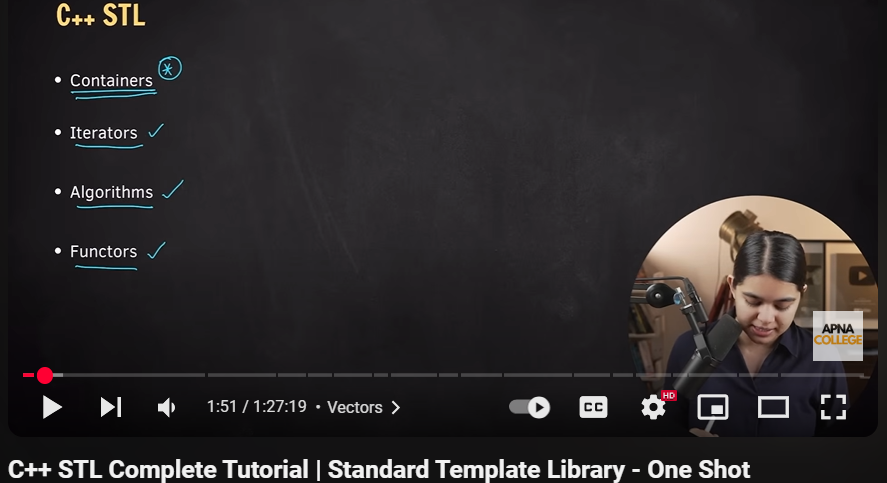
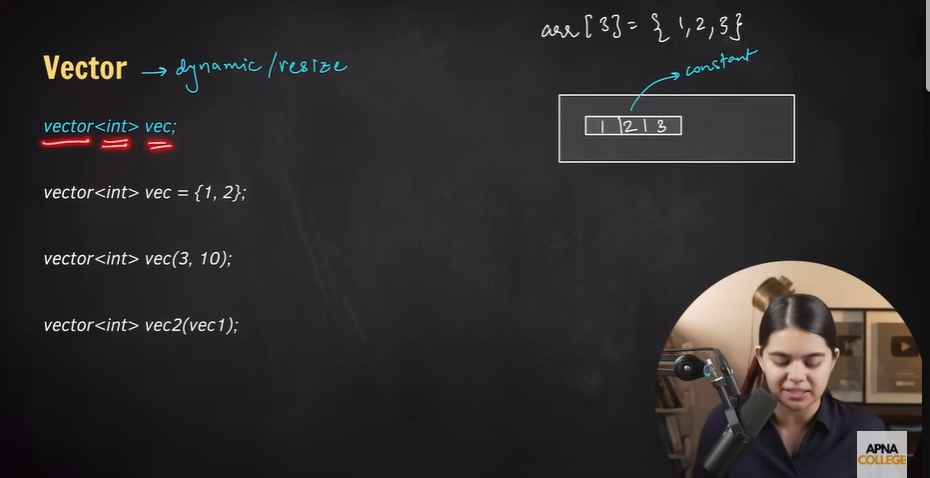
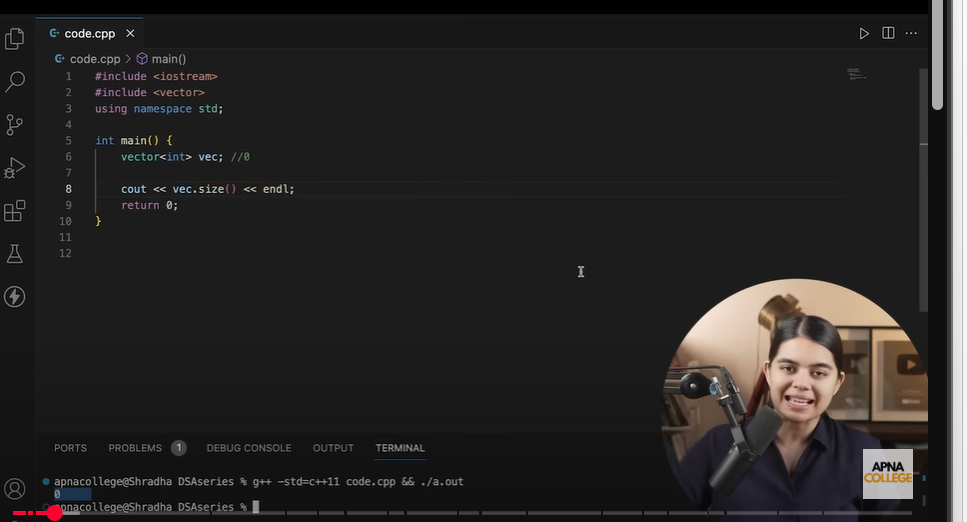
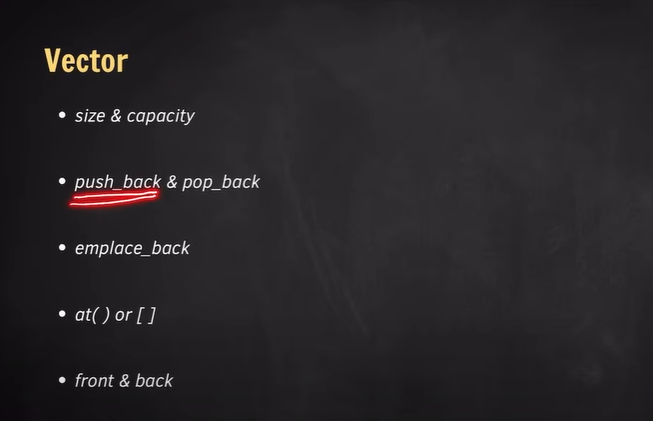
C++ STL

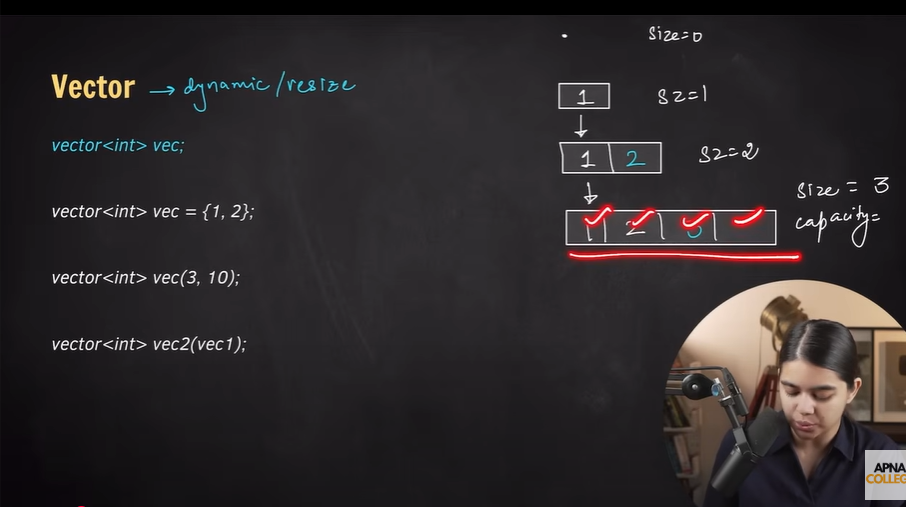


Vector:









Code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec; //size 0

    vec.push\_back(1);

    vec.push\_back(2);

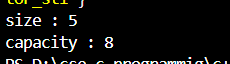
    vec.push\_back(3);

    cout << "size : " << vec.size() << endl;

    cout << "capacity : " << vec.capacity() << endl;

    return 0;

}

Result : 

