end()){
           pair<student,int> temp=(pair<student,int>)*start;
           cout<<temp.first.name<<"\t"<<temp.first.lastName
                                 <<"\t"<<temp.second<<"\n";
           start++;
     }
}
```

```
int main()
{
    int hold;
    string name[]={"ali","reza","hasan","ahad",
                                "taghi","naghi","reza","reza"};
    string family[]={"ali_e","reza_e","hasan_e","ahad_e",
                                "taghi_e","naghi_e","reza_e","akbar_e"};
    int     num[]=   {18    ,15      ,16       , 17
                                       ,18       ,16       ,  19    , 14};
    for(int i=0;i<8;i++){
            student temp;
            temp.name=name[i];temp.lastName=family[i];
            sts.insert(pair<student,int>(temp,num[i]));
    }
    print(sts);
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
ahad    ahad_e  17
ali     ali_e   18
hasan   hasan_e 16
naghi   naghi_e 16
reza    akbar_e 14
reza    reza_e  15
taghi   taghi_e 18
```

نگاشت چندگانه (Multimap)

نگاشت چندگانه همانند نگاشت است با این تفاوت که جفت های با کلید تکراری را نیز نگه داری می کند.

برای استفاده از این ساختمان داده باید ابتدا کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم.

```
#include <map>
```

مثال 1:تفاوت map و multimap(با مثال های قبلی مقایسه کنید)

```
#include <iostream>
#include <map>
#include <string>
using namespace std;
multimap<string,int> mp;
void print(multimap<string,int> m){
     cout<<"\n---------------\n";
     if(m.empty()){
           cout<<"Empty\n";
     }else{
           map<string,int>::iterator start=m.begin();
           while(start!=m.end()){
                 pair<string,int> p=(pair<string,int>)*start;
                 cout<<p.first<<"\t"<<p.second<<"\n";
                 start++;
           }
     }
     cout<<"\n---------------\n";
}
```

```
int main()
{
    int hold;
    string key[]={"ali","reza","hasan","ali","ali","taghi","naghi"};
    int  value[]={ 12  ,10    ,12     ,13   ,12,11     ,9};
    for(int i=0;i<7;i++)
            mp.insert(pair<string,int>(key[i],value[i]));
    print(mp);
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
---------------
ali     12
ali     13
ali     12
hasan   12
naghi   9
reza    10
taghi   11

---------------
```

مثال 2:lower_bound،upper_bound،equal_range

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

multimap<string,int> mp;

void print(multimap<string,int> m){
     cout<<"\n---------------\n";
     if(m.empty()){
           cout<<"Empty\n";
     }else{
           map<string,int>::iterator start=m.begin();
           while(start!=m.end()){
                 pair<string,int> p=(pair<string,int>)*start;
                 cout<<p.first<<"\t"<<p.second<<"\n";
                 start++;
           }
     }
     cout<<"\n---------------\n";
}
```

```
int main()
{
    int hold;
    string key[]={"ali","reza","hasan","ali","ali",
                                          "taghi","reza","naghi","reza"};
    int  value[]={ 12  ,10    ,12     ,13   ,12   ,
                                           11   ,8    ,  12   , 9};
    for(int i=0;i<9;i++)
            mp.insert(pair<string,int>(key[i],value[i]));
    print(mp);
    multimap<string,int>::iterator lb=mp.lower_bound("ali");
    multimap<string,int>::iterator ub=mp.upper_bound("ali");
    pair<string,int> p=(pair<string,int>)*lb;
    cout<<"lower_bound:"<<p.first<<" "<<p.second<<"\n";
    while(lb!=ub){
            p=(pair<string,int>)*lb;
            cout<<p.first<<" "<<p.second<<"\n";
            lb++;
    }
    if(ub!=mp.end()){
           p=(pair<string,int>)*ub;
           cout<<"upper_bound:"<<p.first<<" "<<p.second<<"\n";
    }
    pair<multimap<string,int>::iterator,
                  multimap<string,int>::iterator>  temp;
    temp=mp.equal_range("reza");
    while(temp.first!=temp.second){
          p=(pair<string,int>)*temp.first;
          cout<<p.first<<" "<<p.second<<"\n";
          temp.first++;
    }
    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
---------------
ali     12
ali     13
ali     12
hasan   12
naghi   12
reza    10
reza    8
reza    9
taghi   11

---------------
lower_bound:ali 12
ali 12
ali 13
ali 12
upper_bound:hasan 12
reza 10
reza 8
reza 9
```

