BASIC STL  
#include <iostream>

#include<bits/stdc++.h>

using namespace std;

int main()

{

   int a[]={10,15,6,20};

   int sum=0;

   sort(a,a+4);    //sort the array in increasing order

   for(int i=0;i<4;i++)

   {

       cout<<a[i]<<" ";

   }

   cout<<endl;

   if(binary\_search(a,a+4,6))     //binary search  and returns the boolean value

   {

       cout<<"present";

   }else

   {

       cout<<"not present";

   }

   cout<<endl;

   reverse(a,a+4);           // used to reverse the array

   for(int i=0;i<4;i++)

   {

       cout<<a[i]<<" ";

   }

   cout<<endl;

   cout<<\*max\_element(a,a+4);    //used to find the maximum element in the array

   cout<<endl;

   cout<<\*min\_element(a,a+4);    //used to find the minimum element in the array;

    cout<<endl;

    cout<<accumulate(a,a+4,sum);  // used to find the sum of element between starting and ending point and sum is the initial value

    cout<<endl;

    cout<<count(a,a+4,6);

        return 0;

}

ITRATOR

#include <iostream>

#include<bits/stdc++.h>

using namespace std;

int

main ()

{

  vector < int >v = { 10, 15, 6, 20, 56 };

  auto i = v.begin ();      // i is the iterator which point to begining of vector

  cout << \*i << endl;

  i = next (i);         // returns the iterator next to i

  cout << \*i << endl;

  i = next (i, 2);      //return the iterator 2 position ahead

  cout << \*i << endl;

  i = prev (i);         //return the previous iterator

  cout << \*i << endl;

  i = prev (i, 2);      // return the 2 position back iterator

  cout << \*i << endl;

  return 0;

}

**Priority Queue**

// priority queue is same as binary heap

//Binary heap is a binary tree with the following properties

//-->>it is complete binary tree and this property of Binary heap makes it suitable to be stored in a array.

//            -->>Max Heap: the Key at the root must be maximum among all keys present in binary heap.

//            |

//Binary tree--

//            |

//             -->>Min Heap: the key at root must be minimum among all keys present in binary heap

// priority queue is used to set priority among element

#include<iostream>

#include<queue>// header file is queue itself

using namespace std;

int main()

{

    priority\_queue <int> pq;

    pq.push(10);        //------

    pq.push(2);         //      |

    pq.push(20);        //      |-->> by default this is "MAX" heap this can be proved by printing top element which will be max among all that is 30

    pq.push(15);        //      |

    pq.push(30);        //------

    cout<<pq.top()<<endl;

    //MIN heap

    priority\_queue <int, vector<int>, greater<int> > pq1,pq2;// declaring min heap

    pq1.push(10);

    pq1.push(2);

    pq1.push(20);

    pq1.push(15);

    pq1.push(30);

    cout<<pq1.top()<<endl;

    cout<<pq.size()<<endl; // size of heap

    if(pq.empty())

    {

        cout<<"pq is empty\n";

    }else

    {

        cout<<"pq is not empty\n";

    }

     pq1.swap(pq2);

     cout<<pq.top()<<endl;

    cout<<pq1.top()<<endl;

}

**PAIR**

//-->>pair is used to combine two values that may be different in type.

//-->>pair provided a way to store two heterogenous objects as a single unit.

//-->>it is basically used if we want to store tuples.

//it is defined in <utility> header file

#include<iostream>

#include<queue>

#include<utility>

using namespace std;

int main()

{

    pair <int, int> p;// pair can be made of any type both elements can be of same data type or different or like <int,int>,<int,char>,<char,double>,<int,pair<int,int> >

    p=make\_pair(5,3);// make pair is used to insert value it is a generic function

    cout<<p.first<<endl;// pairname.first is used to access first element

    cout<<p.second<<endl;// pairname.second is used to access the second element

    // we can make prorty queue of pair

    priority\_queue<pair<int,int> > pq;

    pq.push(make\_pair(30,20));

    pq.push(make\_pair(20,30));

    pq.push(make\_pair(10,30));

    pq.push(make\_pair(5,30));

  //we know that priority queue behaves as binary heap and by default it is a max heap so our

    cout<<pq.top().first<<endl;

    pq.pop();

    cout<<pq.top().first<<endl;

     pq.pop();

    cout<<pq.top().first<<endl;

}

**SET**

/\* SET:->> set is a type of associative containers in which each element has to be UNIQUE

    ->> the values are stored in a specific order

syntax: set<datatype> set\_name;

       -->> ordered set: the set stores the element in sorted order. and is implemented using balanced Binary search tree. time complexity of insertion, search, and deletion on set is O(logn) where n is the number of elements in sets

       |

 SET-->>

       |

       -->> unordered set: the set stores the element in unsorted order. And is implemented using hashing. time complexity of insertion, search, and deletion on unordered set is O(1)

MULTISET:->> multiset is similar to the set, with the exception can have the duplicates values. MULTISET is of 2 types ordered and unordered

\*/

#include<iostream>

#include<set>

using namespace std;

int main()

{

    set <int> s;