Print value code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec; //size 0

    vec.push\_back(1);

    vec.push\_back(2);

    vec.push\_back(3);

    vec.push\_back(4);

    vec.push\_back(5);

    for(int value : vec)

    {

        cout << value << " ";

    }

    return 0;

}

Result:



Emplace\_back code :

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec; //size 0

    vec.push\_back(1);

    vec.push\_back(2);

    vec.push\_back(3);

    vec.push\_back(4);

    vec.push\_back(5);

    vec.emplace\_back(6);

    for(int value : vec)

    {

        cout << value << " ";

    }

    return 0;

}

Result:



Push\_back,,,pop\_back code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec; //size 0

    vec.push\_back(1);

    vec.push\_back(2);

    vec.push\_back(3);

    vec.push\_back(4);

    vec.push\_back(5);

    vec.emplace\_back(6);

    vec.pop\_back();

    vec.pop\_back();

    for(int value : vec)

    {

        cout << value << " ";

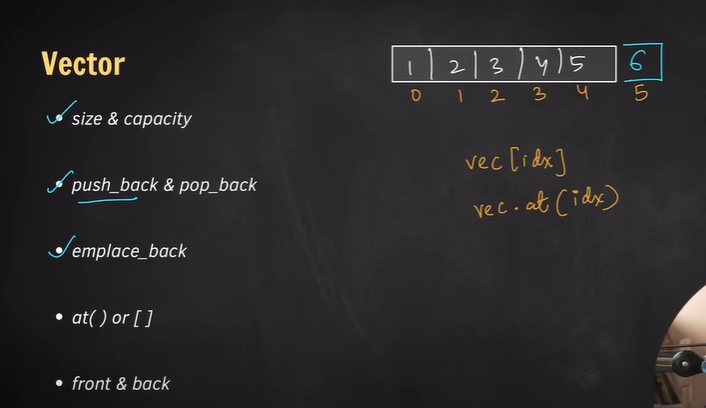
    }

    return 0;

}

Result:





At() or [] code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec; //size 0

    vec.push\_back(1);

    vec.push\_back(2);

    vec.push\_back(3);

    vec.push\_back(4);

    vec.push\_back(5);

    vec.emplace\_back(6);

    vec.pop\_back();

    vec.pop\_back();

    for(int value : vec)

    {

        cout << value << " ";

    }

    cout << endl;

    cout << "value at index 2 : " << vec.at(2) << " or " << vec[2] << endl;

    return 0;

}

Result:



Front ,,,,,back code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec; //size 0

    vec.push\_back(1);

    vec.push\_back(2);

    vec.push\_back(3);

    vec.push\_back(4);

    vec.push\_back(5);

    vec.emplace\_back(6);

    vec.pop\_back();

    vec.pop\_back();

    for(int value : vec)

    {

        cout << value << " ";

    }

    cout << endl;

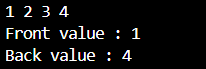
    cout << "Front value : " << vec.front() << endl;

    cout << "Back value : " << vec.back() << endl;

    return 0;

}

Result:



Vector initialize:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {1 , 2 , 3 , 4 , 5};

    vec.pop\_back();

    for(int value : vec)

    {

        cout << value << " ";

    }

    cout << endl;

    return 0;

}

Result :



Vector initialize same value code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec(6, 10); //vec.(size, value);

    for(int value : vec)

    {

        cout << value << " ";

    }

    cout << endl;

    return 0;

}

Result:



Vector initialize other vector value code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec1 = {1 , 2 , 3 , 4 , 5};

    vector<int> vec2(vec1);

    for(int value : vec2)

    {

        cout << value << " ";

    }

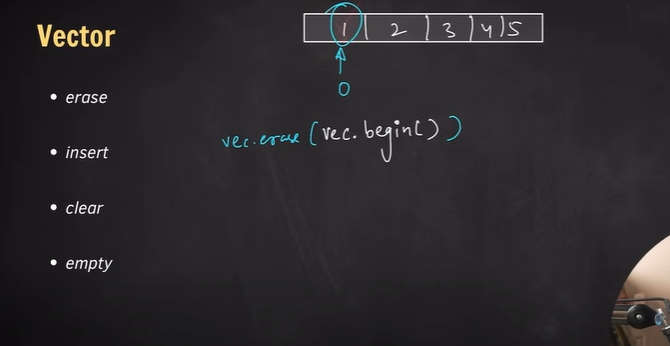
    cout << endl;

    return 0;

}

Result :





Erase code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {1, 2, 3, 4, 5};

    //vec.erase(vec.begin()); //intdex no 0 will be erased

    vec.erase(vec.begin() + 2); //intdex no 2 will be erased

    for(int val : vec)

    {

        cout << val << " ";

    }

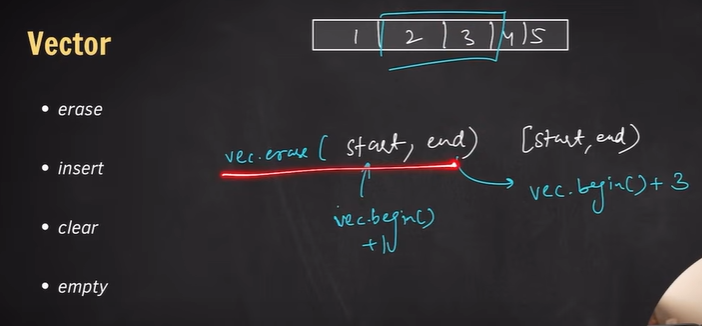
    cout << endl;

    return 0;

}

Result :





Erase(start , end) code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {1, 2, 3, 4, 5};

    //vec.erase(vec.begin()); //index no 0 will be erased

    //vec.erase(vec.begin() + 2); //index no 2 will be erased

    vec.erase(vec.begin() + 1 , vec.begin() + 3);//index no 1 to 2 will be erased

    //start is included but end is not included

    for(int val : vec)

    {

        cout << val << " ";

    }

    cout << endl;

    return 0;

}

Result:



\*\*\*erase size change kore, but capacity same thake

Code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {1, 2, 3, 4, 5};

    cout << "size before erasing : " << vec.size() << endl;

    cout << "capacity before erasing : " << vec.capacity() << endl;

    //vec.erase(vec.begin()); //index no 0 will be erased

    //vec.erase(vec.begin() + 2); //index no 2 will be erased

    vec.erase(vec.begin() + 1 , vec.begin() + 3);//index no 1 to 2 will be erased

    //start is included but end is not included

    cout << "values : " ;

    for(int val : vec)

    {

        cout << val << " ";

    }

    cout << endl;

    //erase size change kore but capacity change korena

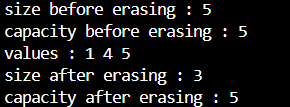
    cout << "size after erasing : " << vec.size() << endl;

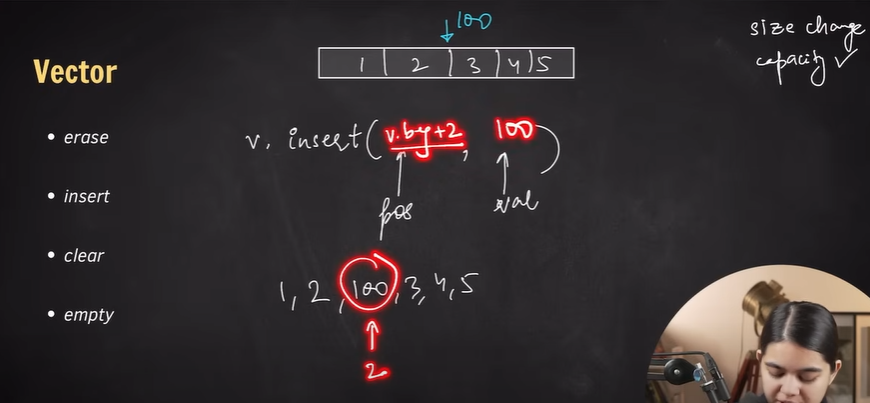
    cout << "capacity after erasing : " << vec.capacity() << endl;

    return 0;