ساختمان داده هایی که براساس الگوریتم های درهم سازی کار می کنند

ساختمان داده هایی که بر اساس درهم سازی یا همان hash کار می کنند هنوز (تا این زمان که بنده در حال تایپ این جملات هستم) به صورت استاندارد در نیامده اند و جزئی اصلی از کتابخانه های استاندارد c++ نیستند برای همین در اینجا نیز به تفصیل به آن ها نمی پردازیم.با همه این ها هنوز می توانید از چنین ساختمان داده هایی استفاده کنید ولی باید برای جزئیات دقیق به مستندات کامپایلری که از آن استفاده می کنید مراجعه می کنید.در ضمن یک توصیه دوستانه از طرف نویسنده :اگر می خواهید بیشتر از بیش از c++ لذت ببرید به سمت c++0x کوچ کنید این نسخه از c++ ویژگی ها بسیار جذاب و دلربایی دارد که هر برنامه نویسی را از خود بی خود می کند.توصیه دوم استفاده از سایت ideone.com است ، حتماً به این سایت سر بزنید و ویژگی ها هر کامپایلری را که دوست دارید در آن آزمایش کنید.

و اما معرفی چندی از این نوع ساختمان داده ها:

اول برای دوست داران DEV:

:hash_set

برای استفاده از این ساختمان داده باید هم کتابخانه ای که در بردارنده آن است را به برنامه اضافه کنیم و هم از gnu_cxx__ به عنوان یکی از namespace هایمان استفاده کنیم.

```
#include <ext/hash_set>
using namespace   __gnu_cxx;
```

مثال 1:استفاده از hash_set برای نگه داری مجموعه ای از اعداد.

```
#include <ext/hash_set>
#include <iostream>
using namespace std;
using namespace __gnu_cxx;

hash_set<int> hs;
int main(){
    int hold;
    hs.insert(12);
    hs.insert(14);
    hs.insert(16);

    hash_set<int>::iterator it=hs.find(12);
    if(it==hs.end()){
                  cout<<"NOOOOOOOOO!\n";
    }else{
          cout<<"YESSSSSSSSSSSS\n";
    }

    it=hs.find(15);
    if(it==hs.end()){
                  cout<<"NOOOOOOOOO!\n";
    }else{
          cout<<"YESSSSSSSSSSSS\n";
    }

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
YESSSSSSSSSSSS
NOOOOOOOOO!
```

مثال 2:استفاده از hash_set برای مجموعه ای از رشته ها

```
#include <ext/hash_set>
#include <iostream>
using namespace std;
using namespace __gnu_cxx;

hash_set<char*> hs;

bool contain(char* name){
    hash_set<char*>::iterator it=hs.find(name);

    if(it==hs.end())
          return false;

    return true;
}

int main(){
    int hold;

    hs.insert("ahad");
    hs.insert("alireza");
    hs.insert("jamshid");
    hs.insert("safe");
    hs.insert("mansour");
    hs.insert("hosein");
    hs.insert("akbar");
    hs.insert("navand");

    char* names[]={"nosrat","changiz","mohammad","payam",
                   "navand","akbar","safeallah","hashem"};

    for(int i=0;i<8;i++){
            cout<<names[i]<<"    \t: "<<contain(names[i])<<"\n";
    }

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

خروجی برنامه فقط برای navand و akbar یک می شود.

```
nosrat          : 0
changiz         : 0
mohammad        : 0
payam           : 0
navand          : 1
akbar           : 1
safeallah       : 0
hashem          : 0
```

:hash_map

از hash_map می توان برای نگه داری جفت هایی از کلید و مقدار نگه داری کرد.برای استفاده از hash_map باید کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم:

```
#include <ext/hash_map>
```

مثال1:استفاده از hash_map برای نگه داری نام دانشجو ها و نمره هایشان

```
#include <ext/hash_map>
#include <iostream>
using namespace std;
using namespace __gnu_cxx;

hash_map<char*,double> hm;

int main(){
    int hold;

    hm.insert(pair<char*,double>("ali",20));
    hm.insert(pair<char*,double>("jamshid",8));
    hm.insert(pair<char*,double>("taghi",12));

    cout<<"jamshid\t"<<hm["jamshid"]<<"\n";
    cout<<"ali\t"<<hm["ali"]<<"\n";
    cout<<"taghi\t"<<hm["taghi"]<<"\n";

    cin>>hold;
    return 0;
}
```

خروجی برنامه:

```
jamshid 8
ali     20
taghi   12
```

به سمت c++0x:

هر چند c++0x بحثی است که جداگانه باید به آن پرداخته شود ولی در اینجا به چند کتابخانه از این نسخه که از درهم سازی استفاده می کنند اشاره می کنیم (کد های این قسمت در dev قابل اجرا نیستند مگر اینکه کامپایلر مروبطه را دانلود کنید و مسیر آن را در dev مشخص سازید).

:unordered_set

از unordered_set مانند set برای نگه داری یک مجموعه مقادیر استفاده می شود ، منتهی در اینجا چون از درهم سازی برای مشخص سازی مکان ذخیره سازی مقادیر استفاده می شود ترتیب کلید ها معنی خود را از دست می دهد.

برای استفاده از این ساختمان باید ابتدا کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم:

```
#include <unordered_set>
```

متد های مهم این ساختمان داده عبارتند از:

begin: یک iterator به ابتدای مجموعه را بر می گرداند.
end: یک iterator به انتهای مجموعه را بر می گرداند.
empty: مشخص می کند که آیا مجموعه خالی است یا نه؟
size: اندازه مجموعه را مشخص می کند.
clear: مجموعه را پاک می کند.
insert: برای درج یک عنصر در مجموعه از این متد استفاده می کنیم.
erase: برای پاک کردن یک عنصر به کار می رود.
count: تعداد تکرار های یک عنصر را می شمارد.
find: یک iterator به عنصر مورد جست و جو بر می گرداند اگر عنصر در مجموعه وجود نداشته باشد end را بر می گرداند.

مثال 1: آشنایی با unordered_set

این مثال نشان می دهد چگونه می توانیم از unordered_set برای نگه داری مجموعه از اعداد استفاده کنیم.همچنین این مثال نشان می دهد unordered_set مقادیر تکراری را نگه داری نمی کند.

```
#include <unordered_set>
#include <iostream>

using namespace std;

unordered_set<int> uset;

int main(){
        int hold;
        uset.insert(2);
        uset.insert(4);
        uset.insert(5);
        uset.insert(3);
        uset.insert(2);
        uset.insert(3);

        cout<<"size of set :"<<uset.size()<<"\n";
        cout<<"# of 3 :"<<uset.count(3)<<"\n";
        cout<<"# of 2 :"<<uset.count(2)<<"\n";
        cout<<"# of 5 :"<<uset.count(5)<<"\n";
        cout<<"# of 6 :"<<uset.count(6)<<"\n";

        cin>>hold;
        return 0;
}
```

خروجی برنامه:

```
size of set :4
# of 3 :1
# of 2 :1
# of 5 :1
# of 6 :0
```

همانطور که از خروجی برنامه پیداست unordered_set مقادیر تکراری را نگه داری نمی کند و مقادیر 2 و 5 هر کدام فقط یک بار در مجموعه قرار گرفتند در نتیجه در مجموعه تنها 4 کلید متفاوت 2,4,5,3 وجود دارند.

همچنین مشاهده می کنید که از متد count می توانیم برای اینکه مشخص کنیم کلیدی در مجموعه وجود دارد یا نه استفاده کنیم در مثال فوق 6 در مجموعه درج نشده بود لذا count(6) برابر با صفر است.

مثال2: استفاده از unordered_set برای نگه داری مجموعه از رشته ها

در این مثال از unordered_set برای نگه داری مقادیر رشته ای استفاده می کنیم ، همچنین این مثال نشان می دهد چگونه می توانیم از متد های begin و end برای پیمایش کل مجموعه استفاده کنیم.کاربرد متد erase و find نیز نشان داده شده است.