    //insert->> To insert new element in a set

    s.insert(10);

    s.insert(20);

    s.insert(40);

    s.insert(45);

    s.insert(12);

    s.insert(19);

    for(auto it=s.begin(); it!=s.end();it++)

    {

        cout<<\*it<<" ";

    }

    cout<<endl;

    //find()->>searches element in the set: it returns the iterator of that  searched number if it is found otherwise it returns the end element iterator

    if(s.find(10) != s.end())

    {

        cout<<"10 is found in a set"<<endl;

    }else

    {

        cout<<"10 not found in a set"<<endl;

    }

    //erase()-->> deletes an element or set of elements from the set

    s.erase(10);  // this can also be done by giving iterator as an argument in the erase function EXAMPLE:- s.erase(s.find(10)); ->this returns the iterator/pointer to element 10

     for(auto it=s.begin(); it!=s.end();it++)

    {

        cout<<\*it<<" ";

    }

    cout<<endl;

   //for set of elements

   s.erase(s.find(12),s.find(20));  //here 12 will be included and 20 will be excluded mean 12,19 will be deleted. \*\*\*never write big number first in the range it will give the segmentation fault\*\*\*....\*\*also do not write the non existing element in the set bcoz when element is not found using find function it return the iterator of end element that later create a problem\*\*

    for(auto it=s.begin(); it!=s.end();it++)

    {

        cout<<\*it<<" ";

    }

    cout<<endl;

    multiset <int> s;

    //1.insert->> inserts new element in set

    s.insert(10);

    s.insert(23);

    s.insert(10);

    s.insert(15);

    s.insert(30);

    s.insert(20);

    s.insert(15);

    for(auto it=s.begin();it!=s.end();it++)

    {

        cout<<\*it<<" ";

    }

    cout<<endl;

    s.erase(s.find(10)); // it just erases the first  10 other duplicates are not touched

    s.erase(10); // it earses all the 10

    for(auto it:s)

    {

        cout<<it<<" ";

    }

}

**Unordered Set**

#include<iostream>

#include<unordered\_set>

using namespace std;

 int main()

 {

     unordered\_set <int> s;  // unordered set do not allow duplicate value

     s.insert(10);

     s.insert(20);

     s.insert(15);

     s.insert(30);

     for(auto it=s.begin();it!=s.end();it++)

     {

         cout<<\*it<<" ";

     }

     cout<<endl;

     //s.erase(10); // remove all the 10

     s.erase(s.find(10));

     for(auto it=s.begin();it!=s.end();it++)

     {

         cout<<\*it<<" ";

     }

     cout<<endl;

      unordered\_multiset <int> s1;  // unordered multiset do allow duplicate value

     s1.insert(10);

     s1.insert(20);

     s1.insert(15);

     s1.insert(10);

     s1.insert(32);

     s1.insert(20);

     s1.insert(15);

     for(auto it=s1.begin();it!=s1.end();it++)

     {

         cout<<\*it<<" ";

     }

     cout<<endl;

     //s.erase(10); // remove all the 10

     s1.erase(s1.find(10));

     for(auto it=s1.begin();it!=s1.end();it++)

     {

         cout<<\*it<<" ";

     }

 }

**UNORDERED Map**

/\* map is a associative containers that store elements in mapped fashion

each element is has a key value and a mapped value.  we cannot repeat key but can override value

    -->>ordered:-> based on balanced Binary tree of keys, keys are in sorted order o(logn)

    |

map--

    |

    -->>unordered:->implemented using hashing O(1)

Multimap : multimap takes duplicate value

 \*/

#include<iostream>

#include<map>

using namespace std;

int main()

{

    map <int ,int> m;  //<int,string>, <any data\_type,data\_type>

     //1.insert-> ut insert a new key value pair in the map

     m.insert({10,5});

     m.insert(make\_pair(4,15));

     m.insert(make\_pair(10,6)); //using insert function we cannot override the value

     for(auto it =m.begin();it!=m.end();it++)

     {

         cout<<it->first<<" "<<(\*it).second<<endl; // the output will show that value of key 10 is not override

     }

     m[10]=6;  // this method can be used to override the value and also be used to insert the new values as shown below

     m[11]=34; // but this is not allowed in multimap

cout<<endl;

     for(auto it=m.begin();it!=m.end();it++)

     {

         cout<<it->first<<" "<<it->second<<endl;

     }

     //2.find-> searchess for given value and retuns the iterator

     if(m.find(10)!=m.end())

     {

         cout<<m.find(10)->second<<endl;// it can also be replaced by

         cout<<m[10];

     }

     //3.erase-->> it is used to erase the key value

     m.erase(10);

     for( auto it=m.begin();it!=m.end();it++)

     {

         cout<<it->first<<" "<<it->second<<endl;

     }

     cout<<m[100]<<endl;//when we try to a access some random key it returns 0(Zero) and it also insert that key in map;

    multimap <int,int> m1;

    m1.insert(make\_pair(4,6));

    m1.insert(make\_pair(10,9));

    m1.insert(make\_pair(12,6));

    m1.insert(make\_pair(10,7));

    for(auto it=m1.begin();it!=m1.end();it++)

    {

        cout<<it->first<<" "<<it->second<<endl;

    }

}