}

Result:





Insert function code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {1 , 2 , 3 , 4 , 5};

    vec.insert(vec.begin() + 2 , 100);

    cout << "values : ";

    for(int value : vec)

    {

        cout << value << " ";

    }

    cout << endl;

    return 0;

}

Result:



Clear function code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {1 , 2 , 3 , 4 , 5};

    vec.clear();

    cout << "values : ";

    for(int value : vec)

    {

        cout << value << " ";

    }

    cout << endl;

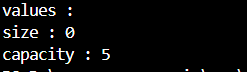
    cout << "size : " << vec.size() << endl;

    cout << "capacity : " << vec.capacity() << endl;

    return 0;

}

Result:



Empty function code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {1 , 2 , 3 , 4 , 5};

    vec.clear();

    cout << "values : ";

    for(int value : vec)

    {

        cout << value << " ";

    }

    cout << endl;

    cout << "size : " << vec.size() << endl;

    cout << "capacity : " << vec.capacity() << endl;

    cout << "Is empty? : " << vec.empty() << endl;

    //vec.empty() returns bool value

    cout << "Is empty? : ";

    if(vec.empty())

    {

        cout << "Yes";

    }

    else

    {

        cout << "No";

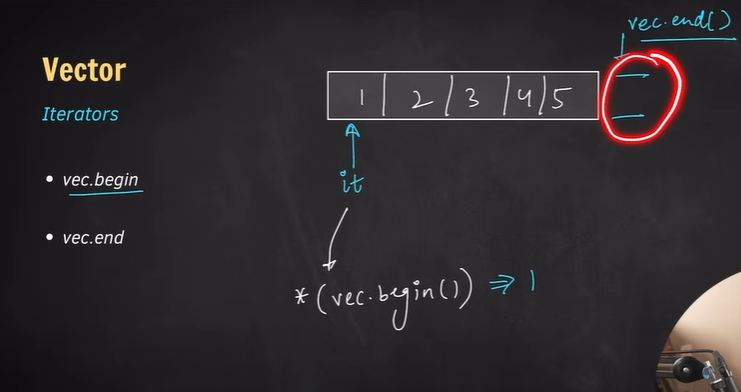
    }

    return 0;

}

Result:





Begin() ,,,,end() code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {1 , 2 , 3 , 4 , 5};

    cout << "values : ";

    for(int value : vec)

    {

        cout << value << " ";

    }

    cout << endl;

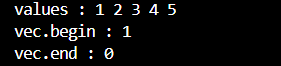
    cout << "vec.begin : " << \*(vec.begin()) << endl;

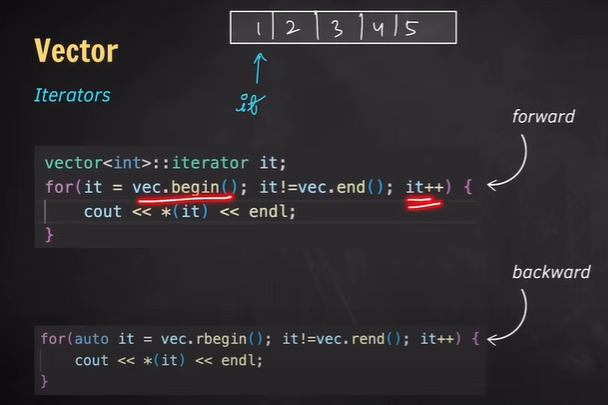
    cout << "vec.end : " << \*(vec.end()) << endl; //vec.end() garbage value outpuut dei

    return 0;

}

Result:





Iterator code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {1 , 2 , 3 , 4 , 5};

    cout << "values : ";

    vector<int>::iterator it;

    for(it = vec.begin(); it != vec.end(); it++)

    {

        cout << \*it << " ";

    }

    cout << endl;

    return 0;

}

Result:



Reverse iterator code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {1 , 2 , 3 , 4 , 5};

    cout << "values : ";

    vector<int>::reverse\_iterator it;

    for(it = vec.rbegin(); it != vec.rend(); it++)

    {

        cout << \*it << " ";

    }

    cout << endl;

    //or, amra auto keyword use koreo type define korte pari

    cout << "values : ";

    for(auto it = vec.rbegin(); it != vec.rend(); it++)

    {

        cout << \*it << " ";

    }

    cout << endl;

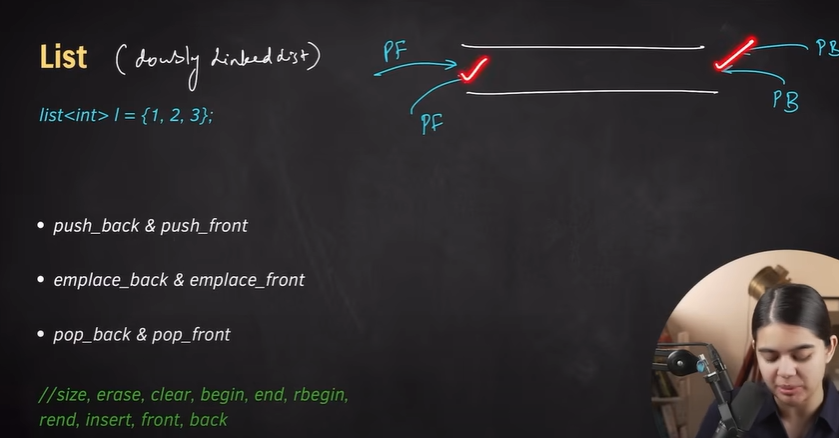
    return 0;

}

Result :



List



Push\_back, push\_front,

Emplace\_back, emplace\_front,

Pop\_back, pop\_front

Code:

#include<iostream>

#include<list>

using namespace std;

int main()

{

    list<int> list1 = {1, 2, 3, 4, 5};

    cout << "List 1 elements : ";

    for(int element : list1)

    {

        cout << element << " ";

    }

    cout << endl;

    list<int> list2;

    list2.push\_back(1);

    list2.emplace\_back(2);

    list2.push\_front(3);

    list2.emplace\_front(5);

    cout << "List 2 elements before poping: ";

    for(int element : list2)

    {

        cout << element << " ";

    }

    cout << endl;

    list2.pop\_back();

    list2.pop\_front();

    cout << "List 2 elements after poping: ";

    for(int element : list2)

    {

        cout << element << " ";

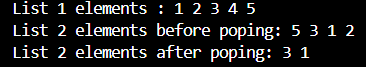
    }

    cout << endl;

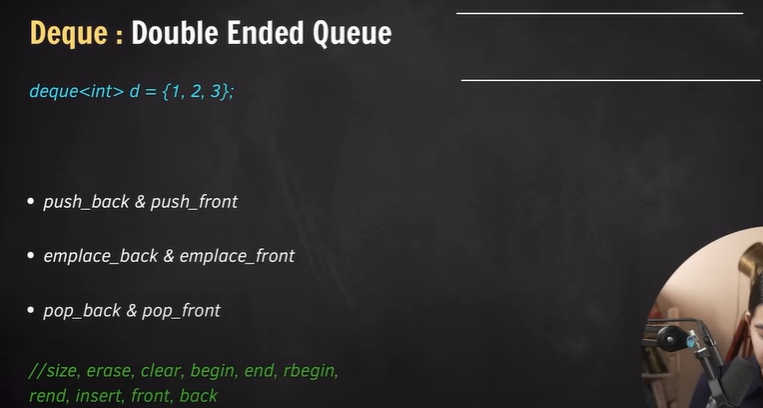
    return 0;