```
#include <unordered_set>
#include <iostream>

using namespace std;

unordered_set<string> uset;

void print(){
	unordered_set<string>::iterator it;
	it=uset.begin();
	while(it!=uset.end()){
		cout<<*it<<"\n";
		it++;
	}
}
int main(){
	int hold;
	uset.insert("ali");
	uset.insert("reza");
	uset.insert("hasan");
	uset.insert("ahamd");
	uset.insert("ali");

	cout<<"size of set :"<<uset.size()<<"\n";
	print();
	cout<<"---------\n";
	if(uset.find("ali")!=uset.end()){
		cout<<"YES\n";
		uset.erase("ali");
		print();
	}

	cin>>hold;
	return 0;
}
```

خروجی برنامه:

```
size of set :4
hasan
ahamd
reza
ali
---------
YES
hasan
ahamd
reza
```

همانطور که مشاهده می کنید چون در ابتدا ali در مجموعه قرار دارد لذا متد find یک iterator بر می گرداند که با end برابر نیست لذا کد داخل if اجرا می شود و ali از مجموعه حذف می شود.
متد print نیز نیاز به توضیح خاصی ندارد مشابه آن را در این متن بسیار نوشته ایم.

مثال3: دسترسی به توابع hash

اگر می خواهید از توابع hash ای که در unordered_set از آن استفاده می شود با خبر شوید می توانید به صورت زیر عمل کنید:

```
#include <unordered_set>
#include <iostream>

using namespace std;

int main(){
        int hold;
        unordered_set<int>::hasher hfunc1;
        unordered_set<string>::hasher hfunc2;

        cout<<12345<<"\t: "<<hfunc1(12345)<<endl;
        cout<<76788<<"\t: "<<hfunc1(76788)<<endl;

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

        cout<<"ali"<<"\t: "<<hfunc2("ali")<<endl;
        cout<<"reza"<<"\t: "<<hfunc2("reza")<<endl;



        cin>>hold;
        return 0;
}
```

خروجی برنامه:

```
12345 : 12345
76788 : 76788
-------
ali   : 270878475
reza  : 30321389
```

توجه کنید که از hasher های متناسب با نوع داده ای استفاده کردیم.

unordered_multiset:

این ساختمان داده مشابه unordered_set است و در همان کتابخانه قرار دارد تنها تفاوت در این است که کلید های تکراری نیز نگه داری می شوند.

مثال 1:

```
#include <unordered_set>
#include <iostream>

using namespace std;

unordered_multiset<string> umset;

void print(){
	unordered_multiset<string>::iterator it;
	it=umset.begin();
	while(it!=umset.end()){
		cout<<*it<<" "<<umset.count(*it)<<"\n";
		it++;
	}
}
int main(){
	int hold;
	umset.insert("ali");
	umset.insert("reza");
	umset.insert("hasan");
	umset.insert("ahamd");
	umset.insert("ali");
	umset.insert("reza");
	umset.insert("ali");


	cout<<"size of set :"<<umset.size()<<"\n";
	print();

	cin>>hold;
	return 0;
}
```

خروجی برنامه:

```
size of set :7
hasan 1
ahamd 1
reza 2
reza 2
ali 3
ali 3
ali 3
```

همانطور که مشاهده می کنید unordered_multiset مقادیر تکراری را نیز نگه داری می کند .همچنین توجه کنید که متد print را طوری تغییر دادیم تا تعداد تکرار های یک کلید را نیز چاپ کند.

unordered_map:

از unordered_map مانند map برای نگه داری زوج های کلید و مقدار استفاده می کنیم ، چون این ساختمان داده از hash کردن استفاده می کند ترتیب کلید ها بی اهمیت می شود.

برای استفاده از این ساختمان داده باید کتابخانه ای که آن را در بر دارد به برنامه اضافه کنیم:

```
#include <unordered_map>
```

متد های این ساختمان از لحاظ نام مشابه متدهای unordered_set هستند لذا از ذکر نام و توضیح آنها خود داری می کنیم و در عوض با مثال با این متد ها آشنا می شویم.

مثال1: آشنایی با unordered_map(استفاده از unordered_map برای نگه داری زوج های نام و نمره)

در این مثال نحوه استفاده از find و count نیز نشان داده شده است.