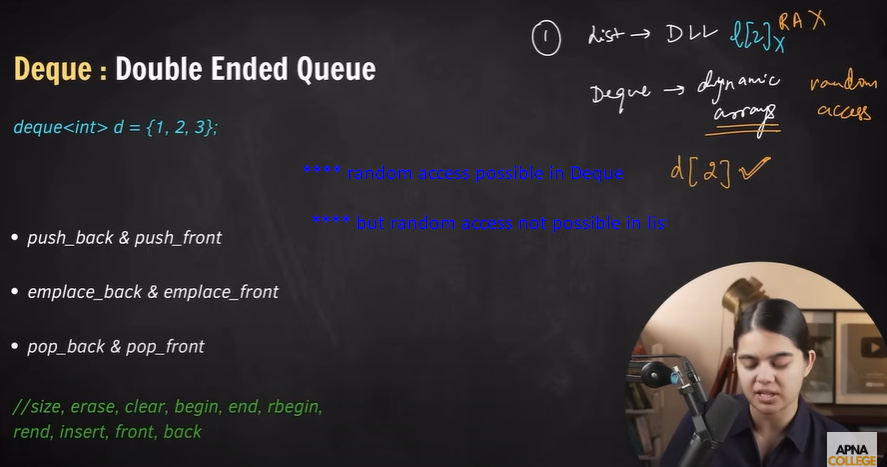
}

Result:



Deque





Deque random access code:

#include<iostream>

#include<deque>

using namespace std;

int main()

{

    deque<int> d = {1, 2, 3, 4, 5};

    for(int val : d)

    {

        cout << val << " ";

    }

    cout << endl;

    cout << d[2] << endl;

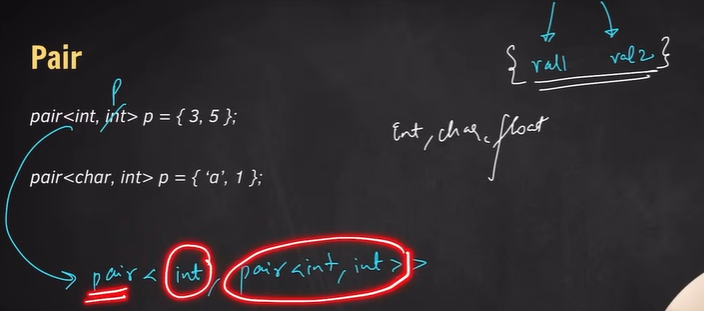
    return 0;

}

Result:



Pair



Pair code:

#include<iostream>

using namespace std;

int main()

{

    pair<string,int> p = {"rangan", 5};

    cout << p.first << endl;

    cout << p.second << endl;

    return 0;

}

Result:



Pair of pair code:

#include<iostream>

using namespace std;

int main()

{

    pair<string, pair<char,int>> p = {"rangan", {'a', 5}};

    cout << p.first << endl;

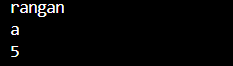
    cout << p.second.first << endl;

    cout << p.second.second << endl;

    return 0;

}

Result:



Pair in vector

Difference between push\_back and emplace\_back

Code:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<pair<char,int>> vec =  {{'a', 5} , {'b' , 6} , {'c' , 9}};

    vec.push\_back({'d' , 7});//push\_back korte parenthesis er moddhe curly braces diye input dite hobe

    vec.emplace\_back('e' , 2);//emplace\_back e curly braces lagbena , in-place ojbects create kore

    for(pair<char, int> element : vec)

    //or for(auto element : vec) likheo element declare kora jabe

    {

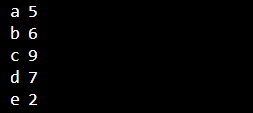
        cout << element.first << " " << element.second << endl;

    }

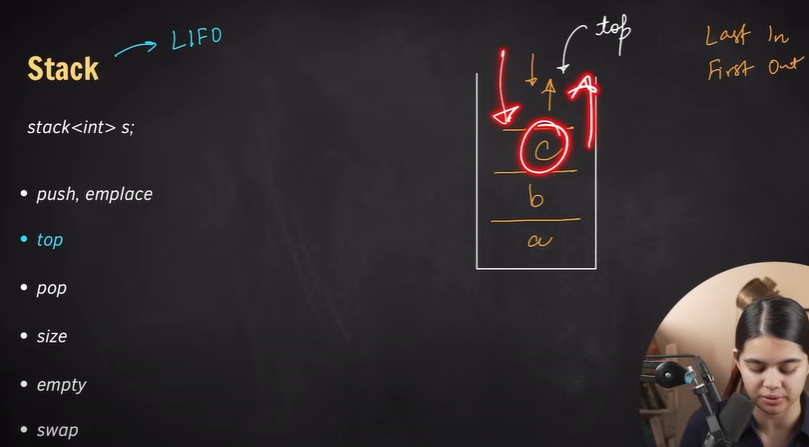
    return 0;

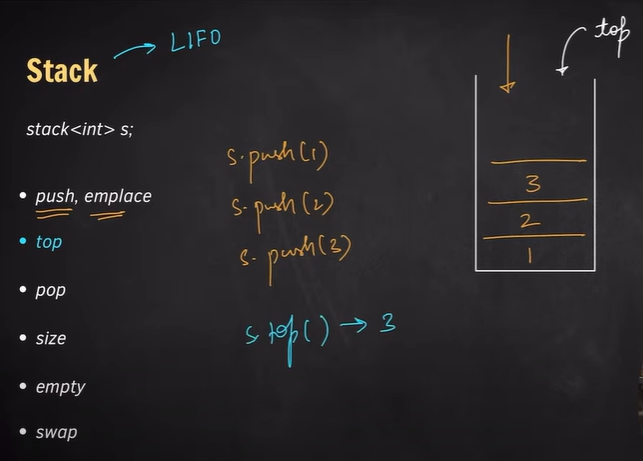
}

Result:



Stack





Push(),,,emplace(),,,,pop(),,,,empty()

code:

#include<iostream>

#include<stack>//last in first out

using namespace std;

int main()

{

    stack<int> s;

    s.push(1);

    s.push(2);

    s.emplace(3);

    s.emplace(4);

    s.pop(); //4 will be popped

    while(!s.empty())

    {

        cout << s.top() << endl;

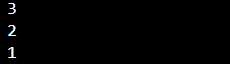
        s.pop();

    }

    return 0;

}

Result:



Swap(),,,,,size()

Code:

#include<iostream>

#include<stack>//last in first out

using namespace std;

int main()

{

    stack<int> s1;

    s1.push(1);

    s1.push(2);

    s1.emplace(3);

    s1.emplace(4);

    stack<int> s2;

    cout << "s1 size before swapping : " << s1.size() <<endl;

    cout << "s2 size before swapping : " << s2.size() <<endl;

    s2.swap(s1);

    cout << "s1 size after swapping : " << s1.size() <<endl;

    cout << "s2 size after swapping : " << s2.size() <<endl;

    cout << "s1 elements : ";

    while(!s1.empty())

    {

        cout << s1.top() << " ";

        s1.pop();

    }

    cout << endl;

    cout << "s2 elements : ";

    while(!s2.empty())

    {

        cout << s2.top() << " ";

        s2.pop();

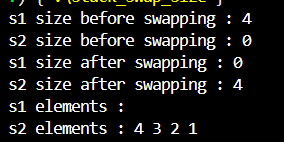
    }

    cout << endl;

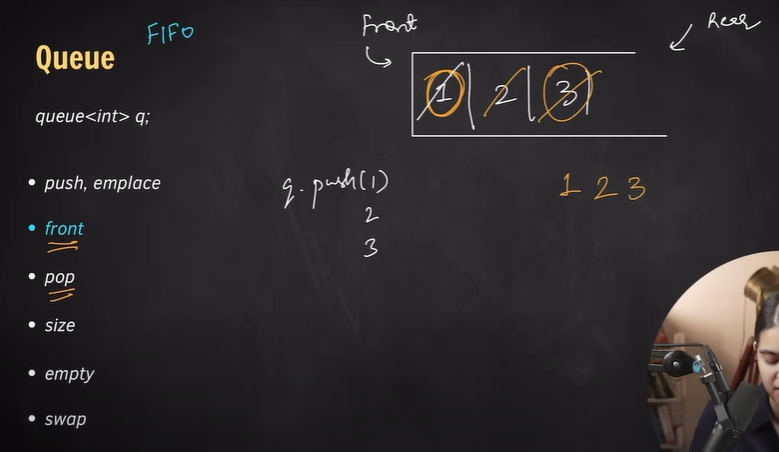
    return 0;

}

Result:



Queue



Push(),,,,,emplace(),,,,pop(),,,,front(),,,,empty()

Code:

#include<iostream>

#include<queue>//first in first out

using namespace std;

int main()

{

    queue<int> q;

    q.push(1);

    q.push(2);

    q.emplace(3);

    q.emplace(4);

    q.pop(); //1 will be popped

    while(!q.empty())

    {

        cout << q.front() << " ";

        q.pop();

    }

    cout << endl;

    return 0;

}

Result:



Priority\_queue



Push,, pop,,emplace,,top,,empty,,size

Code:

#include<iostream>

#include<queue>

using namespace std;

int main()

{

    priority\_queue<int> q;//largest value -> highest priority

    q.push(5);

    q.push(3);

    q.emplace(10);

    q.emplace(4);

    q.emplace(18);

    q.pop(); //18 will be popped

    cout << "size : " << q.size() << endl;

    cout << "priority queue elements :\n";

    while(!q.empty())

    {

        cout << q.top() << endl;

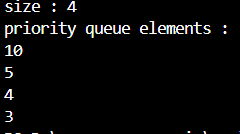
        q.pop();

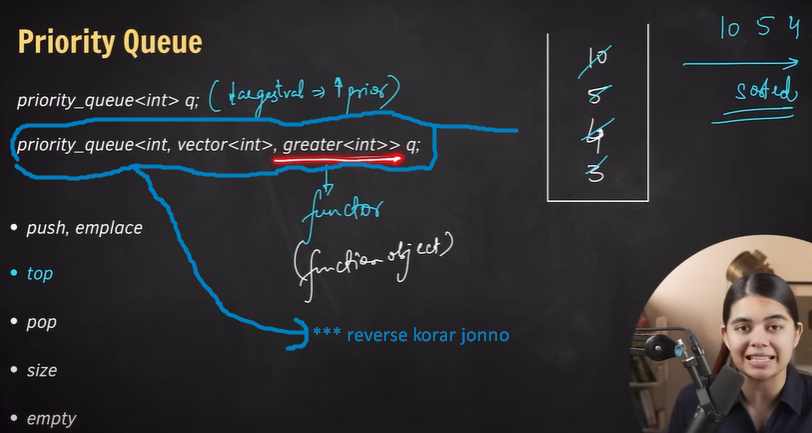
    }

    return 0;

}

Result:





Reverse code:

#include<iostream>

#include<queue>

using namespace std;

int main()

{

    priority\_queue<int, vector<int>, greater<int>> q;//largest value -> highest priority

    q.push(5);

    q.push(3);

    q.emplace(10);

    q.emplace(4);

    q.emplace(18);

    q.pop(); //3 will be popped

    cout << "size : " << q.size() << endl;

    cout << "priority queue elements :\n";

    while(!q.empty())

    {

        cout << q.top() << endl;

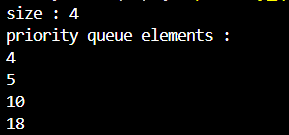
        q.pop();

    }

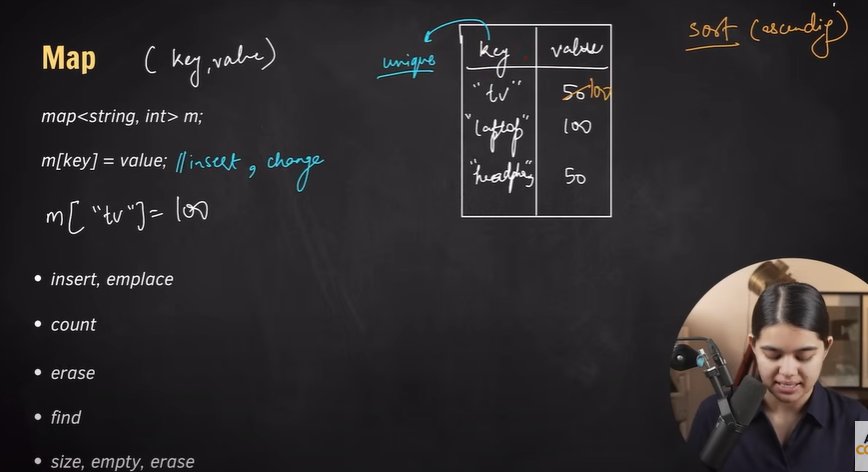
    return 0;

}

Result:



Map



Map code:

(sorted order e iutput ashbe)

#include<iostream>

#include<map>

using namespace std;

int main()

{

    map<string , int> m;

    m["tv"] = 100; //egula sorted order e print hobe

    m["tv"] = 99; /\*same key duibar thakleo ekbar e print hobe

                  second bar same key insert korle prothom barer

                  key er value second barer value diye replaced hobe\*/

    m["laptop"] = 120;

    m["headphone"] = 50;

    m["tablet"] = 130;

    m["watch"] = 70;

    for(auto p : m)

    {

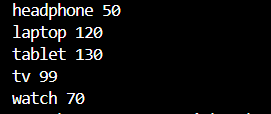
        cout << p.first << " " << p.second << endl;

    }

    return 0;

}

Result:



Insert,,emplace,,count,,,value,,,erase

Code:

#include<iostream>

#include<map>

using namespace std;

int main()

{

    map<string , int> m;

    m["tv"] = 100; //egula sorted order e print hobe

    m["laptop"] = 120;

    m["headphone"] = 50;

    m["tablet"] = 130;

    m["watch"] = 70;

    m.insert({"camera" , 25});

    m.emplace("mobile", 40);

    m.erase("headphone");

    for(auto p : m)

    {

        cout << p.first << " " << p.second << endl;

    }

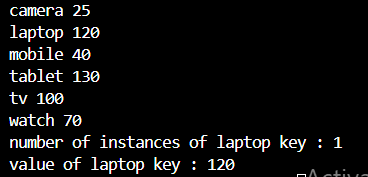
    cout << "number of instances of laptop key : " <<m.count("laptop") << endl;

    cout << "value of laptop key : " << m["laptop"] << endl;