```
#include <unordered_map>
#include <iostream>

using namespace std;

unordered_map<string,double> umap;
int main(){
	int hold;

	umap.insert(pair<string,double>("jamshid",8.2));
	umap.insert(pair<string,double>("safeallah",100));
	umap.insert(pair<string,double>("naghi",45.6));

	cout<<"safeallah"<<"\t"<<umap["safeallah"]<<"\n";
	cout<<"jamshid  "<<"\t"<<umap["jamshid"]<<"\n";

	if(umap.find("taghi")!=umap.end()){
		cout<<"YES\n";
	}else{
		cout<<"NO\n";
	}

	cout<<umap.count("bazi ha");

	cin>>hold;
	return 0;
}
```

خروجی برنامه:

```
safeallah	100
jamshid	8.2
NO
0
```

مثال2: unordered_map زوج های با کلید تکراری را نگه داری نمی کند.

```
#include <unordered_map>
#include <iostream>

using namespace std;

unordered_map<string,double> umap;
void print(){
        unordered_map<string,double>::iterator it;
        it=umap.begin();
        while(it!=umap.end()){
             cout<<it->first<<"\t"<<it->second<<"\n";
             it++;
        }
}
int main(){
        int hold;

        umap.insert(pair<string,double>("ali",12.2));
        umap.insert(pair<string,double>("reza",23.5));
        umap.insert(pair<string,double>("hasan",5.7));
        umap.insert(pair<string,double>("hasan",8.2));
        umap.insert(pair<string,double>("ali",20));

        print();

        cin>>hold;
        return 0;
}
```

خروجی برنامه:

```
hasan 5.7
reza  23.5
ali   12.2
```

همانطور که از خروجی پیداست تنها مقداری که برای اولین بار برای hasan در مجموعه درج کردیم نگه داری می شود و مقدار 8.2 برای دفعه دوم به عنوان مقدار برای hasan درج نمی شود.درج مقدار 20 برای ali نیز عملی نا موفق بوده است و برای ali همان مقدار 12.2 که در ابتدا درج کردیم ثبت شده است.

مثال3:استفاده از unordered_map برای نگه داری ساختمان داده هایی که خودمان تعریف کردیم

اگر بخواهیم از unordered_map برای نگه داری ساختار های اختصاصی ای که خودمان تعریف می کنیم استفاده کنیم باید دو کار اضافه انجام دهیم:

1) مشخص کنیم که چگونه این ساختار اختصاصی hash شود.
2) مشخص کنیم که شرط برابری دو شی از این نوع ساختاری که تعریف می کنیم چیست.

برای مشخص کردن دو شرط فوق باید برای هرکدام یک struct تعریف کنیم که در ادامه نحوه انجام این کار را خواهیم دید.

در این مثال می خواهیم یک ساختار اختصاصی به نام mypoint ایجاد کنیم که ساختاری برای نگه داری اطلاعات نقطه است این ساختار دارای دو عضو x و y است که مختصات x و y یک نقطه را مشخص می کند.از نقاط به عنوان کلید استفاده می کنیم ، سپس به ازای هر کلید مقداری رشته ای به صورت good برای نقطه های خوب و bad برای نقطه های بد اختصاص می دهیم.

ساختار mypoint را به صورت زیر تعریف می کنیم:

```
struct mypoint {
        int x;
        int y;
};
```

برای ایجاد یک تابع اختصاصی hash یک struct به صورت زیر تعریف می کنیم ، بعداً از این struct به عنوان پارامتر unordered_map استفاده خواهیم کرد.

```
struct point_hash {
      size_t operator()(mypoint const& p) const
      {
          unordered_map<int,int>::hasher hfunc;
          size_t s=hfunc(p.x)+hfunc(p.y);
          return s;
      }
};
```

قبلاً با نحوه استفاده از hasher آشنا شدیم ، در اینجا مشاهده می کنید که هر کدام از عضو های mypoint را جداگانه hash می کنیم و سپس مقادیر آنها را با هم جمع می کنیم، تعیین نحوه hash کردن چیزی است که به برنامه نویس بستگی دارد و باید در نوشتن آن دقت شود تا احتمال collision پایین باشد.

داشتن یک روش برای تعین شرط تساوی دو شی از داشتن تابع hash نیز مهم تر است ، زیرا ممکن است چند شی دارای مقدار hash یکسانی شوند (collision اتفاق بیافتد) ولی شرط تساوی باید وجود داشته باشد تا اشیا به درستی با هم مقایسه شوند.

برای انجام این کار یک struct به صورت زیر تعریف می کنیم و بعداً از آن به عنوان یکی از پارامتر های unordered_map استفاده می کنیم.