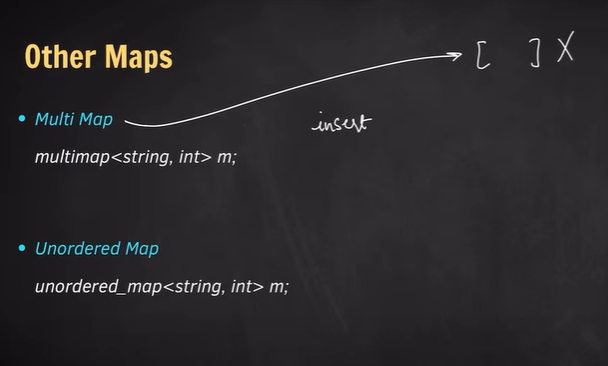
    return 0;

}

Result:



Multimap



Insert,,, emplace code:

#include<iostream>

#include<map>

using namespace std;

int main()

{

    multimap<string , int> m;

    m.insert({"tv", 100});

    m.insert({"tv", 120});

    m.emplace("tv", 99);

    m.emplace("tv", 100);

    for(auto p : m)

    {

        cout << p.first << " " << p.second << endl;

    }

    return 0;

}

Result: (same key multiple times output debe, jei order e input dewa hoyeche shei order e)



Erase code:

Sobgula same key erase korar jonno

#include<iostream>

#include<map>

using namespace std;

int main()

{

    multimap<string , int> m;

    m.insert({"tv", 100});

    m.insert({"tv", 100});

    m.emplace("tv", 100);

    m.emplace("tv", 100);

    m.erase("tv");//sobgula tv key ke erase kore debe

    for(auto p : m)

    {

        cout << p.first << " " << p.second << endl;

    }

    return 0;

}

Result:



Erase code 2:

Sudhu first same key erase korar jonno find() function use korte hobe erase er moddhe. Ebhabe iterator use kore ekta key erase kora jai

#include<iostream>

#include<map>

using namespace std;

int main()

{

    multimap<string , int> m;

    m.insert({"tv", 90});

    m.insert({"tv", 100});

    m.emplace("tv", 282);

    m.emplace("tv", 737);

    m.erase(m.find("tv"));//sudhu prothom tv key ke erase kore debe

    for(auto p : m)

    {

        cout << p.first << " " << p.second << endl;

    }

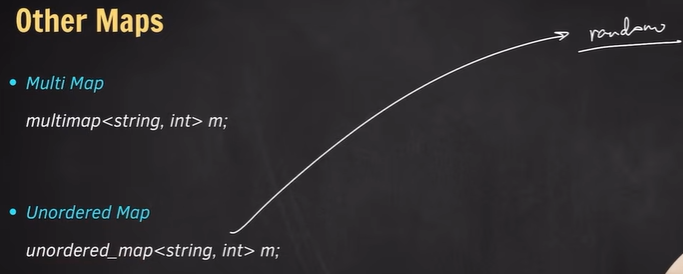
    return 0;

}

Result:



Unordered map



Code:

#include<iostream>

#include<unordered\_map>

using namespace std;

int main()

{

    unordered\_map<string , int> m;

    m.insert({"tv", 33});

    m.insert({"tv", 99});

    m.emplace("tv", 64);

    m.emplace("tv", 83);

    for(auto p : m)

    {

        cout << p.first << " " << p.second << endl;

    }

    return 0;

}

Result:

(same key multiple time insert korleo ekbar e output dibe

Sheta holo first key er value)



Code:

#include<iostream>

#include<unordered\_map>

using namespace std;

int main()

{

    unordered\_map<string , int> m;

    m.insert({"tv", 33});

    m.insert({"laptop", 99});

    m.emplace("fridge", 64);

    m.emplace("watch", 83);

    for(auto p : m)

    {

        cout << p.first << " " << p.second << endl;

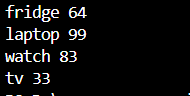
    }

    return 0;

}

Result:

(unsosrted order e output dibe)



Unordered\_multimap

\*\*same key unsorted bhabe multiple times output dei

Code:

#include<iostream>

#include<unordered\_map>

using namespace std;

int main()

{

    unordered\_multimap<string , int> m;

    m.insert({"tv", 33});

    m.insert({"tv", 99});

    m.emplace("tv", 64);

    m.emplace("tv", 83);

    for(auto p : m)

    {

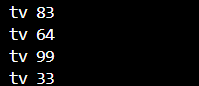
        cout << p.first << " " << p.second << endl;

    }

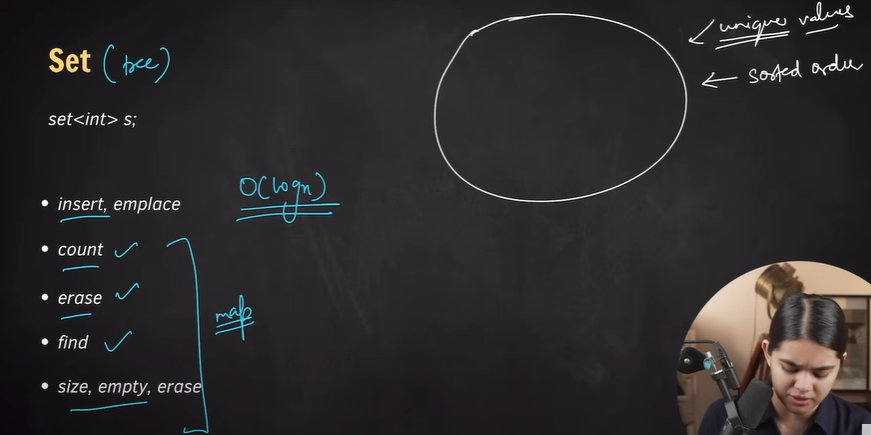
    return 0;

}

Result:



Set



Set code:

#include<iostream>

#include<set>

using namespace std;

int main()

{

    set<int> s;

    s.insert(1);

    s.insert(8);

    s.emplace(3);

    s.emplace(4);

    s.emplace(1); //same value duibar innput dile duibar same output deina

    s.emplace(2); //output er value sorted hobe

    s.emplace(3);

    cout << "size : " <<s.size() << endl;

    for(auto val : s)

    {

        cout << val << " ";

    }

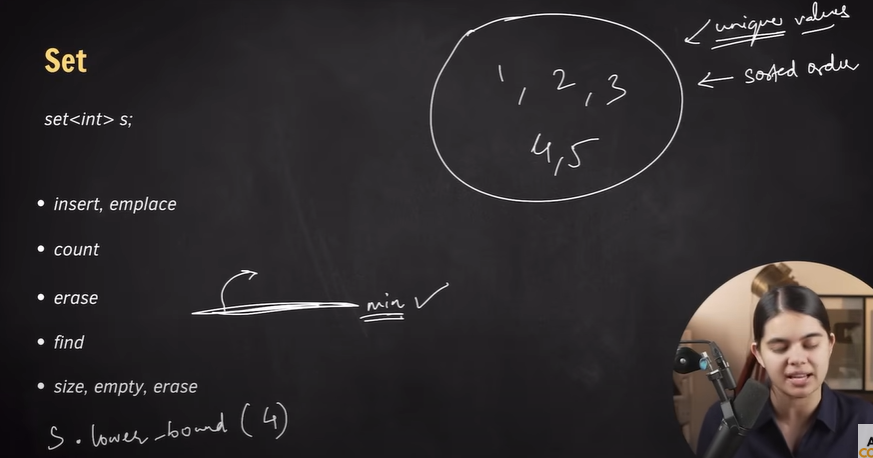
    cout << endl;

    return 0;

}

Result:





Lower\_bound code:

#include<iostream>

#include<set>

using namespace std;

int main()

{

    //lower bound means minimmum

    set<int> s;

    s.insert(1);

    s.insert(2);

    s.insert(3);

    s.insert(4);

    cout << "lower bound of s : " << \*(s.lower\_bound(3)) << endl; //3

    for(auto val : s)

    {

        cout << val << " ";

    }

    cout << endl;

    set<int> s2;

    s2.insert(4);

    s2.insert(2);

    s2.insert(9);

    s2.insert(6);

    cout << "lower bound of s2 : " << \*(s2.lower\_bound(3)) << endl; //4

    for(auto val : s2)

    {

        cout << val << " ";

    }

    cout << endl;

    set<int> s3;

    s3.insert(1);

    s3.insert(2);

    s3.insert(5);

    cout << "lower bound of s3 : " << \*(s3.lower\_bound(7)) << endl; //0

    for(auto val : s3)

    {

        cout << val << " ";

    }

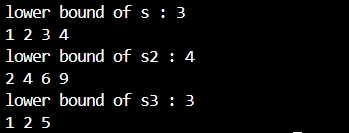
    cout << endl;

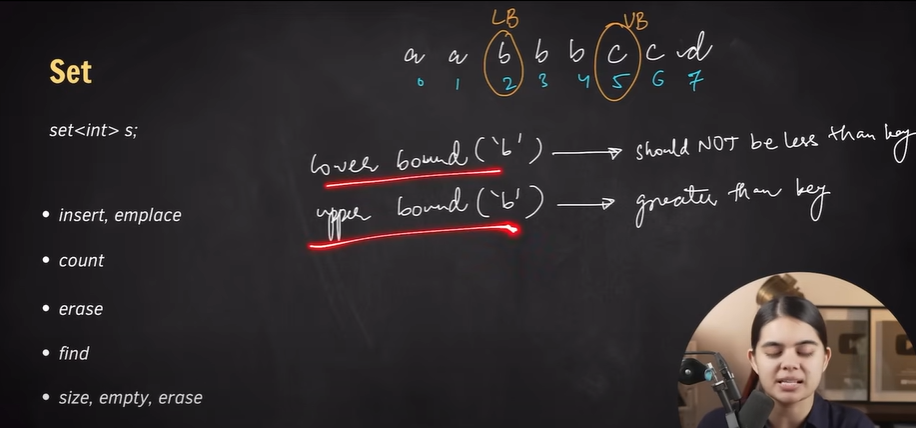
    return 0;

}

Result:

(last er tar lower bound er output debe s3.end() er value ,jeta garbage value, etar output 0 dewar kotha)





Upper\_bound code:

#include<iostream>

#include<set>

using namespace std;

int main()

{

    //lower bound means minimmum or <=

    //upper bound means greater than or <

    set<int> s;

    s.insert(1);

    s.insert(2);

    s.insert(3);

    s.insert(4);

    cout << "lower bound of s : " << \*(s.lower\_bound(3)) << endl; //3

    cout << "upper bound of s : " << \*(s.upper\_bound(3)) << endl; //4

    for(auto val : s)

    {

        cout << val << " ";

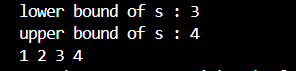
    }

    cout << endl;

    return 0;

}

**Result:**



Char datatype lower\_bound and upper\_bound and index number code:

#include<iostream>

#include<set>

using namespace std;

int main()

{

    //lower bound means minimmum or <=

    //upper bound means greater than or <

    set<char> s;

    s.insert('a');

    s.insert('c');

    s.insert('b');

    s.insert('b');

    s.insert('b');

    s.insert('c');

    s.insert('d');

    //distance(s.begin() , iterator) to show the index number by calculating the distance between begin() and iterator

    cout << "lower bound of s : " << \*(s.lower\_bound('b')) << " at index : " << distance(s.begin() , s.lower\_bound('b')) << endl; //b

    cout << "upper bound of s : " << \*(s.upper\_bound('b')) << " at index : " << distance(s.begin() , s.upper\_bound('b')) << endl; //c

    for(auto val : s)

    {

        cout << val << " ";

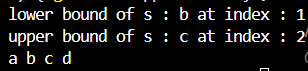
    }

    cout << endl;

    return 0;

}

Result:



Multiset index output code:

#include<iostream>

#include<set>

using namespace std;

int main()

{

    multiset<char> s;

    // Inserting elements (order doesn't matter, multiset will sort them)

    s.insert('a');

    s.insert('c');

    s.insert('b');

    s.insert('b');

    s.insert('b');

    s.insert('c');

    s.insert('d');

    // The multiset will be sorted: a b b b c c d

    // Indices:     0: a

    //              1: b

    //              2: b

    //              3: b

    //              4: c

    //              5: c

    //              6: d

    cout << "lower bound of 'b': " << \*(s.lower\_bound('b'))

         << " at index: " << distance(s.begin(), s.lower\_bound('b')) << endl;

    // Output: b at index 1 (first 'b')

    cout << "upper bound of 'b': " << \*(s.upper\_bound('b'))

         << " at index: " << distance(s.begin(), s.upper\_bound('b')) << endl;

    // Output: c at index 4 (first element after all 'b's)

    cout << "Multiset elements: ";

    for(auto val : s) {

        cout << val << " ";

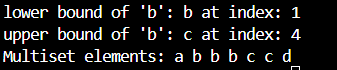
    }

    cout << endl;

    // Output: a b b b c c d

    return 0;

}

Result:  


Unordered set code:

#include<iostream>

#include<unordered\_set>

using namespace std;

int main()

{

    unordered\_set<char> s;

    s.insert('a');

    s.insert('c');

    s.insert('b');

    s.insert('b');

    s.insert('b');

    s.insert('c');

    s.insert('d');

    for(auto val : s)

    {

        cout << val << " ";

    }

    cout << endl;

    return 0;

}

Result:



Unordered multiset code:

#include<iostream>

#include<unordered\_set>

using namespace std;

int main()

{

    unordered\_multiset<char> s;

    s.insert('a');

    s.insert('c');

    s.insert('b');

    s.insert('b');

    s.insert('b');

    s.insert('c');

    s.insert('d');

    cout << "unordered\_multiset elements: ";

    for(auto val : s)

    {

        cout << val << " ";

    }

    cout << endl;

    return 0;

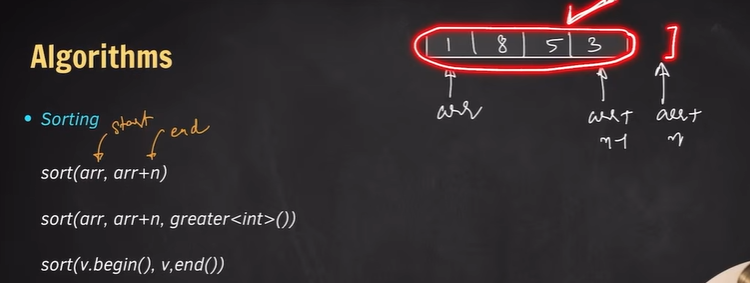
}

Result:



Algorithms

Sorting



Array sorting code:

#include<iostream>

#include<algorithm>

using namespace std;

int main()

{

    int arr[] = {3, 8, 5, 2, 1};

    sort(arr + 1 , arr + 3); /\*will sort elements from index1 to

                              index2 ,,, index3 will not be included \*/

    cout << "Sorted Array elements(index\_1 to index\_2) : ";

    for(int val : arr)

    {

        cout << val << " ";