```
struct point_equal
{
  bool operator() ( mypoint const& p1,mypoint const& p2) const
  {
    return p1.x == p2.x && p1.y == p2.y;
  }
};
```

در تعریف unordered_map باید تمام این چیز هایی را که گفتیم به عنوان پارامتر استفاده کنیم:

```
unordered_map<mypoint,string,point_hash,point_equal> pts;
```

با توجه به این توضیحات کد نهایی برنامه به صورت زیر خواهد بود:

```
#include <unordered_map>
#include <iostream>
using namespace std;

struct mypoint {
  int x;
  int y;
};

struct point_hash {
      size_t operator()(mypoint const& p) const
      {
            unordered_map<int,int>::hasher hfunc;
            size_t s=hfunc(p.x)+hfunc(p.y);
      return s;
      }
};

struct point_equal
{
  bool operator() ( mypoint const& p1,mypoint const& p2) const
  {
    return p1.x == p2.x && p1.y == p2.y;
  }
};

unordered_map<mypoint,string,point_hash,point_equal> pts;

void print(){
        unordered_map<mypoint,string>::iterator it;
        it=pts.begin();
        while(it!=pts.end()){
                mypoint p=it->first;

                cout<<"[ "<<p.x<<","<<p.y<<"] : "<<it->second<<"\n";

                it++;
        }
        cout<<"end print\n";
}
```

نکته جالبی که در متد print وجود دارد این است دردسر cast کردن iterator ها در آن وجود ندارد!

```
int main(){
        struct mypoint p1,p2,p3;

        p1.x=2;
        p1.y=2;

        p2.x=3;
        p2.y=4;

        p3.x=2;
        p3.y=2;

        pts.insert(pair<mypoint,string>(p1,"good"));
        pts.insert(pair<mypoint,string>(p2,"bad"));
        pts.insert(pair<mypoint,string>(p3,"bad"));

        mypoint q;
        q.x=2;
        q.y=2;
        print();
        cout<<"size of hash map :"<<pts.size()<<"\n";
        unordered_map<mypoint,string>::iterator it;
        it=pts.find(q);
        if(it==pts.end()){
                cout<<"NO!\n";
        }else{
                cout<<it->second<<"\n";
                //OR:
                cout<<pts[q];
        }

return 0;

}
```

در برنامه فوق مشاهده می کنید که سه نقطه تعریف کردیم و در مجموعه مورد نظر قرار دادیم که نقطه سوم در حقیقت تکراری است.

خروجی برنامه:

```
[ 2,2] : good
[ 3,4] : bad
end print
size of hash map :2
good
good
```

همانطور که مشاهده می کنید ابتدا متد print اجرا می شود و نقاط مجموعه به همراه صفتی که به آنها اختصاص دادیم چاپ می شوند ،می بینیم که نقطه 2,2 با صفت good در مجموعه ذخیره است پس نتیجه می گیریم که نقطه p3 تکراری بوده و در مجموعه درج نشده است(به کد main توجه کنید).

نحوه استفاده از متد find و iterator ای که از آن به دست می آید نیز در قسمت انتهایی کد نشان داده شده است.

:unordered_multimap

این ساختمان داده مانند unordered_map است با این تفاوت که کلید های تکراری را نیز نگه داری می کند.این ساختمان داده نیز در کتابخانه unordered_map قرار دارد.

مثال 1:

```
#include <unordered_map>
#include <iostream>

using namespace std;

unordered_multimap<string,double> ummap;
void print(){
        unordered_multimap<string,double>::iterator it;
        it=ummap.begin();
        while(it!=ummap.end()){
             cout<<it->first<<"\t"<<it->second<<"\n";
             it++;
        }
}
int main(){
        int hold;

        ummap.insert(pair<string,double>("ali",12.2));
        ummap.insert(pair<string,double>("reza",23.5));
        ummap.insert(pair<string,double>("hasan",5.7));
        ummap.insert(pair<string,double>("hasan",8.2));
        ummap.insert(pair<string,double>("ali",20));

        print();

        cin>>hold;
        return 0;
}
```

خروجی برنامه:

```
hasan 5.7
hasan 8.2
reza  23.5
ali   12.2
ali   20
```

همانطور که در خروجی مشاهده می کند مقادیر تکراری نیز در unordered_multimap نگه داری می شوند.(با مثالی که برای unordered_map آورده بودیم مقایسه کنید).

<complex>

این کتابخانه حاوی کلاسی با همین نام است که برای نمایش اعداد مختلط از آن استفاده می شود.

یک عدد مختلط شامل دو قسمت حقیقی و مجازی است و به صورت زیر نشان داده می شود:

z=a+bi

که در عبارت بالا $i = \sqrt{-1}$ است.

از عبارت فوق می فهمیم که هر عدد مختلط را می توان با دو عدد a و b مشخص کرد.a و b می توانند حقیقی یا صحیح باشند.

مثال1:نمایش اعداد مختلط با بخش های صحیح

```
#include <iostream>
#include <complex>
using namespace std;

int main(){
        int hold;

        int a=4;
        int b=5;
        //z=a+bi
        complex<int> z(a,b);
        cout<<z;

        cin>>hold;
        return 0;
}
```

خروجی برنامه:

```
(4,5)
```

مثال2:نمایش اعداد مختلط با بخش های حقیقی (استفاده از اعداد ممیز شناور)

```
#include <iostream>
#include <complex>
using namespace std;

int main(){
        int hold;

        double a=5.3;
        double b=2.7;
        //z=a+bi
        complex<double> z(a,b);
        cout<<z;

        cin>>hold;
        return 0;
}
```

خروجی برنامه:

```
(5.3,2.7)
```

مثال3:جمع اعداد مختلط

اگر z1 و z2 دو عدد مختلط باشند جمع آن ها به صورت زیر خواهد بود:

z1=a1+b1i

z2=a2+b2i

z1+z2=(a1+a2)+(b1+b2)i

```
#include <iostream>
#include <complex>
using namespace std;

int main(){
        int hold;
        complex<int> z1(5,2);//z1=5+2i
        complex<int> z2(3,10);//z2=3+10i

        complex<int> z3=z1+z2;
        cout<<z3;

        cin>>hold;
        return 0;
}
```

خروجی برنامه:

```
(8,12)
```

مثال4:تفریق اعداد مختلط

اگر z1 و z2 دو عدد مختلط باشند تفریق آن ها به صورت زیر خواهد بود:

z1=a1+b1i

z2=a2+b2i

z1-z2=(a1-a2)+(b1-b2)i

```
#include <iostream>
#include <complex>
using namespace std;

int main(){
        int hold;
        complex<int> z1(5,2);//z1=5+2i
        complex<int> z2(3,10);//z2=3+10i

        complex<int> z3=z1-z2;
        cout<<z3;

        cin>>hold;
        return 0;
}
```

خروجی برنامه:

```
(2,-8)
```

مثال5:ضرب اعداد مختلط

اگر z1 و z2 دو عدد مختلط باشند ضرب آن ها به صورت زیر خواهد بود:

$$z1=a1+b1i$$

$$z2=a2+b2i$$

$$z1*z2=(a1*a2 - b1*b2)+(a1*b2 + b1*a2)i$$

```
#include <iostream>
#include <complex>
using namespace std;

int main(){
        int hold;
        complex<int> z1(5,2);//z1=5+2i
        complex<int> z2(3,10);//z2=3+10i

        complex<int> z3=z1*z2;
        //z3=(3*5-2*10)+(5*10+2*3)i
        cout<<z3;

        cin>>hold;
        return 0;
}
```

خروجی برنامه:

```
(-5,56)
```

مثال6:تقسیم اعداد مختلط

اگر z1 و z2 دو عدد مختلط باشند تقسیم آن ها به صورت زیر خواهد بود:

z1=a+bi

z2=c+di

$$\frac{z1}{z2} = \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}i$$

```
#include <iostream>
#include <complex>
using namespace std;

int main(){
        int hold;
        complex<double> z1(5,2);//z1=5+2i
        complex<double> z2(3,10);//z2=3+10i

        complex<double> z3=z1/z2;
        cout<<z3;
        cout<<endl;
        cout<<(3.0*5.0+2.0*10.0)/(3.0*3.0+10.0*10.0);
        cout<<endl;
        cout<<(2.0*3.0-5.0*10.0)/(3.0*3.0+10.0*10.0);
        cin>>hold;
        return 0;
}
```

خروجی برنامه:

```
(0.321101,-0.40367)
0.321101
-0.40367
```

نکته پایانی این بخش: از توابع ریاضی هم می توانید برای کار با اعداد مختلط استفاده کنید.