    }

    cout << endl;

    sort(arr , arr + 5);

    cout << "Sorted Array elements(index\_0 to index\_5) : ";

    for(int val : arr)

    {

        cout << val << " ";

    }

    cout << endl;

    return 0;

}

Result:



Vector sorting code:

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {3, 8, 5, 2, 1};

    sort(vec.begin() + 1, vec.begin() + 3); /\*will sort elements from index1 to

                              index2 ,,, index3 will not be included \*/

    cout << "Sorted Array elements(index\_1 to index\_2) : ";

    for(int val : vec)

    {

        cout << val << " ";

    }

    cout << endl;

    sort(vec.begin() , vec.end());

    cout << "Sorted vector elements(index\_0 to index\_5) : ";

    for(int val : vec)

    {

        cout << val << " ";

    }

    cout << endl;

    return 0;

}

Result:



Sorted array(decending) code:

#include<iostream>

#include<algorithm>

using namespace std;

int main()

{

    int arr[] = {3, 8, 5, 2, 1};

    sort(arr + 1 , arr + 3, greater<int>()); /\*will sort elements from index1 to

                              index2 ,,, index3 will not be included \*/

    cout << "Sorted(decending) Array elements(index\_1 to index\_2) : ";

    for(int val : arr)

    {

        cout << val << " ";

    }

    cout << endl;

    sort(arr , arr + 5 , greater<int>());

    cout << "Sorted(decending) Array elements(index\_0 to index\_5) : ";

    for(int val : arr)

    {

        cout << val << " ";

    }

    cout << endl;

    return 0;

}

Result:



Sorted vector(decending) code:

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {3, 8, 5, 2, 1};

    sort(vec.begin() + 1, vec.begin() + 3 , greater<int>()); /\*will sort elements from index1 to

                              index2 ,,, index3 will not be included \*/

    cout << "Sorted(decending) vector elements(index\_1 to index\_2) : ";

    for(int val : vec)

    {

        cout << val << " ";

    }

    cout << endl;

    sort(vec.begin() , vec.end() , greater<int>());

    cout << "Sorted(decending) vector elements(index\_0 to index\_5) : ";

    for(int val : vec)

    {

        cout << val << " ";

    }

    cout << endl;

    return 0;

}

Result:



Sort pair code:

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

int main()

{

    vector<pair<int, int>> vec = {{3 , 1} , {7 , 1} , {2 , 1} , {5 , 2}};

    sort(vec.begin() + 1, vec.begin() + 3); /\*will sort elements from index1 to

                              index2 ,,, index3 will not be included \*/

    cout << "Sorted vector elements(index\_1 to index\_2) : \n";

    for(auto p : vec)

    {

        cout << p.first << " " << p.second << endl;

    }

    sort(vec.begin() , vec.end());

    cout << "Sorted vector elements(index\_0 to index\_5) : \n";

    for(auto p : vec)

    {

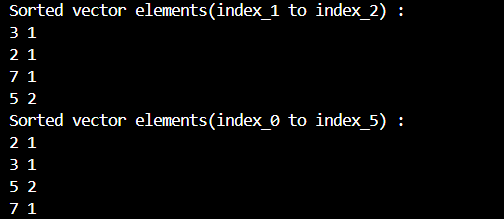
        cout << p.first << " " << p.second << endl;

    }

    return 0;

}

Result:



Sort pair according to second value code:

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

bool comparator(pair<int , int> p1 , pair<int , int> p2)

{

    if(p1.second < p2.second) return true;

    if(p1.second > p2.second) return false;

    if(p1.first < p2.first) return true;

    else return false;

}

int main()

{

    vector<pair<int, int>> vec = {{3 , 1} , {7 , 1} , {2 , 1} , {5 , 2}};

    sort(vec.begin() + 1, vec.begin() + 3 , comparator); /\*will sort elements from index1 to

                              index2 ,,, index3 will not be included \*/

    cout << "Sorted vector elements(index\_1 to index\_2) : \n";

    for(auto p : vec)

    {

        cout << p.first << " " << p.second << endl;

    }

    sort(vec.begin() , vec.end() , comparator);

    cout << "Sorted vector elements(index\_0 to index\_5) : \n";

    for(auto p : vec)

    {

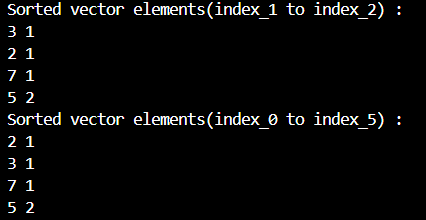
        cout << p.first << " " << p.second << endl;

    }

    return 0;

}

Result:



Reverse

Code:

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

int main()

{

    vector<int> vec = {2, 5 ,1 , 4, 3};

    reverse(vec.begin() , vec.end()); //reverse(vec.begin()+1 , vec.begin()+3) emon o kora jabe

    for(auto val : vec)

    {

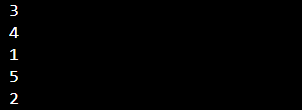
        cout << val << endl;

    }

    return 0;

}

Result:



Next\_permutation

Code:

#include<iostream>

#include<algorithm>

using namespace std;

int main()

{

    string s = "abc";

    next\_permutation(s.begin() , s.end());

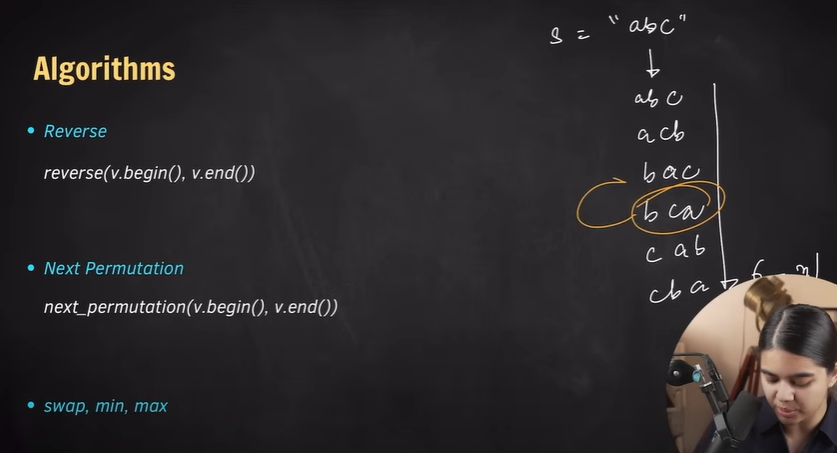
    cout << s << endl;

    return 0;

}

Result:





Previous permutation

Code:

#include<iostream>

#include<algorithm>

using namespace std;

int main()

{

    string s = "bca";

    prev\_permutation(s.begin() , s.end());

    cout << s << endl;

    return 0;

}

Result:



Max element

Code:

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int main()

{

    vector<int> vec = {1, 2 , 3, 4, 5};

    cout << \*(max\_element(vec.begin() , vec.end())) << endl;

    return 0;

}

Result:



Min element

Code:

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int main()

{

    vector<int> vec = {1, 2 , 3, 4, 5};

    cout << \*(min\_element(vec.begin() , vec.end())) << endl;

    return 0;

}

Result:



Binary search

Code:

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int main()

{

    vector<int> vec = {1, 2 , 3, 4, 5};

    cout << binary\_search(vec.begin() , vec.end() , 2) << endl;

    //or

    if(binary\_search(vec.begin() , vec.end() , 2))

    {

        cout << "found";

    }

    else

    {

        cout << "not found";

    }

    return 0;

}

Result:

