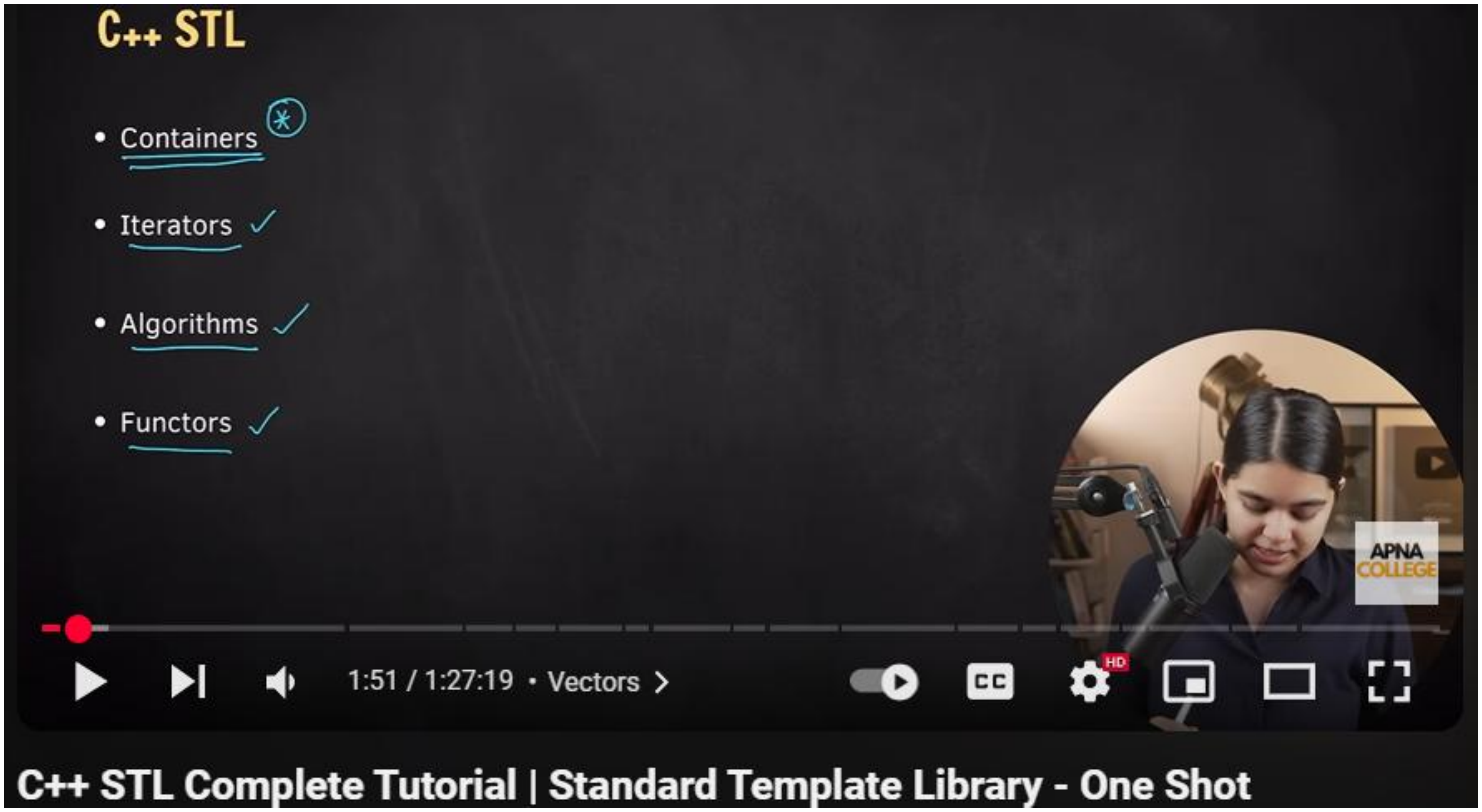
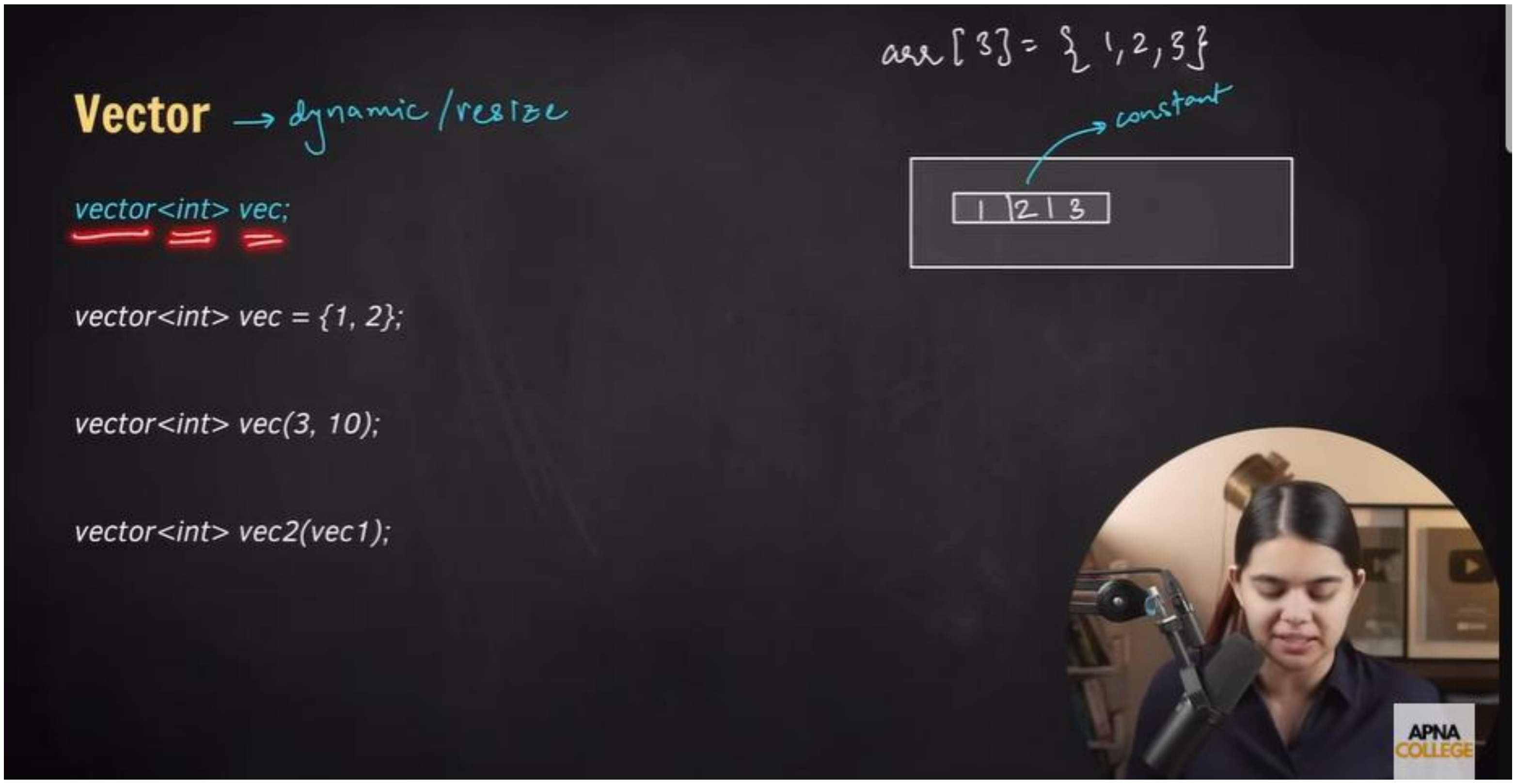
C++ STL



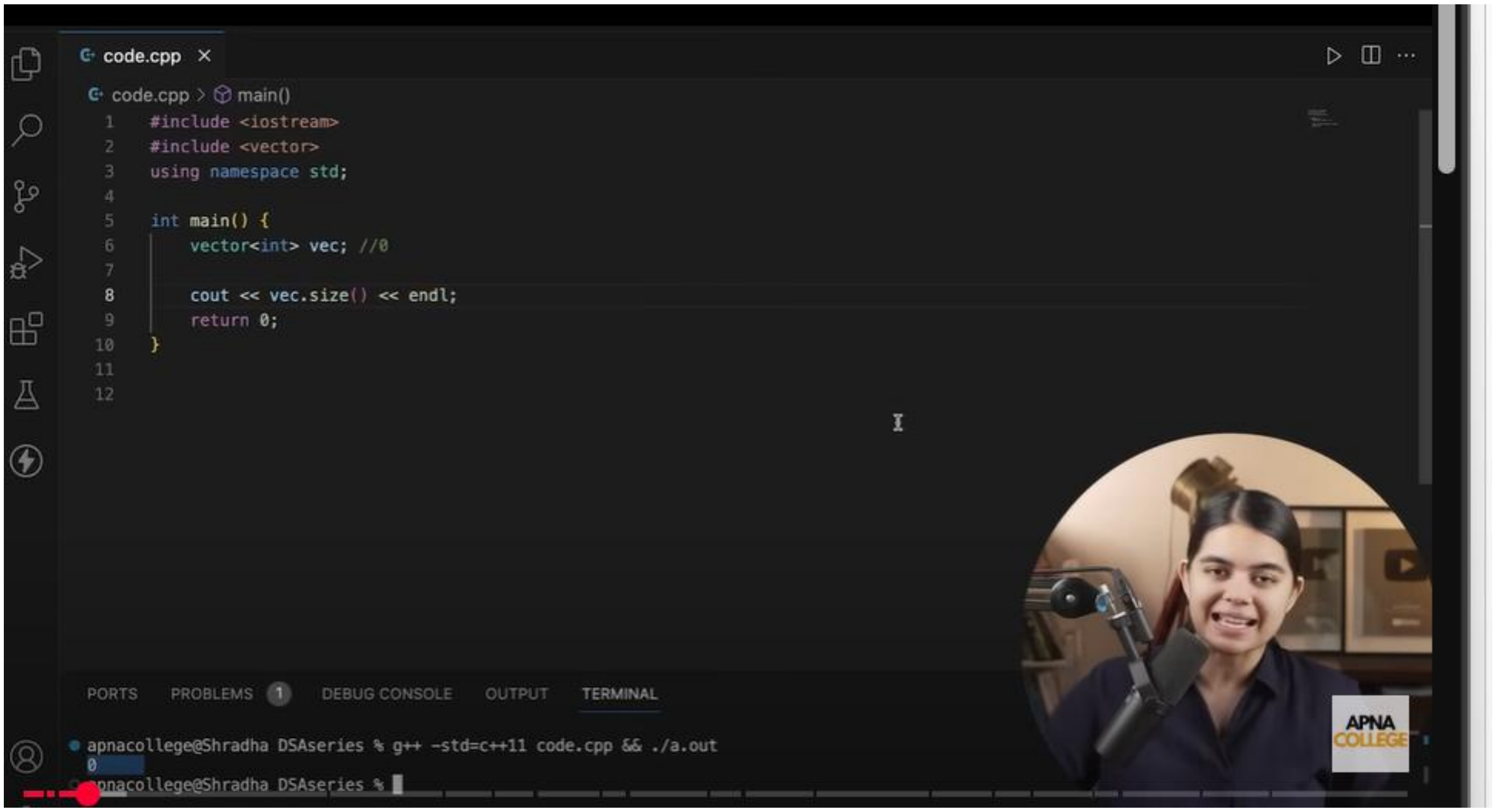
C++ STL Complete Tutorial | Standard Template Library - One Shot

Vector:



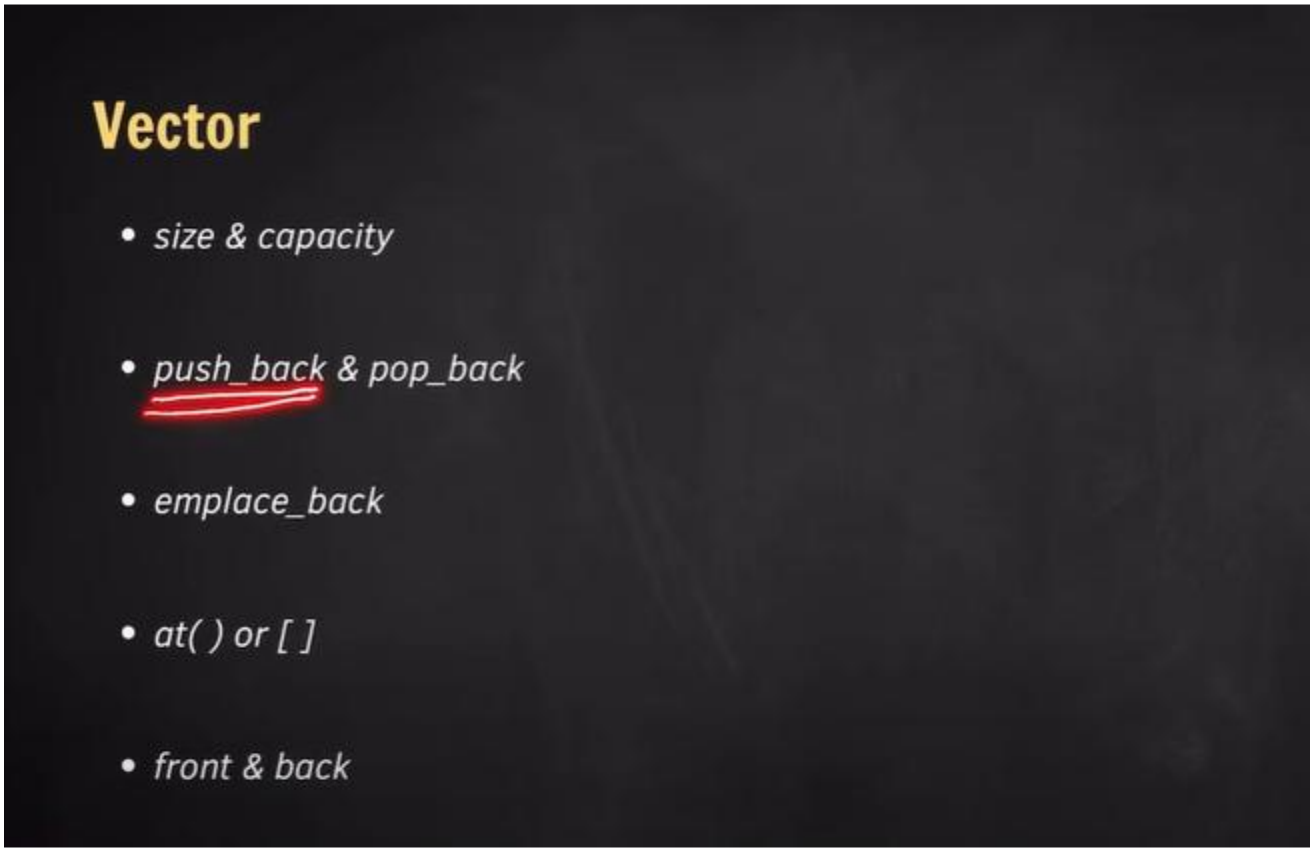
Vector → dynamic/resize

```
vector<int> vec;

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

vector<int> vec(3, 10);

vector<int> vec2(vec1);
```



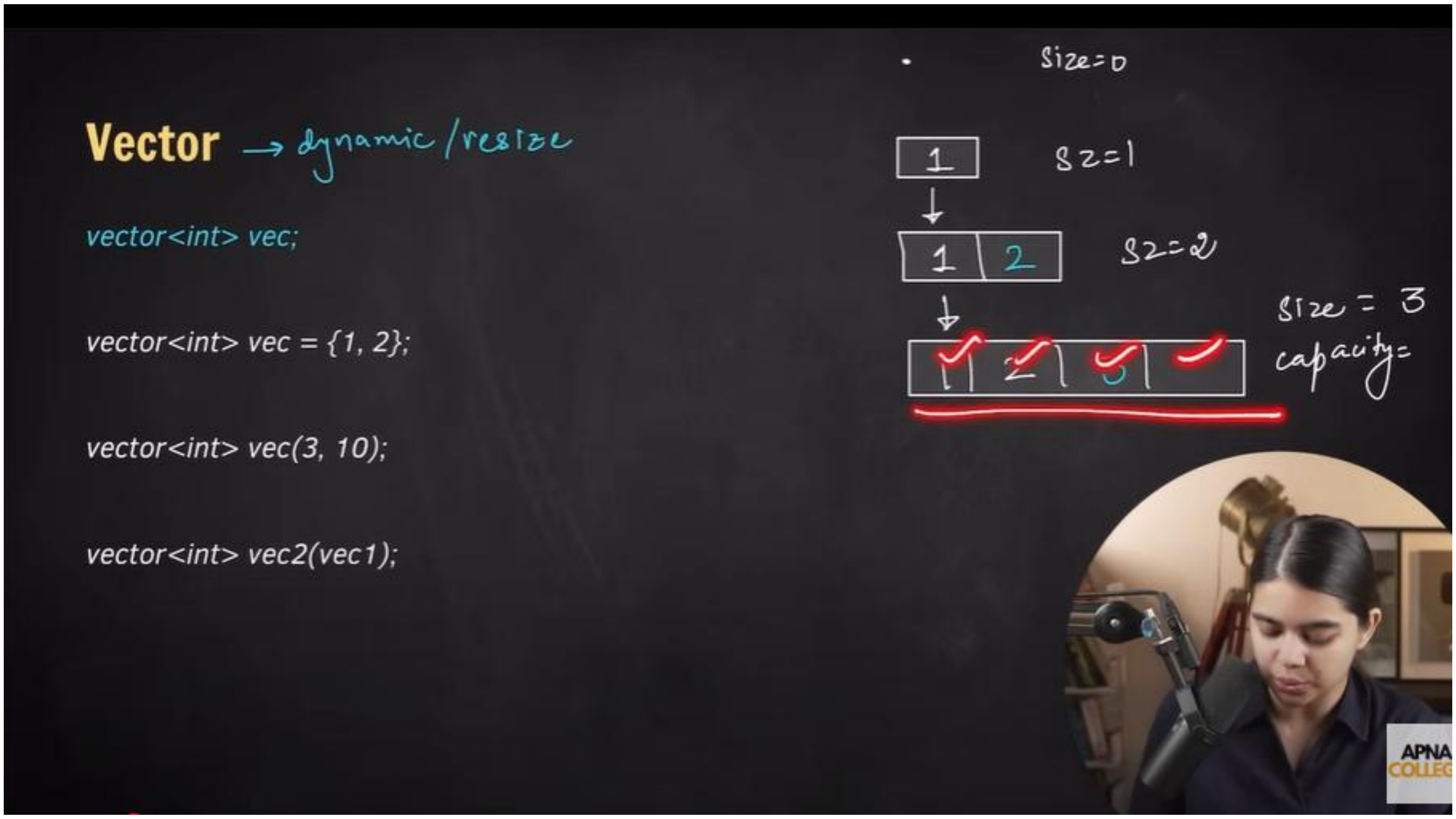
```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vec; //0

    cout << vec.size() << endl;
    return 0;
}
```

```
apnacollege@Shradha DSAseries % g++ -std=c++11 code.cpp && ./a.out
0
apnacollege@Shradha DSAseries %
```



Vector

- *size & capacity*
- *push_back & pop_back*
- *emplace_back*
- *at( ) or [ ]*
- *front & back*



Vector → dynamic/resize

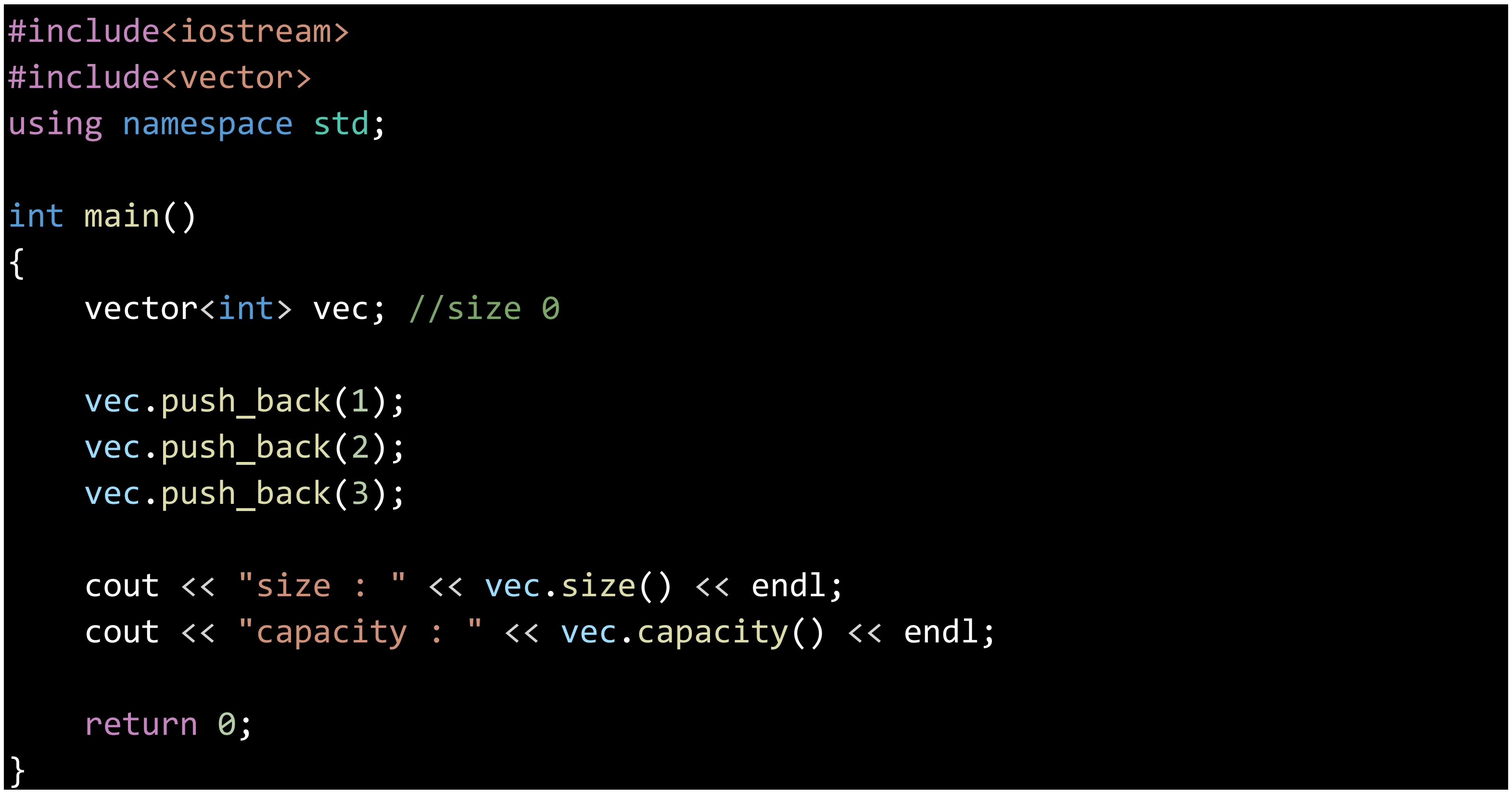
vector<int> vec;

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

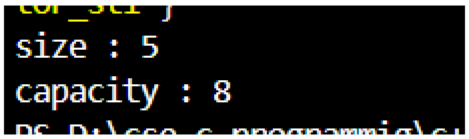
vector<int> vec(3, 10);

vector<int> vec2(vec1);

Code:

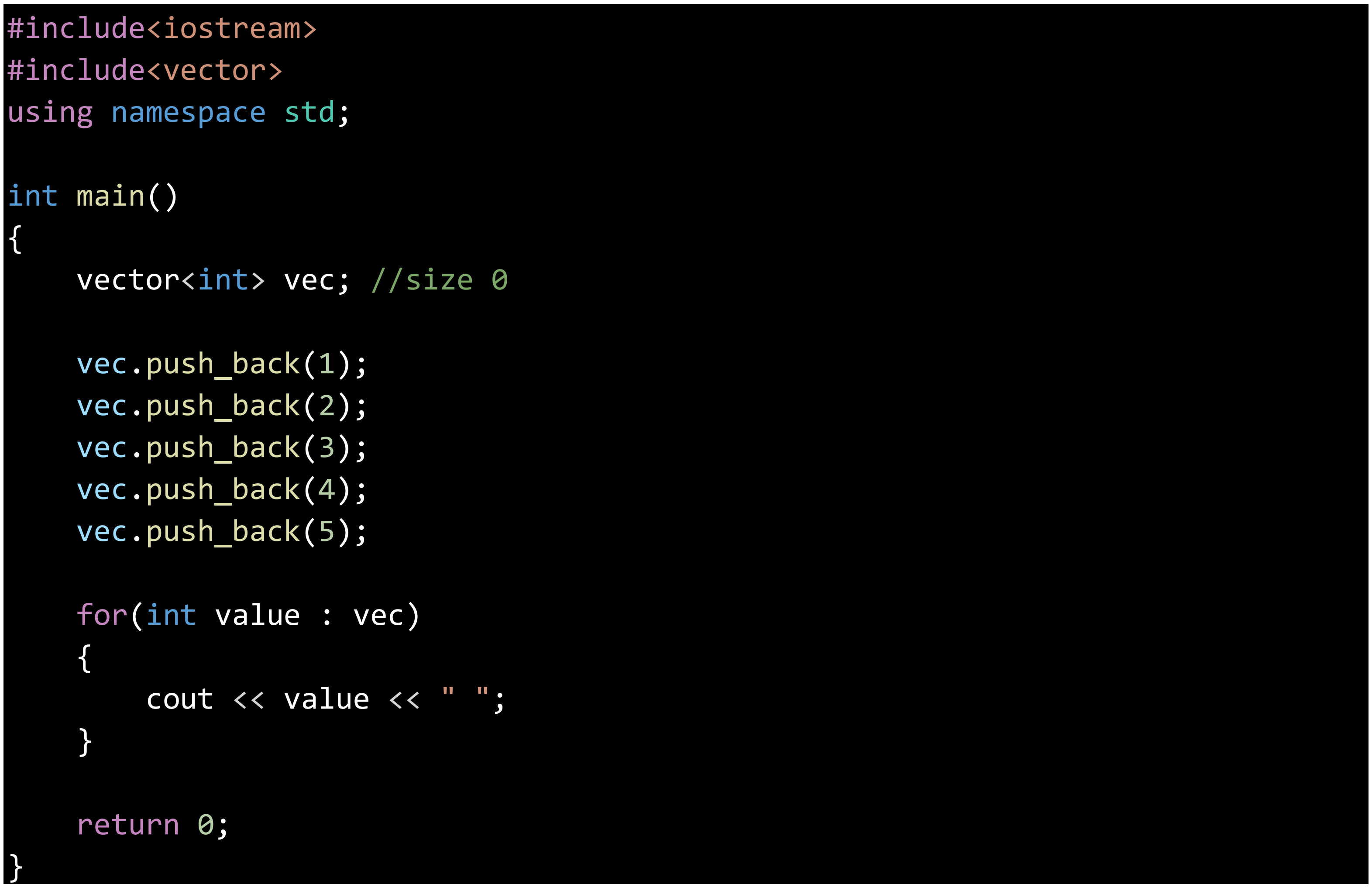


```
#include<iostream>
#include<vector>
using namespace std;

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 : 

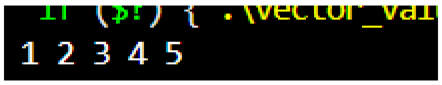
```
size : 5
capacity : 8
```

Print value code:



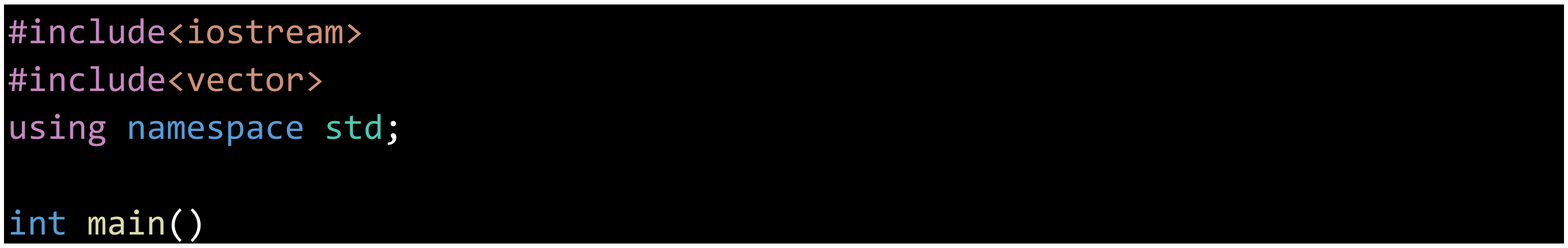
```cpp
#include<iostream>
#include<vector>
using namespace std;

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:



```
if ($?) { .\vector_val
1 2 3 4 5
```

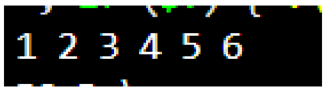
Emplace_back code :



```cpp
#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:

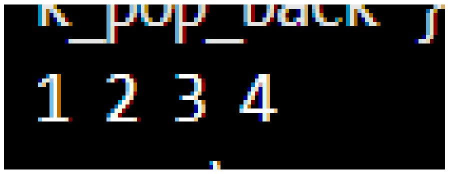


```
1 2 3 4 5 6
```

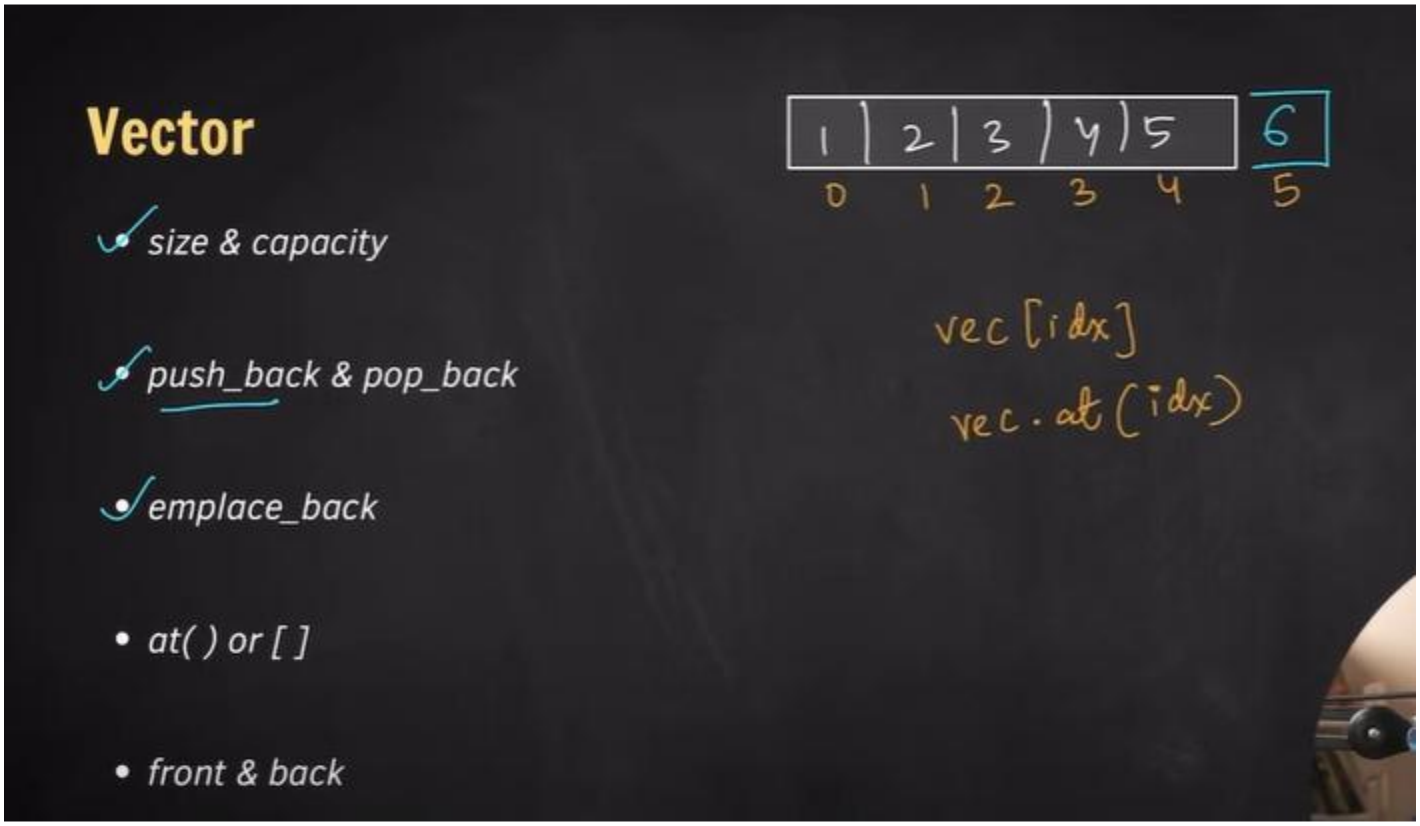
Push_back,,,pop_back code:

```
#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:



```
k_pop_back
1 2 3 4
```



Vector

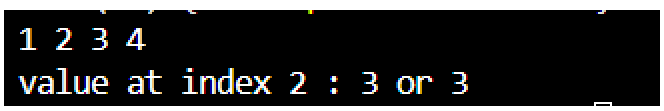
- ✓ size & capacity
- ✓ push_back & pop_back
- ✓ emplace_back
- at( ) or [ ]
- front & back

At() or [] code:

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

```cpp
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:



```
1 2 3 4
value at index 2 : 3 or 3
```

Front ,,,,,back code:

```cpp
#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:

```
1 2 3 4
Front value : 1
Back value : 4
```

Vector initialize:

```cpp
#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 :

```
1 2 3 4
```

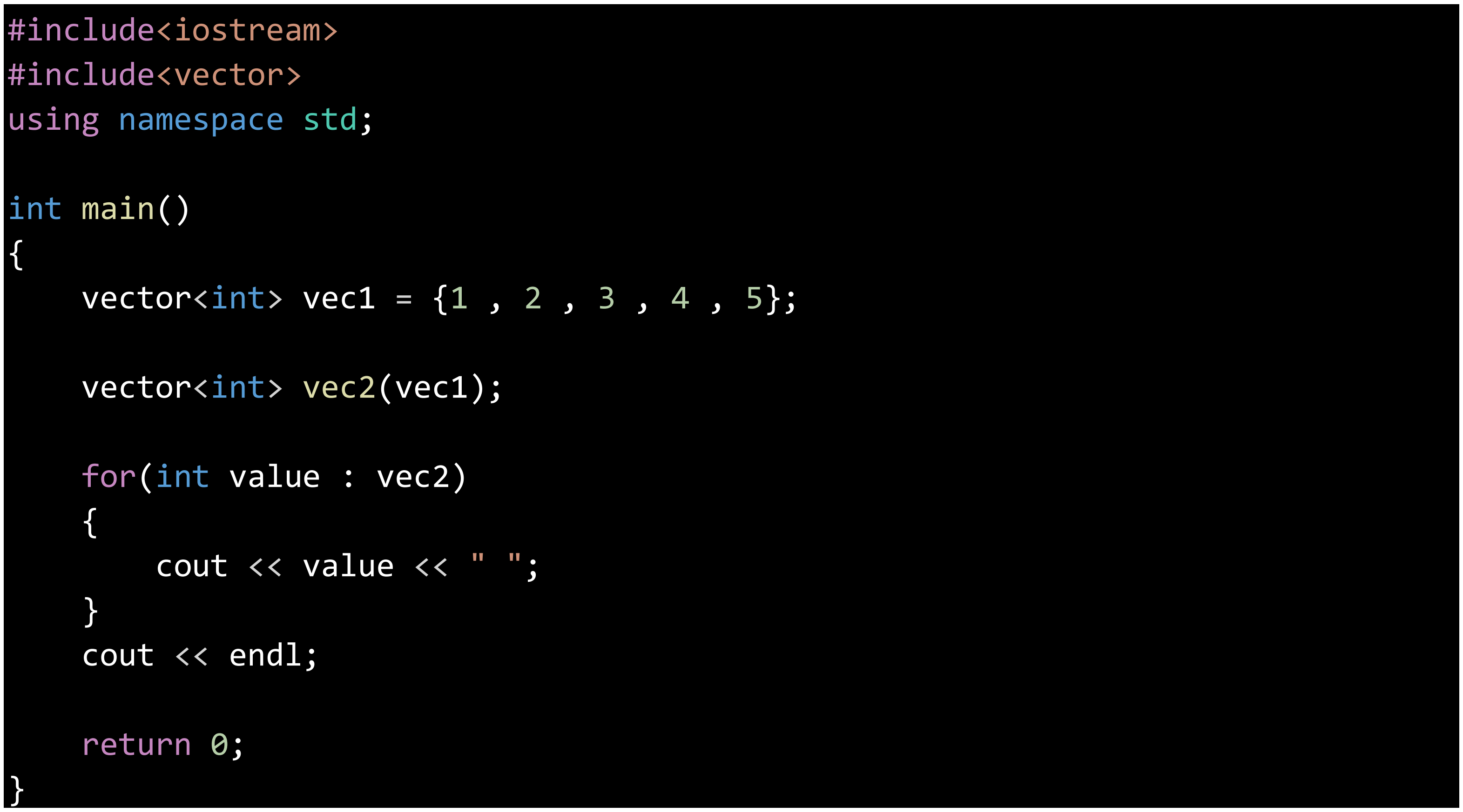
Vector initialize same value code:

```
#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:

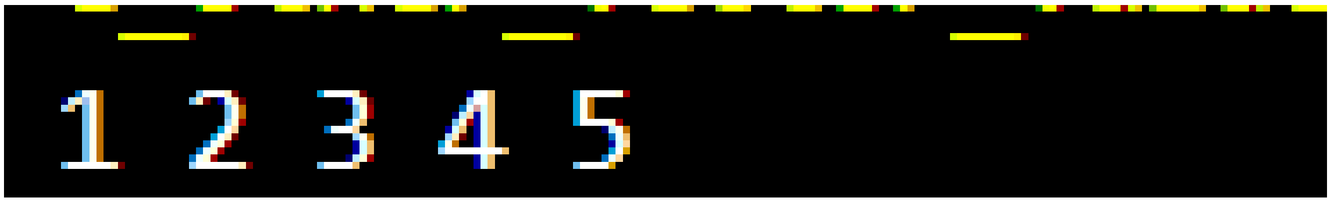
```
10 10 10 10 10 10
```

Vector initialize other vector value code:

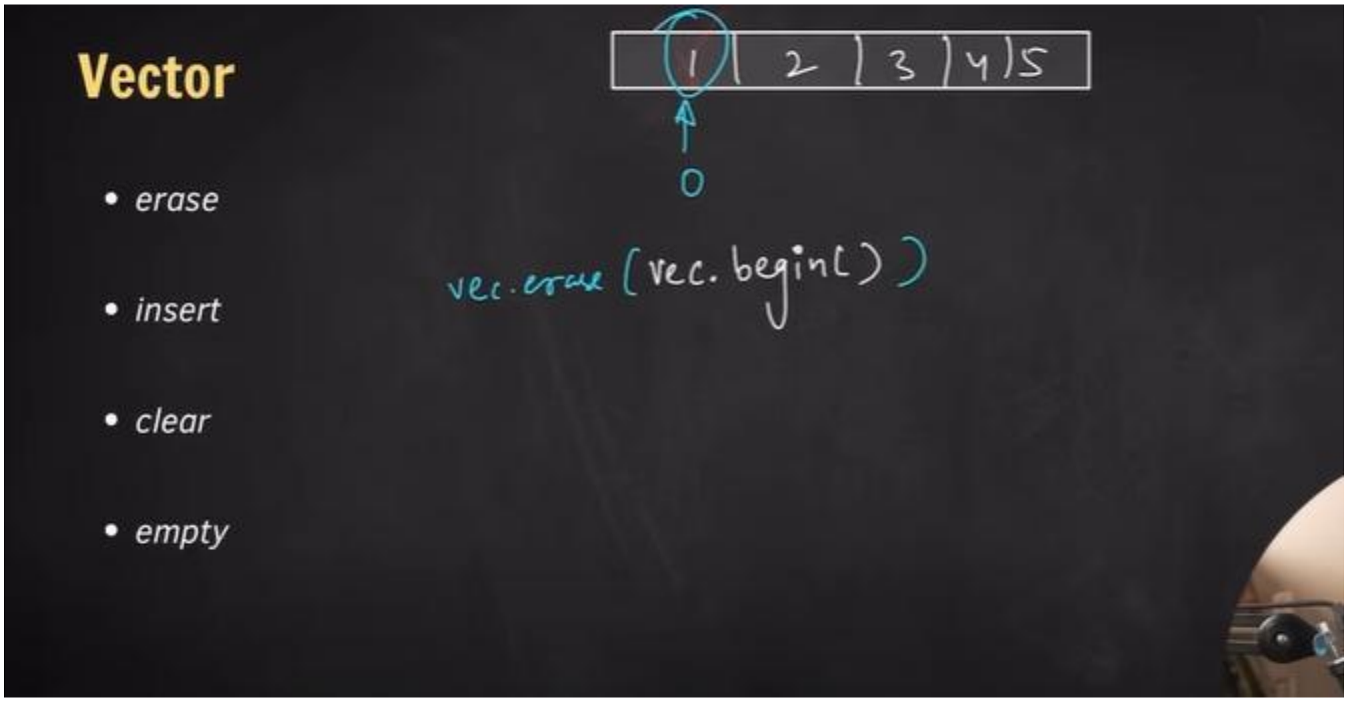


```cpp
#include<iostream>
#include<vector>
using namespace std;

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 :



```
1 2 3 4 5
```


1 2 3 4 5
0
vec.erase(vec.begin())

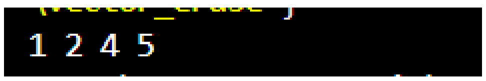
Vector

- erase
- insert
- clear
- empty

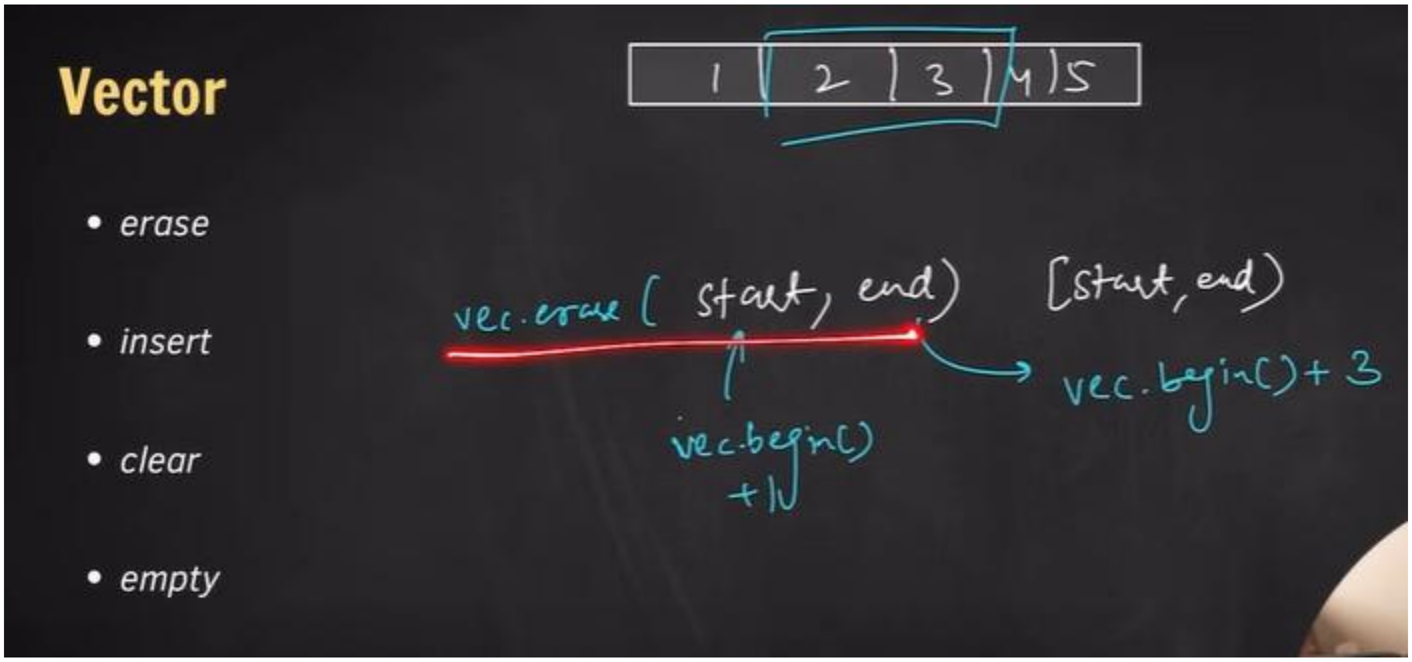
Erase code:

```cpp
#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 :



```
1 2 4 5
```



Vector

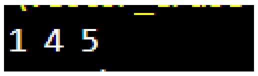
- erase
- insert
- clear
- empty

1
2
3
4
5
vec.erase (start, end)
[start, end)
vec.begin() + 1
vec.begin() + 3

Erase(start , end) code:

```cpp
#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:



```
1 4 5
```

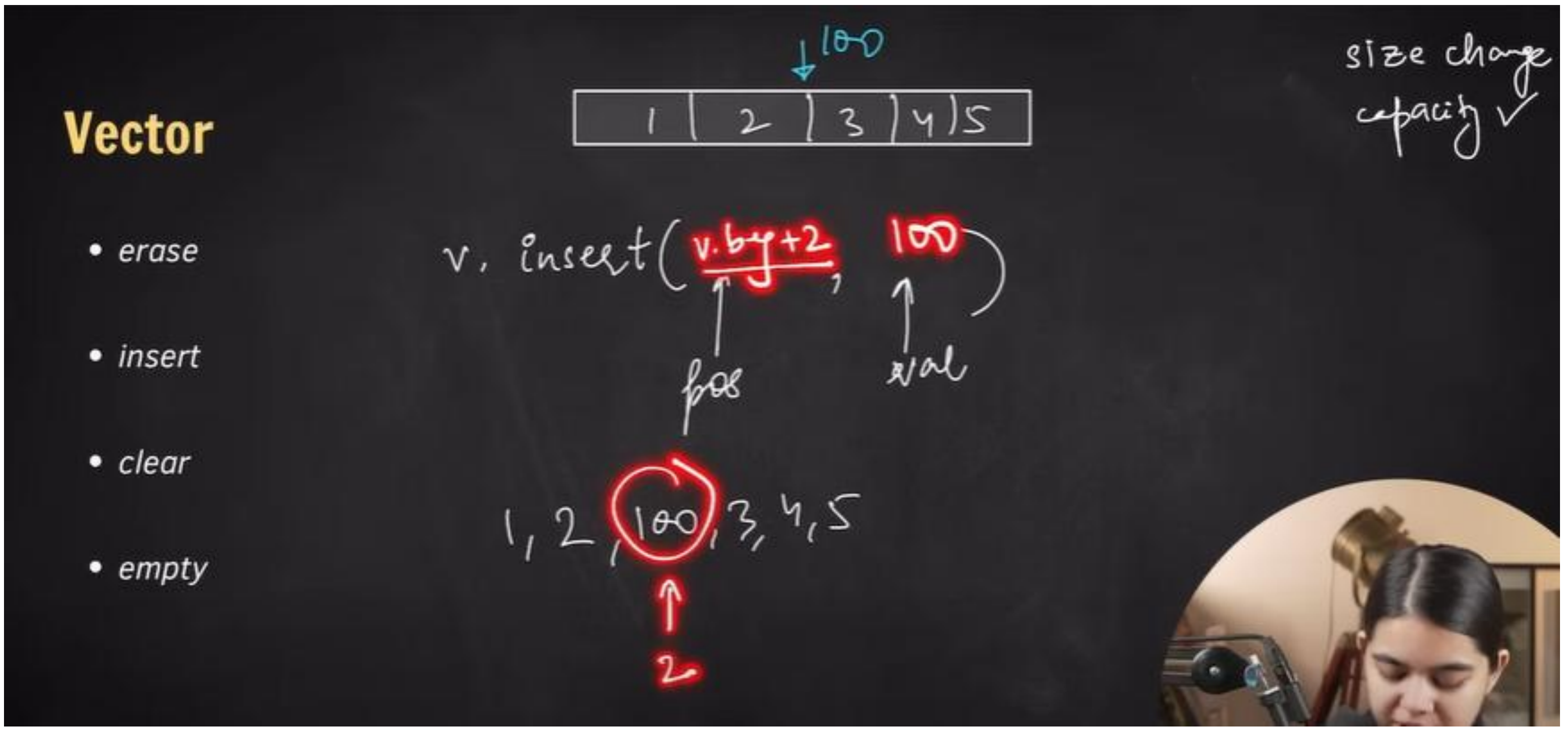
***erase size change kore, but capacity same thake

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:

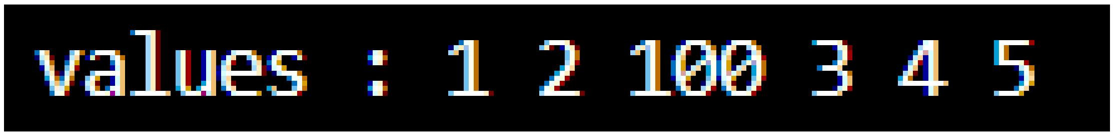
```
size before erasing : 5
capacity before erasing : 5
values : 1 4 5
size after erasing : 3
capacity after erasing : 5
```



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:

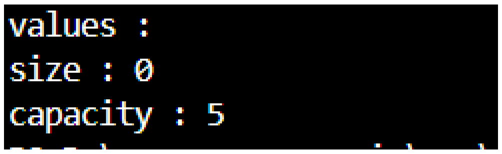


```
values : 1 2 100 3 4 5
```

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:



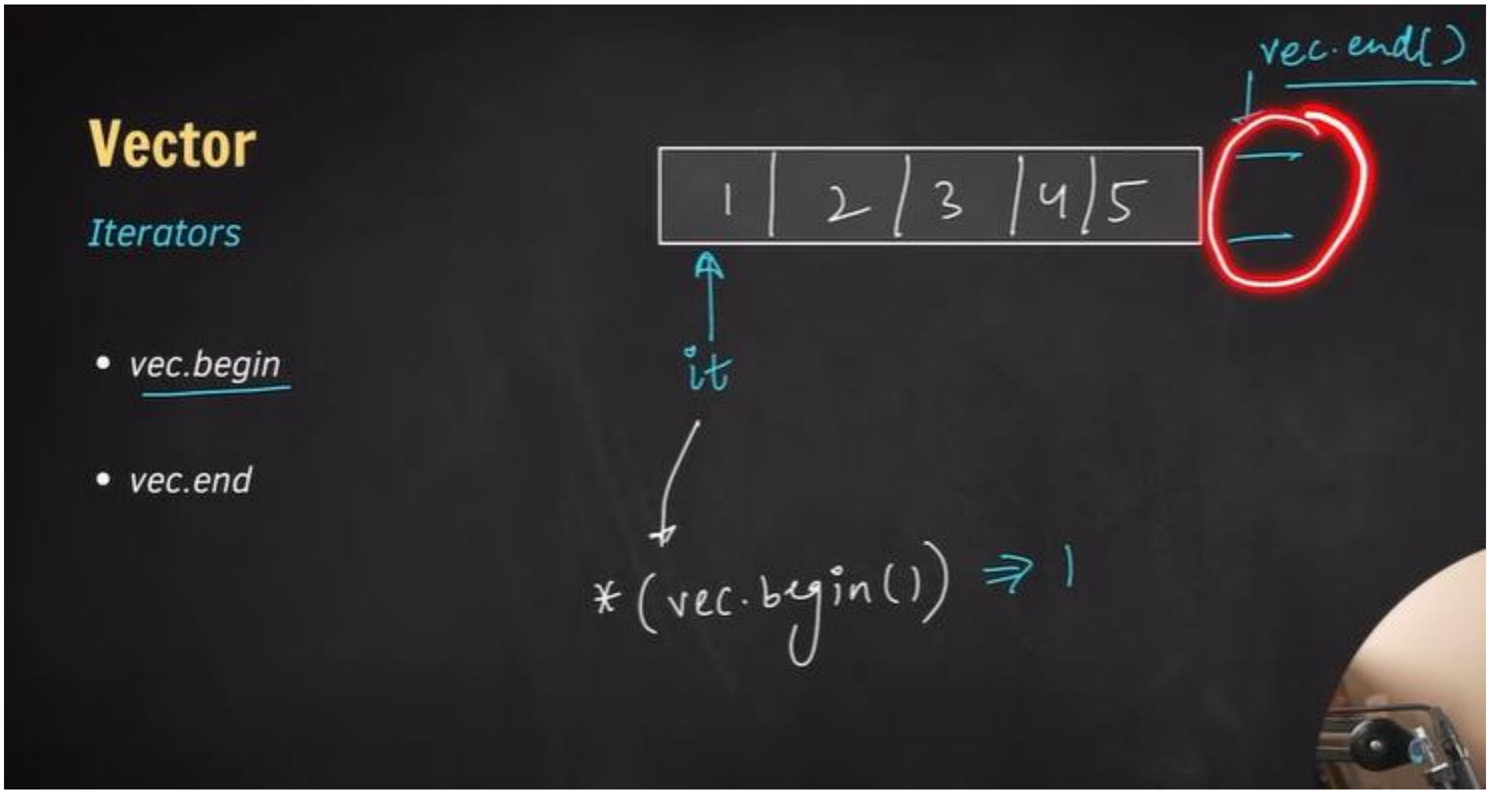
```
values :
size : 0
capacity : 5
```

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:

```
values :
size : 0
capacity : 5
Is empty? : 1
Is empty? : Yes
```



Vector

Iterators

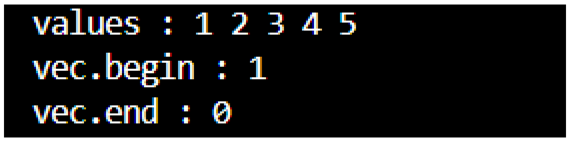
- vec.begin
- vec.end

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

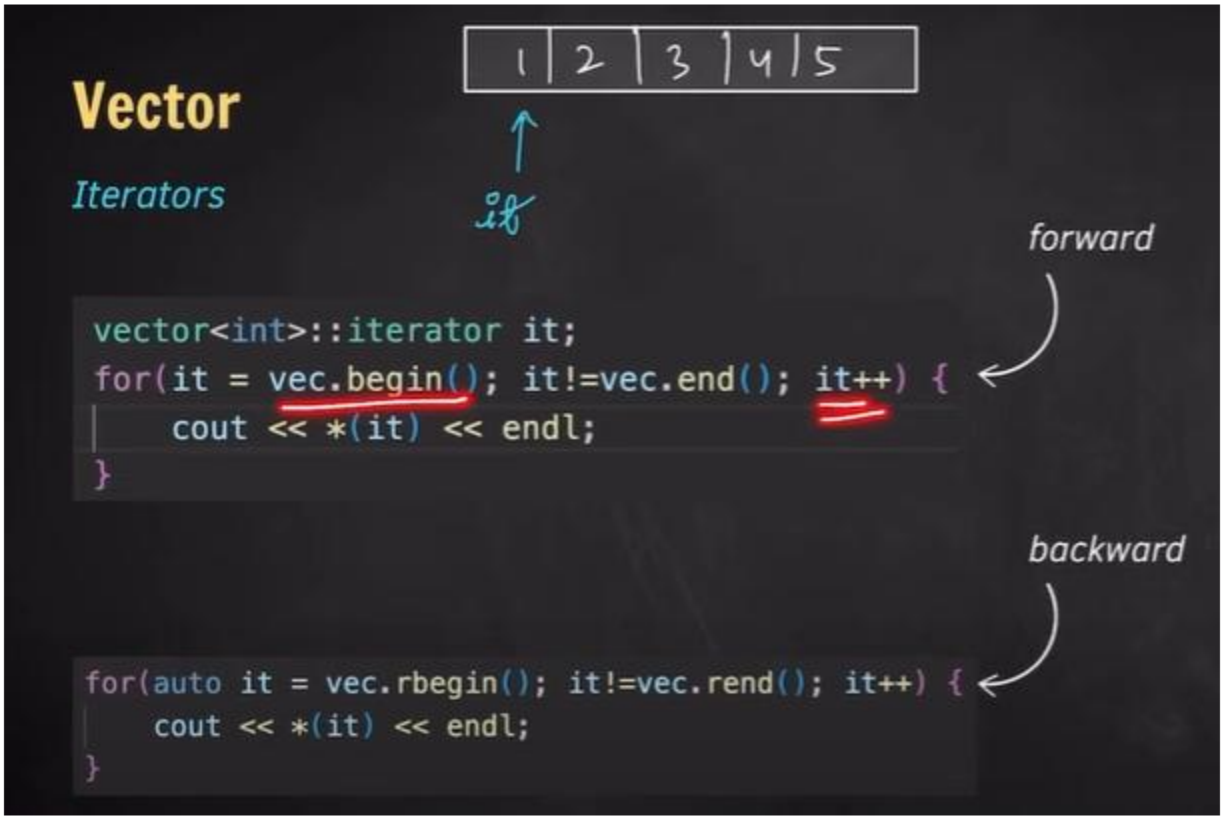
```
#include<iostream>
#include<vector>
using namespace std;

```

```
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:



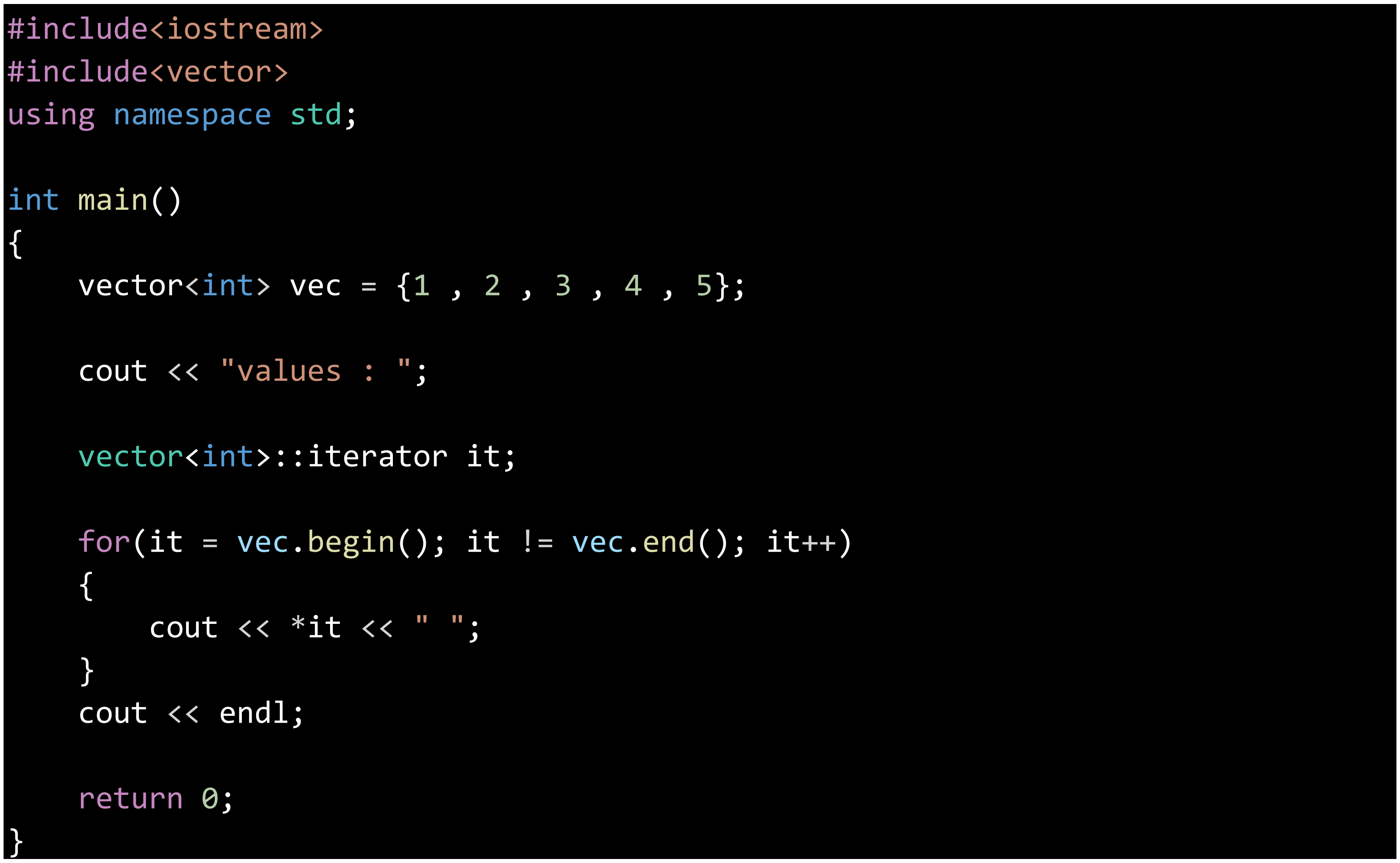
```
values : 1 2 3 4 5
vec.begin : 1
vec.end : 0
```



```
vector<int>::iterator it;
for(it = vec.begin(); it!=vec.end(); it++) {
    cout << *(it) << endl;
}
```

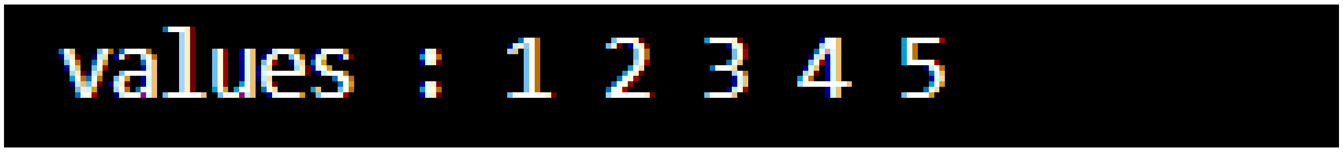
```
for(auto it = vec.rbegin(); it!=vec.rend(); it++) {
    cout << *(it) << endl;
}
```

Iterator code:



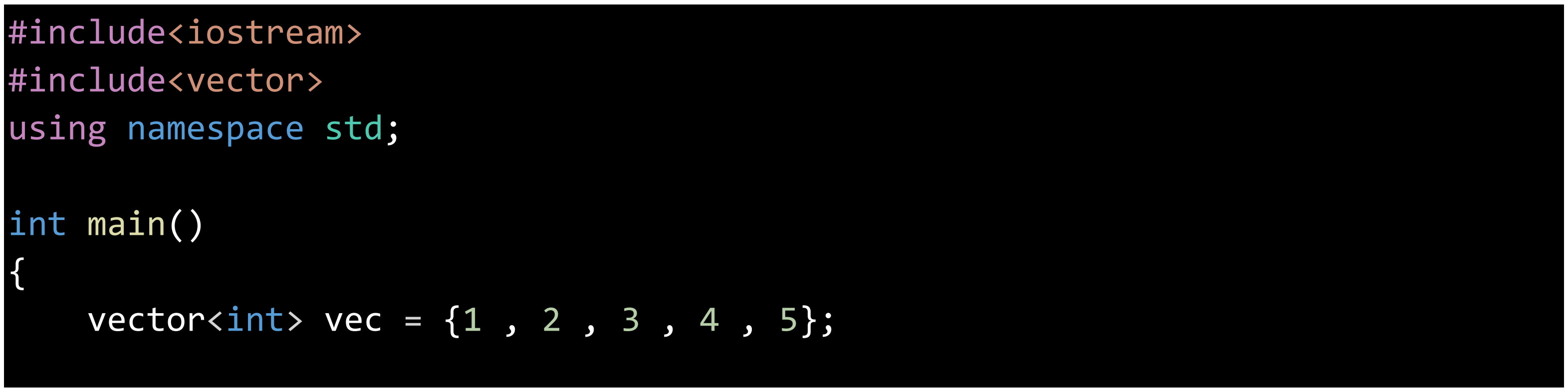
```
#include<iostream>
#include<vector>
using namespace std;

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:

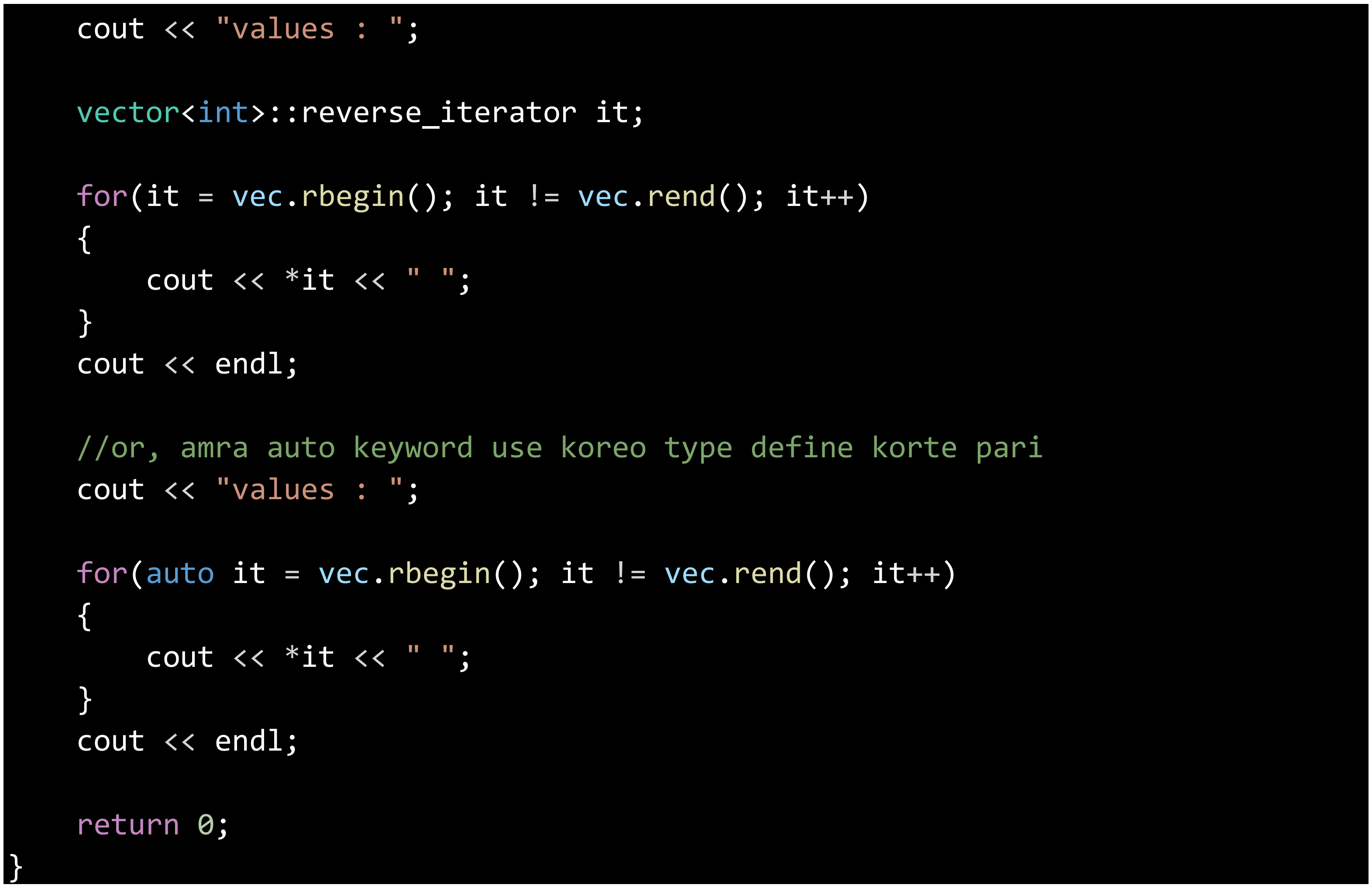


```
values : 1 2 3 4 5
```

Reverse iterator code:

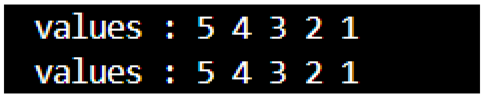


```
#include<iostream>
#include<vector>
using namespace std;

int main()
{
    vector<int> vec = {1 , 2 , 3 , 4 , 5};
```



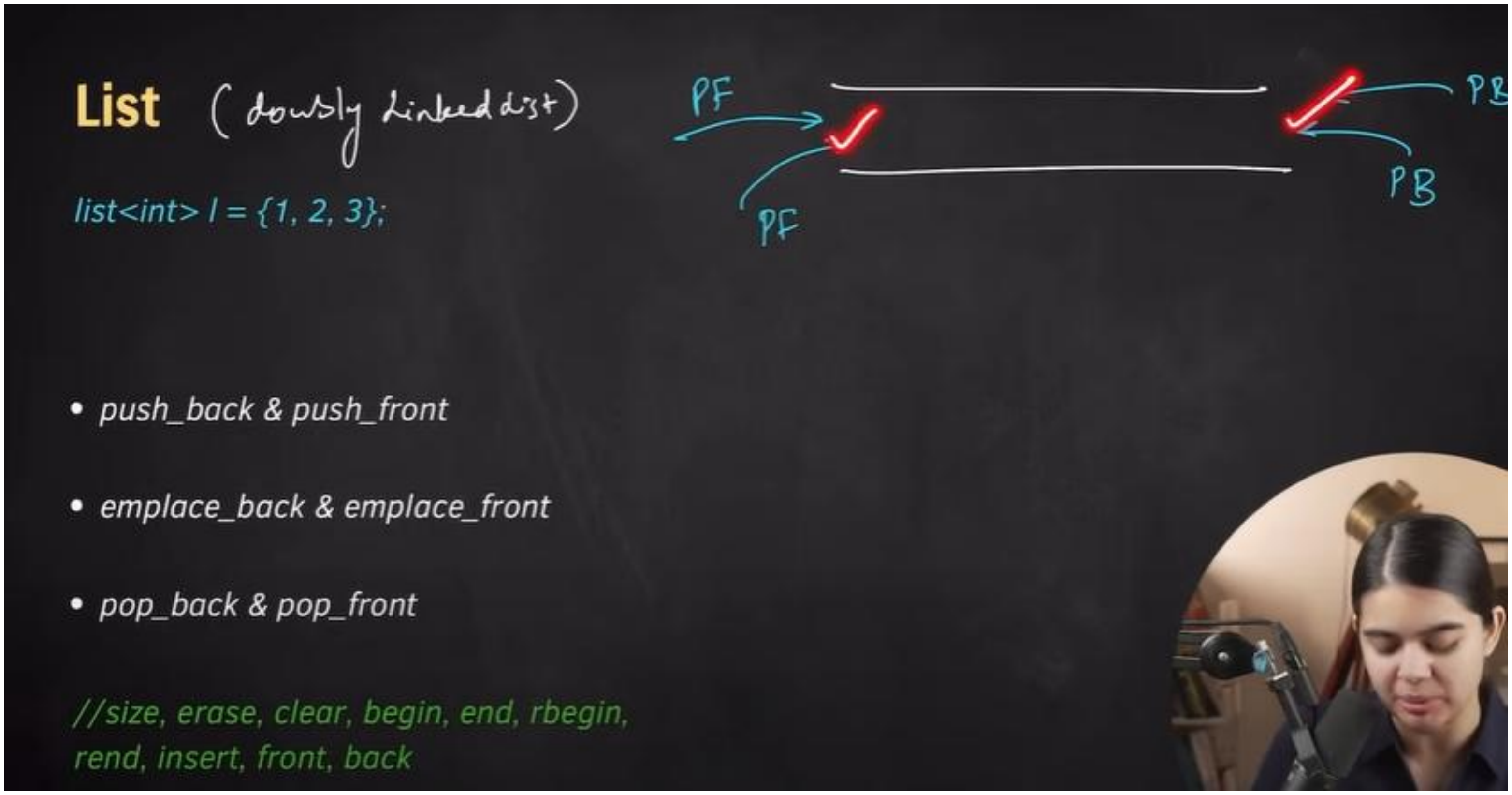
```
    cout << "values : ";

    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 :



```
values : 5 4 3 2 1
values : 5 4 3 2 1
```

List



List (doubly linked list)

list<int> l = {1, 2, 3};

- push_back & push_front
- emplace_back & emplace_front
- pop_back & pop_front

//size, erase, clear, begin, end, rbegin,
rend, insert, front, back

Push_back, push_front,

Emplace_back, emplace_front,

Pop_back, pop_front

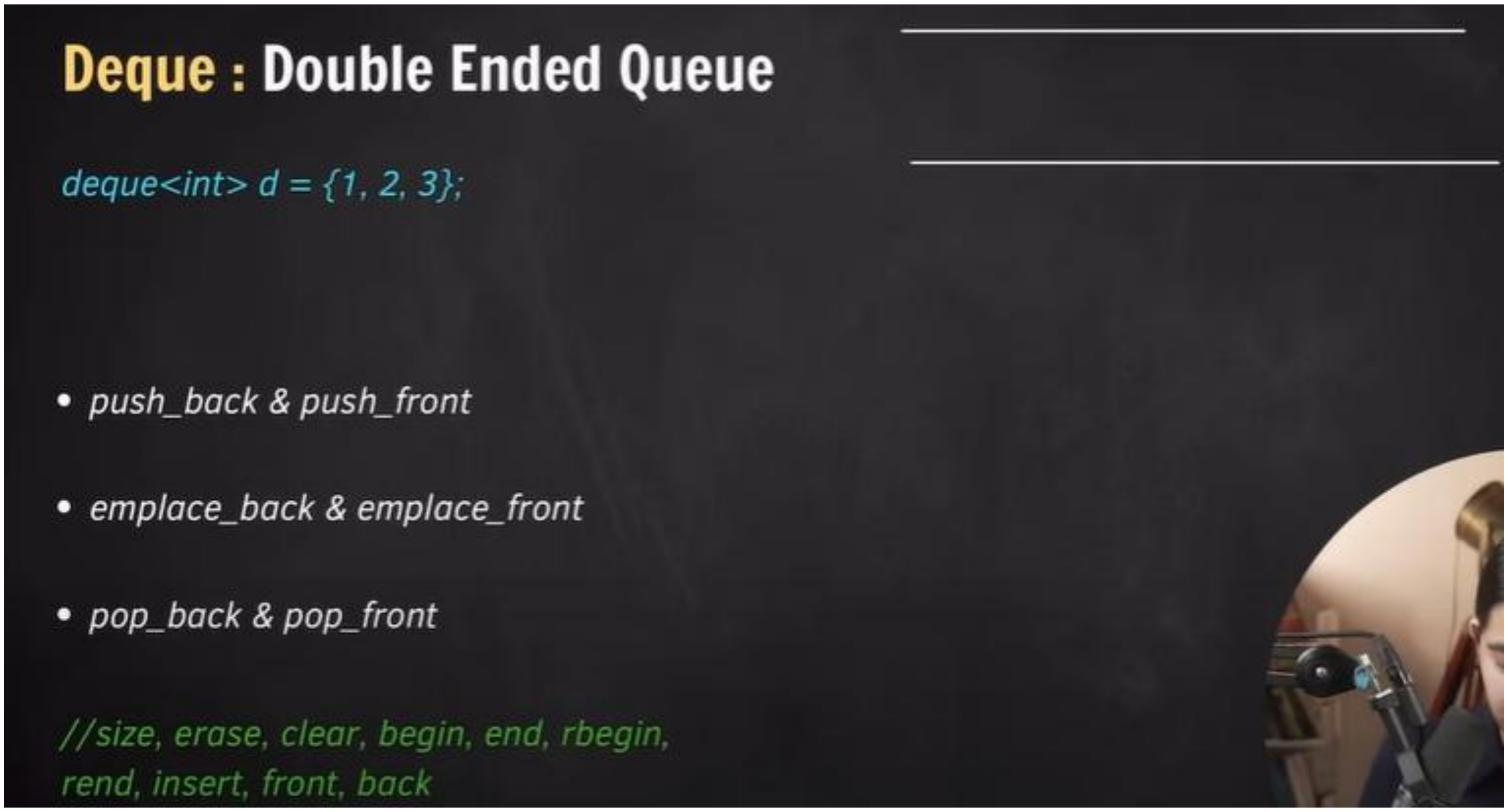
Code:

```
#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:

```
List 1 elements : 1 2 3 4 5
List 2 elements before poping: 5 3 1 2
List 2 elements after poping: 3 1
```

Deque

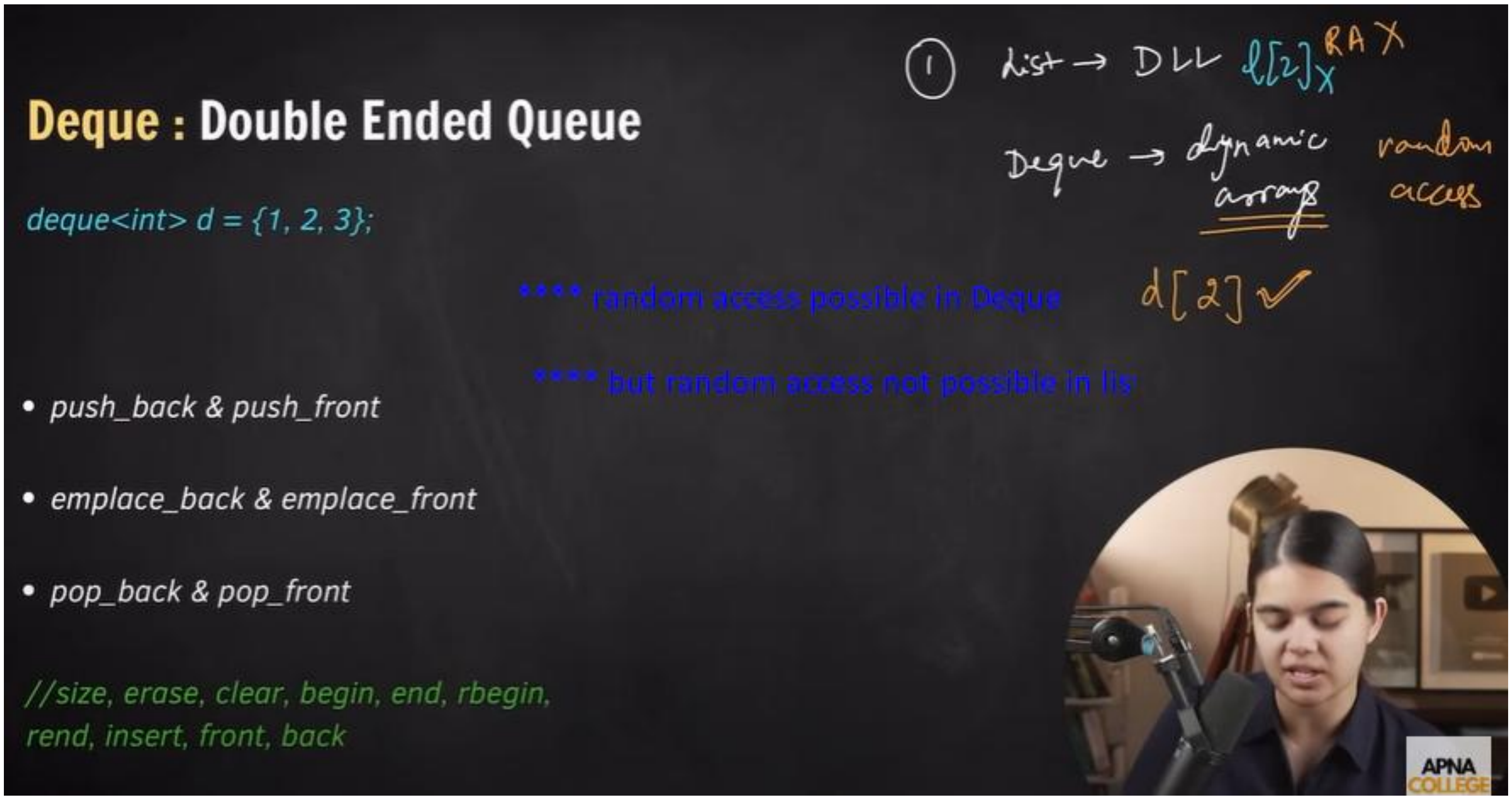


Deque : Double Ended Queue

```
deque<int> d = {1, 2, 3};
```

- push_back & push_front
- emplace_back & emplace_front
- pop_back & pop_front

```
//size, erase, clear, begin, end, rbegin,
rend, insert, front, back
```



Deque : Double Ended Queue

```
deque<int> d = {1, 2, 3};
```

- push_back & push_front
- emplace_back & emplace_front
- pop_back & pop_front

```
//size, erase, clear, begin, end, rbegin,
rend, insert, front, back
```

① list → DLL l[2] ✗ RA ✗

Deque → dynamic arrays   random access

d[2] ✓

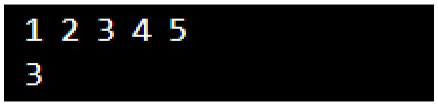
**** random access possible in Deque

**** but random access not possible in lis

Deque random access code:

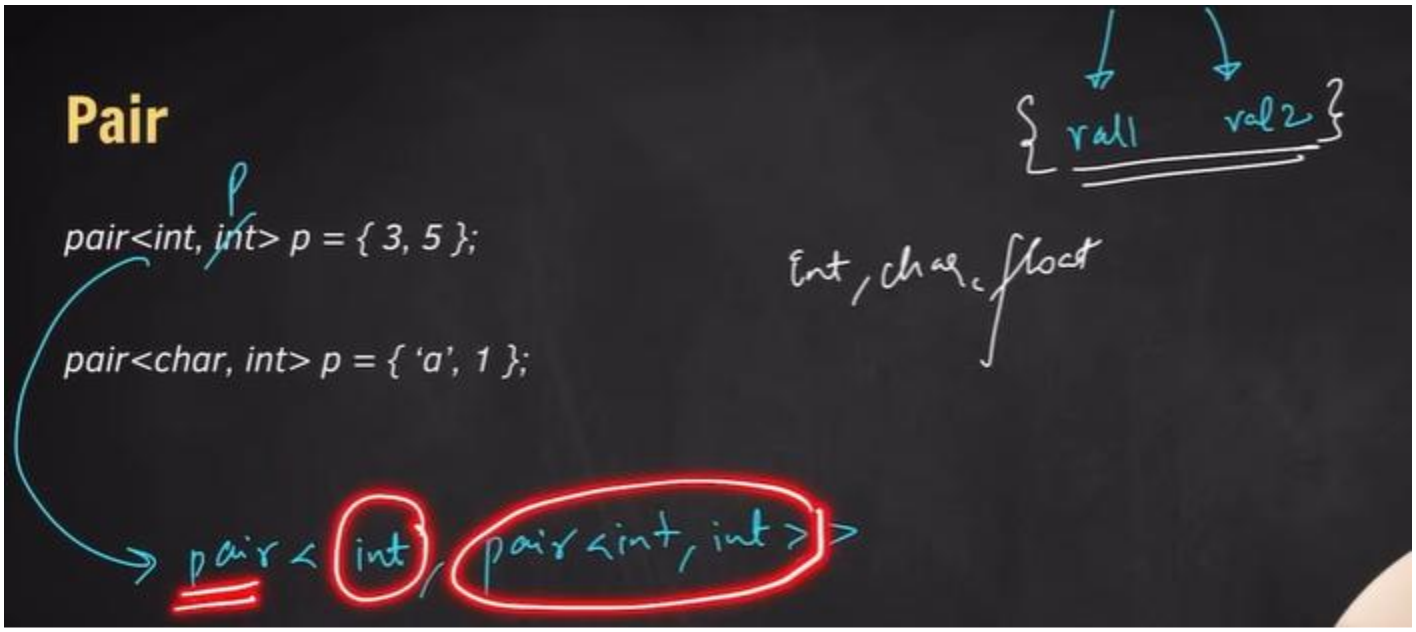
```
#include<iostream>
#include<deque>
using namespace std;

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:



```
1 2 3 4 5
3
```

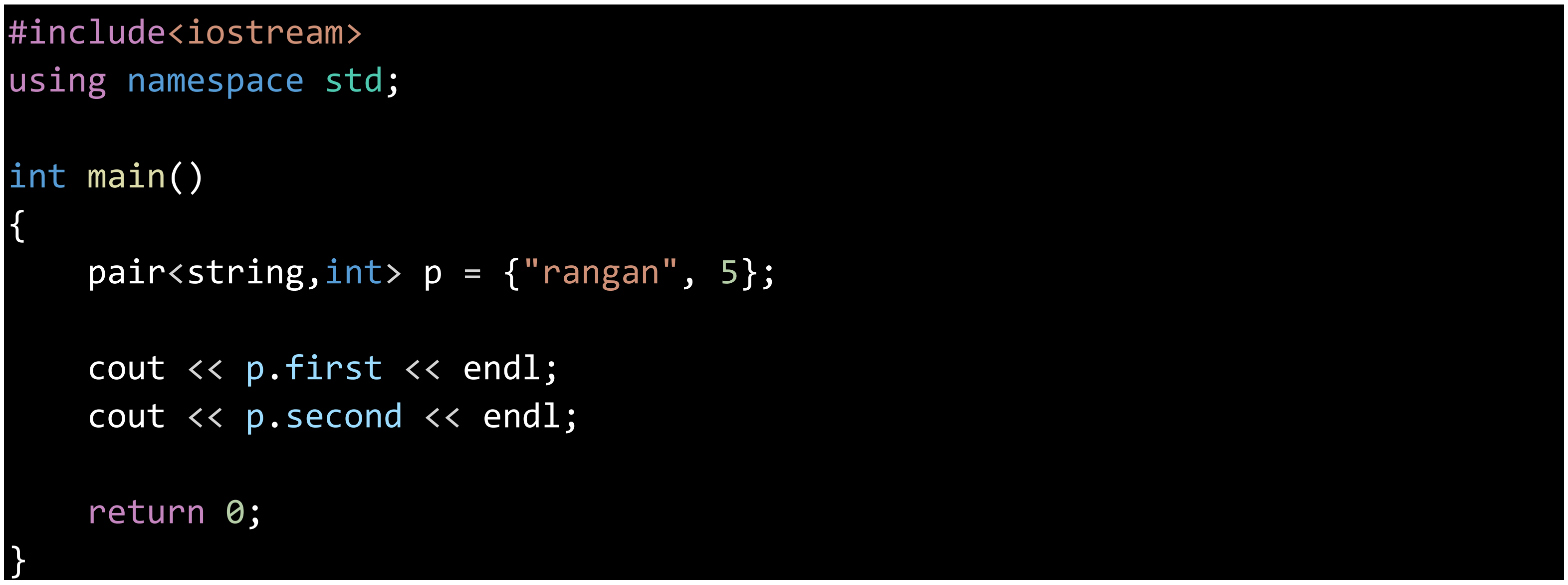
Pair



Pair

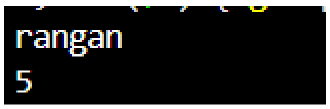
```
pair<int, int> p = { 3, 5 };

pair<char, int> p = { 'a', 1 };
```

Pair code:



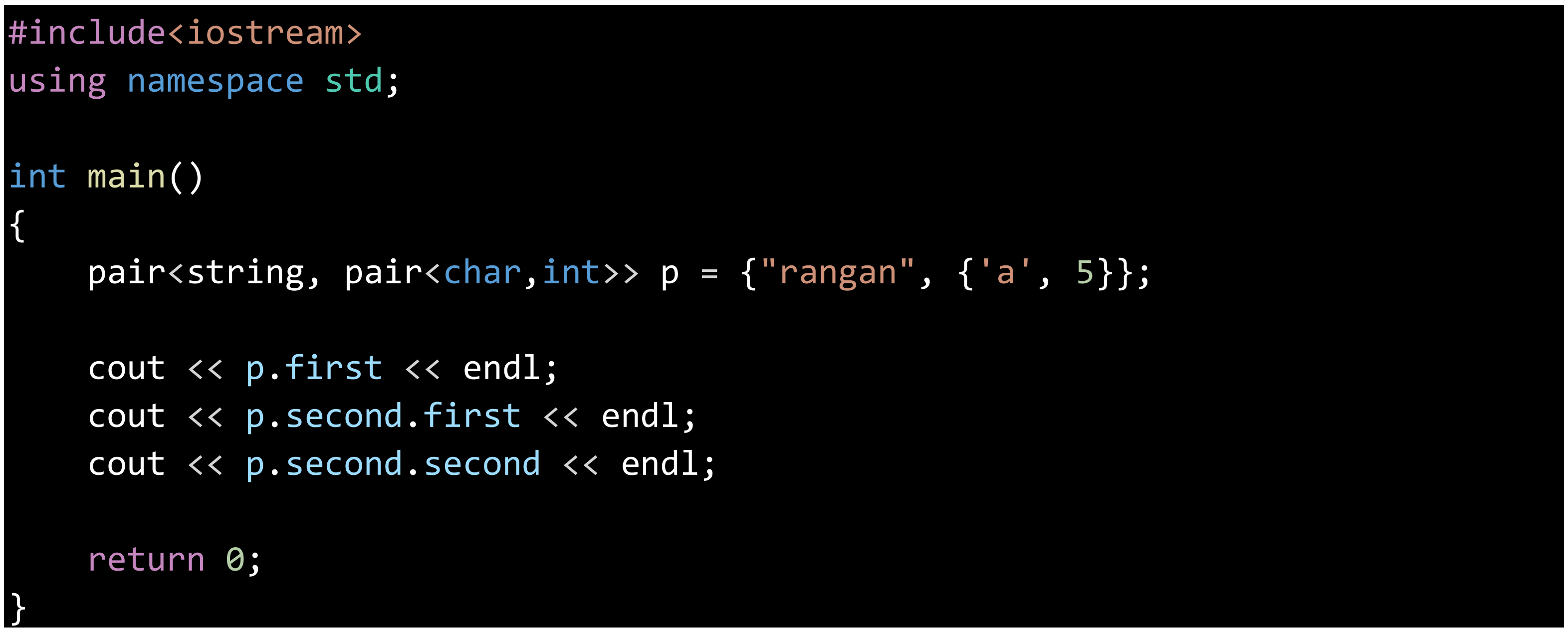
```cpp
#include<iostream>
using namespace std;

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

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

    return 0;
}
```

Result:



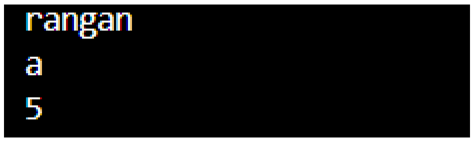
```
rangan
5
```

Pair of pair code:



```cpp
#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:

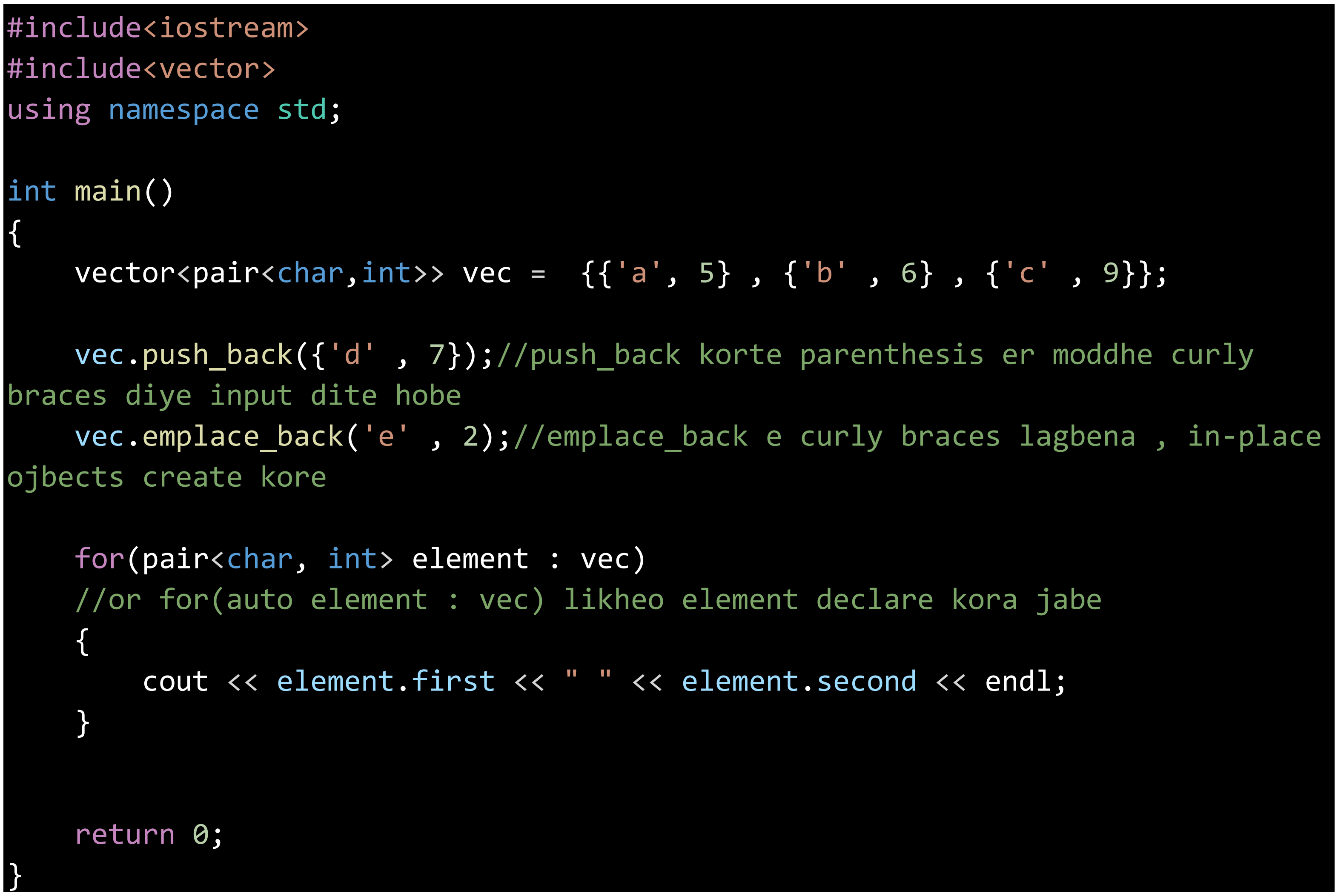


```
rangan
a
5
```

Pair in vector

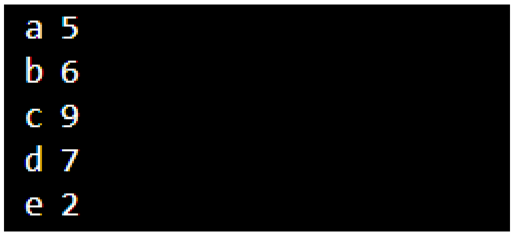
Difference between push_back and emplace_back

Code:



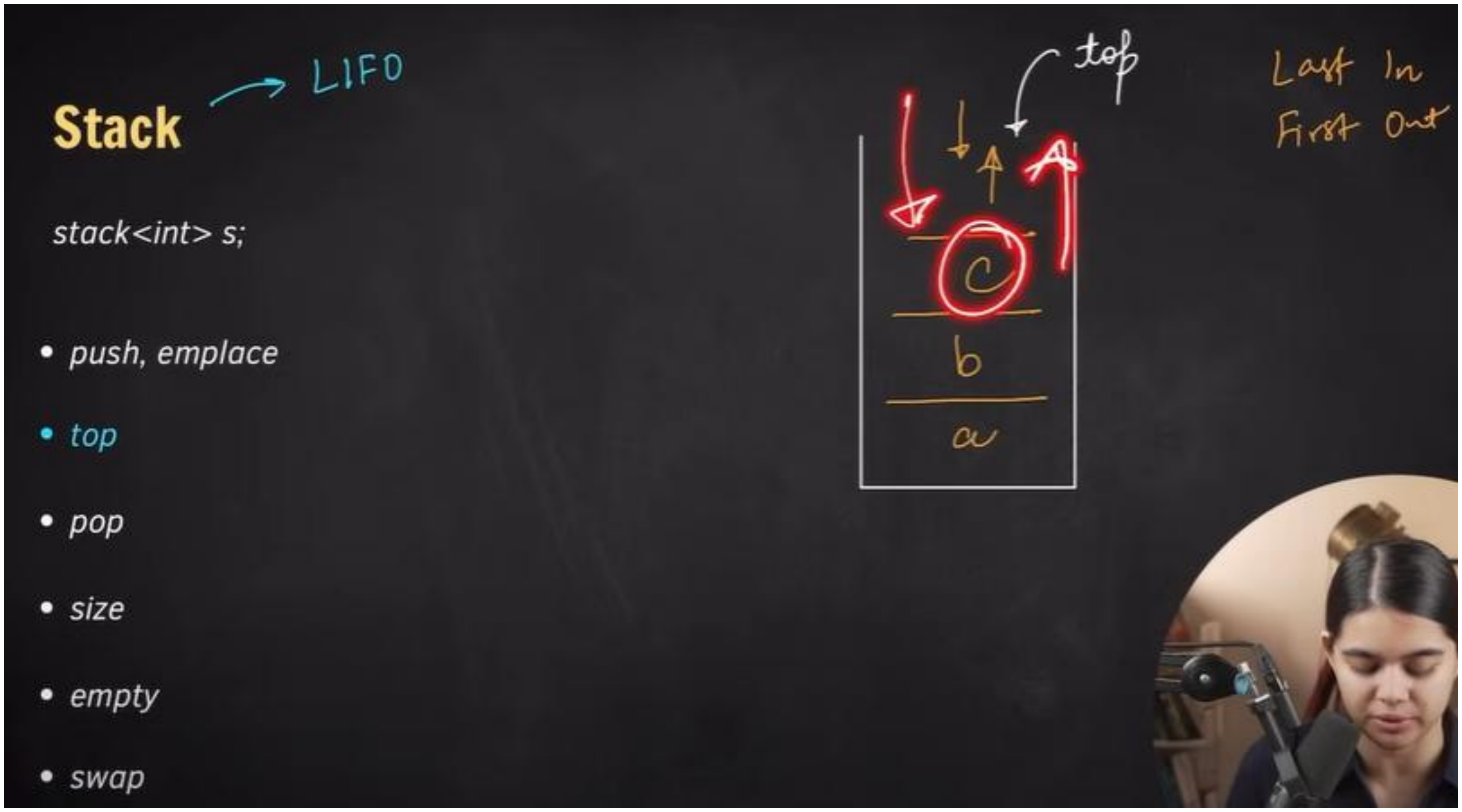
```
#include<iostream>
#include<vector>
using namespace std;

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:



```
a 5
b 6
c 9
d 7
e 2
```

Stack

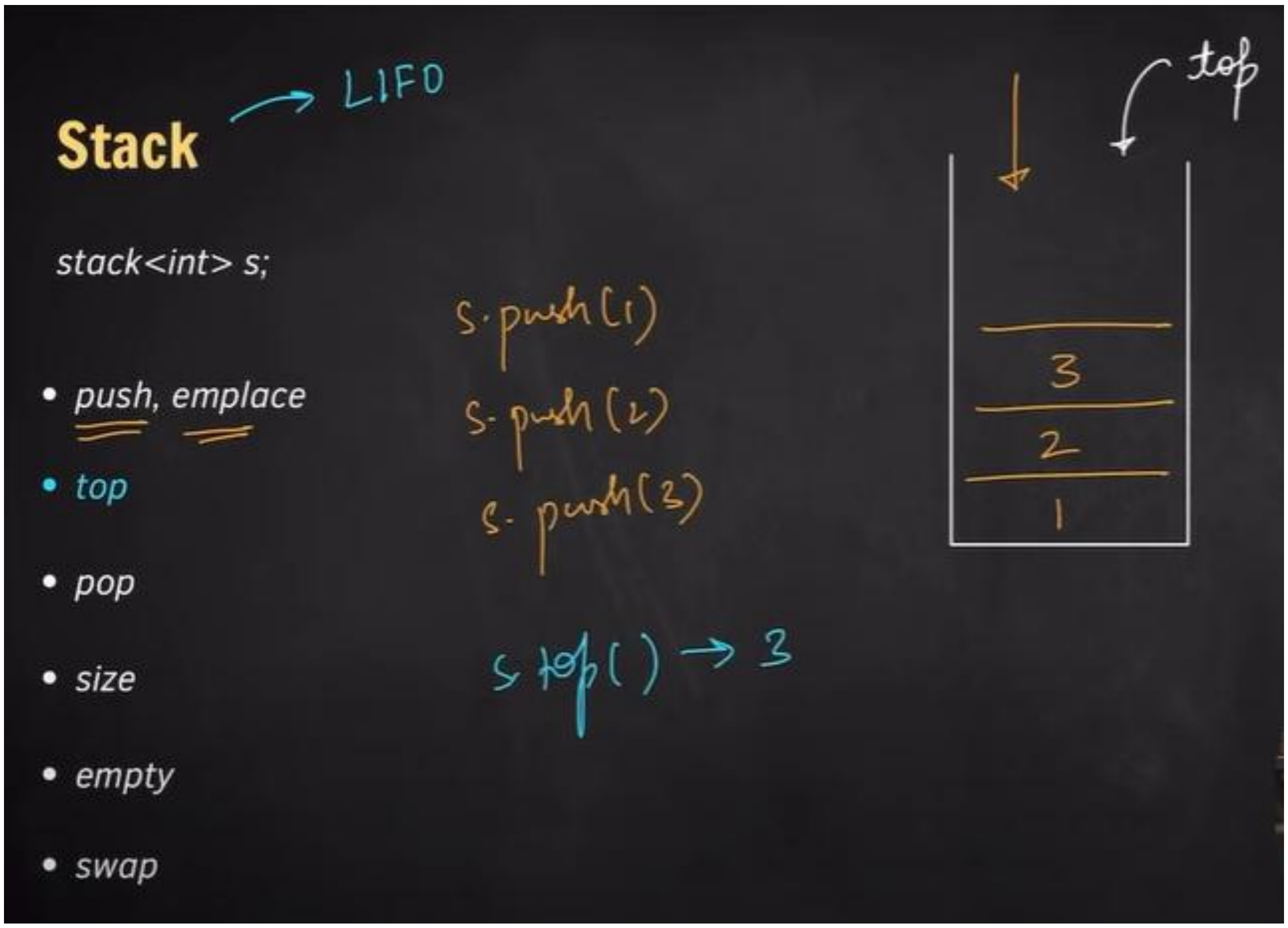


Stack

→ LIFO

stack<int> s;

- push, emplace
- top
- pop
- size
- empty
- swap



Stack

→ LIFO

stack<int> s;

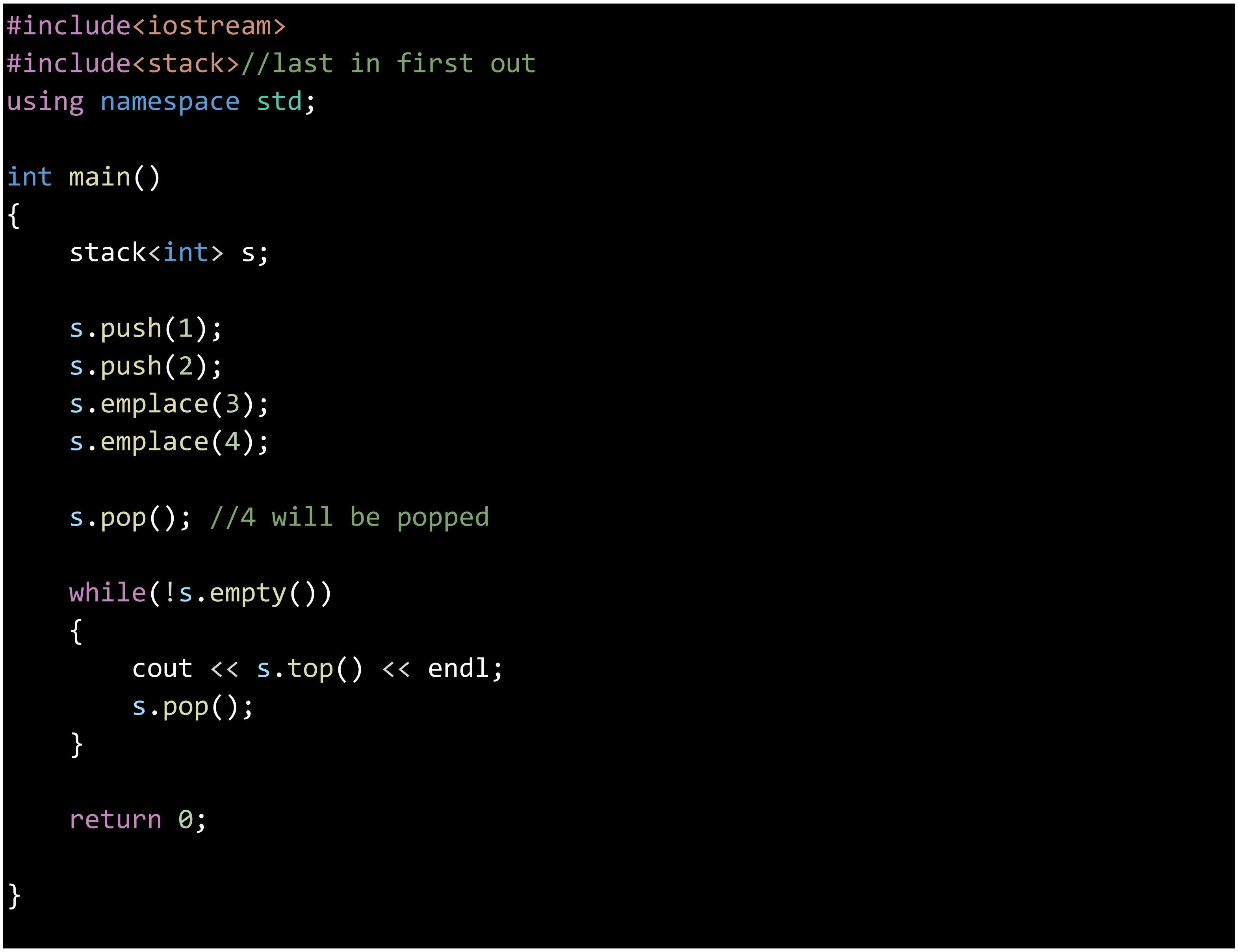
- push, emplace
- top
- pop
- size
- empty
- swap

s.push(1)
s.push(2)
s.push(3)

s.top() → 3

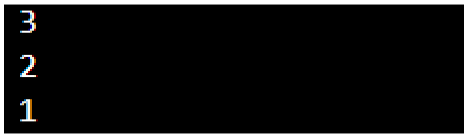
Push(),,,emplace(),,,,pop(),,,,empty()

code:



```cpp
#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:



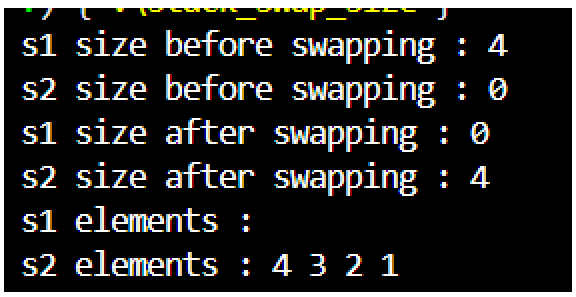
```
3
2
1
```

Swap(),,,,,size()

Code:

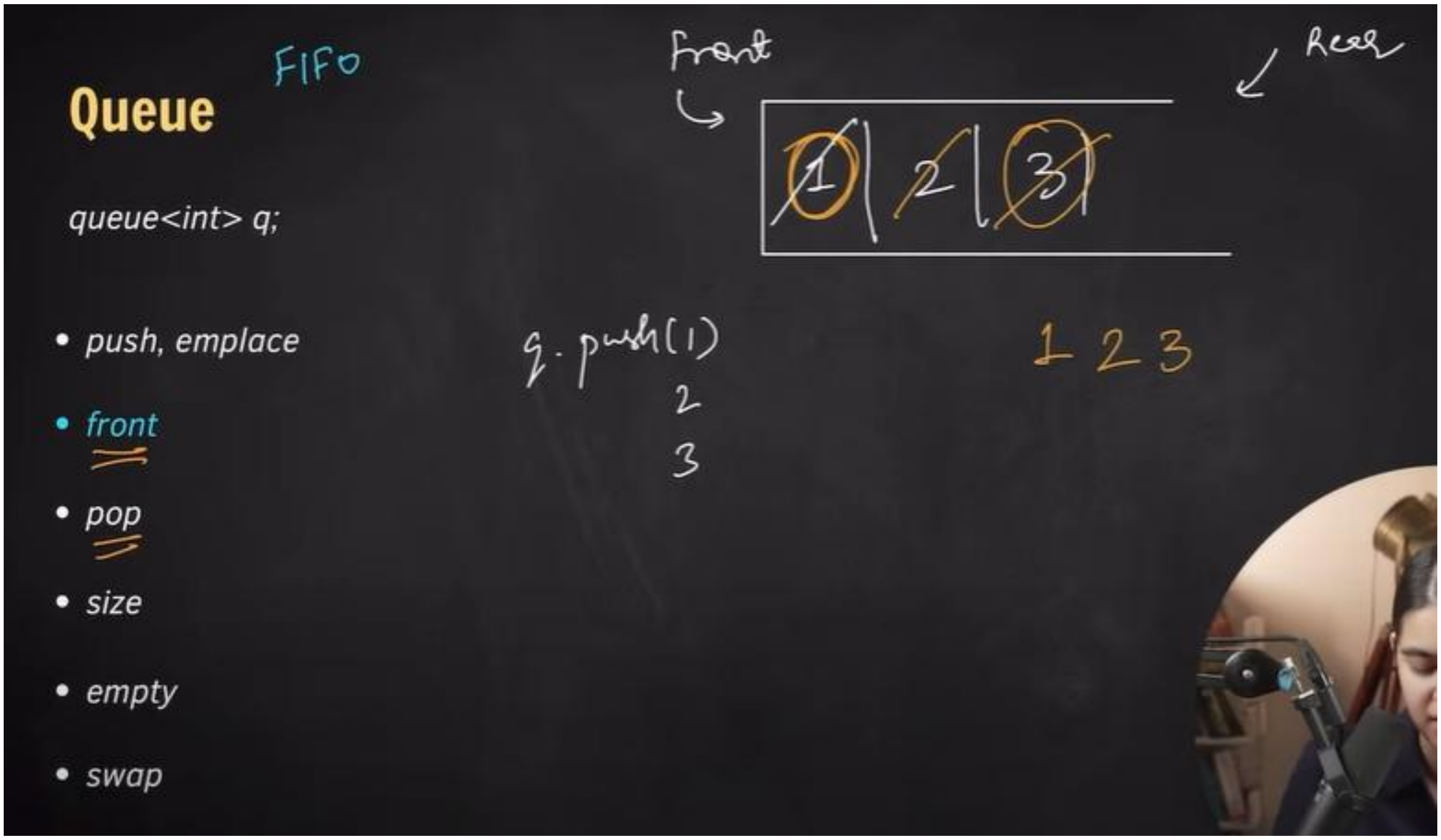
```cpp
#include<iostream>
#include<stack>//last in first out
using namespace std;

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:



```
s1 size before swapping : 4
s2 size before swapping : 0
s1 size after swapping : 0
s2 size after swapping : 4
s1 elements :
s2 elements : 4 3 2 1
```

Queue



Queue

queue<int> q;

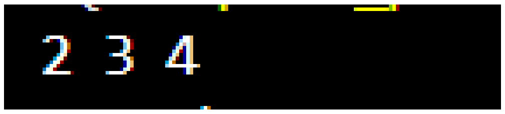
- push, emplace
- front
- pop
- size
- empty
- swap

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

Code:

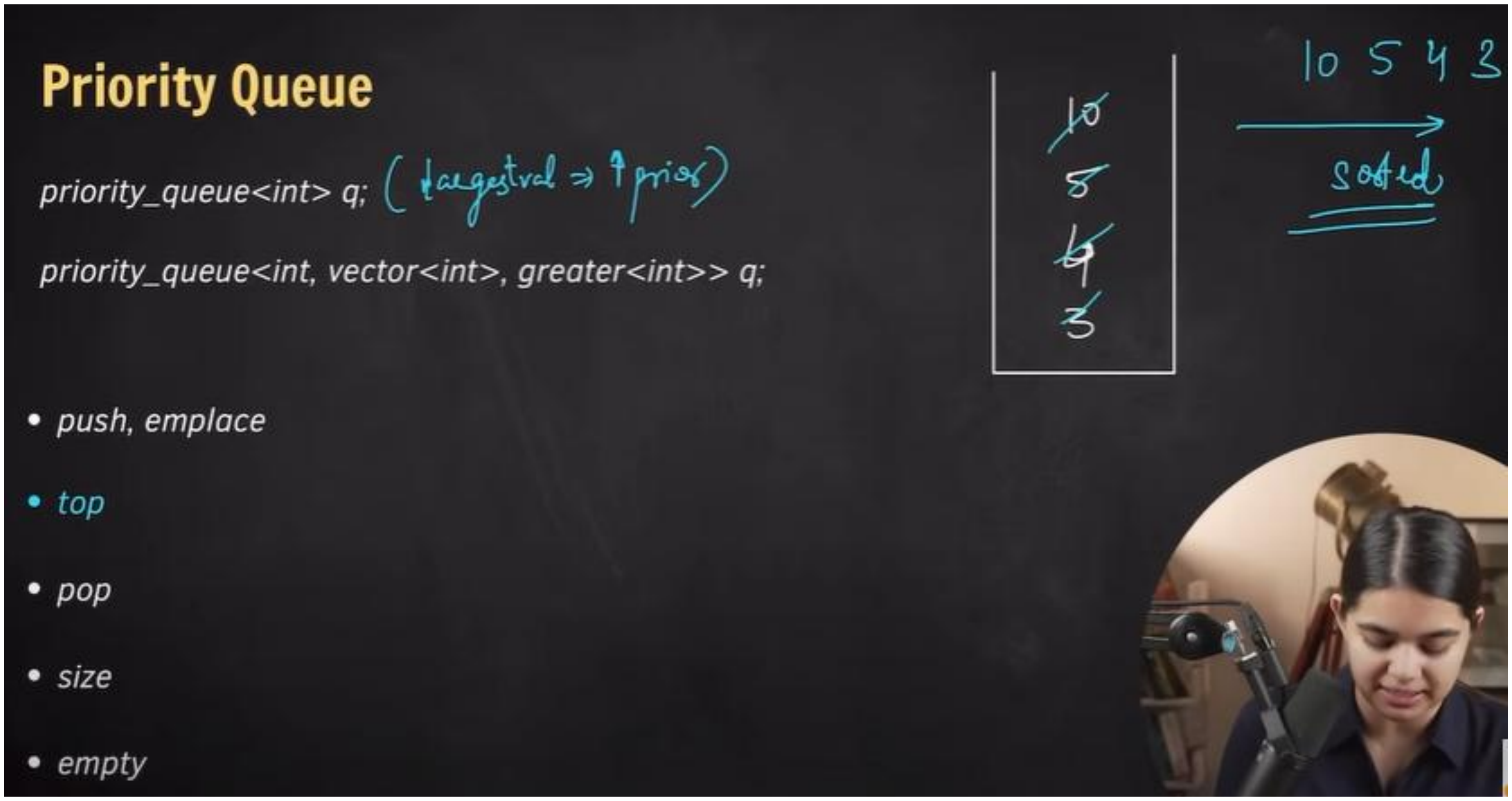
```cpp
#include<iostream>
#include<queue>//first in first out
using namespace std;

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:



```
2 3 4
```

Priority queue



Priority Queue

priority_queue<int> q; (largest val ⇒ ↑ prior)

priority_queue<int, vector<int>, greater<int>> q;

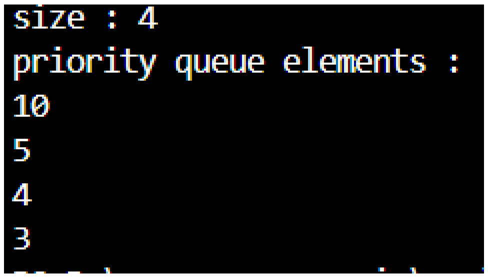
- push, emplace
- top
- pop
- size
- empty

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

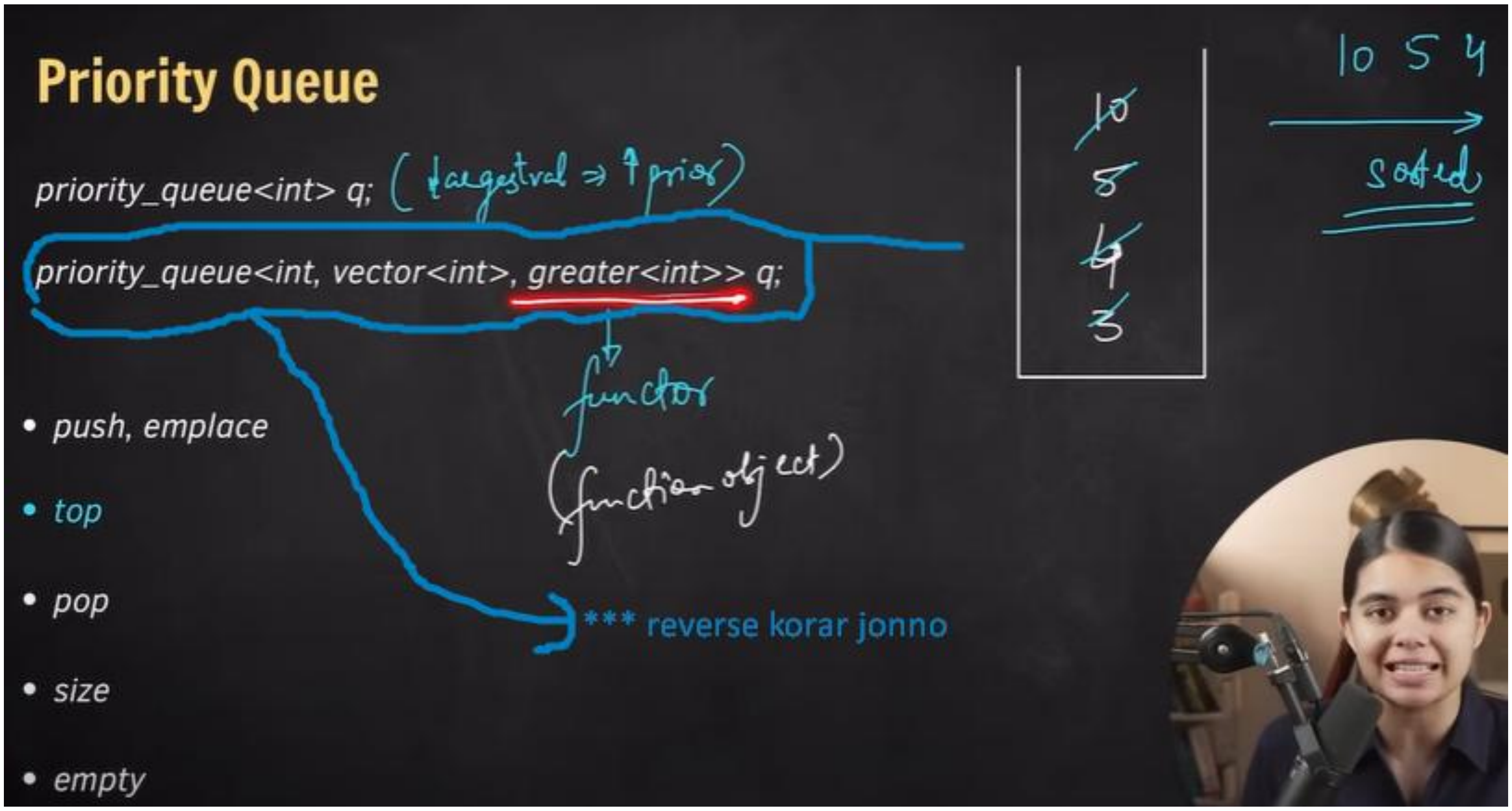
Code:

```
#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:



```
size : 4
priority queue elements :
10
5
4
3
```



Priority Queue

priority_queue<int> q; (largest val ⇒ ↑ prior)

priority_queue<int, vector<int>, greater<int>> q;

functor

(function object)

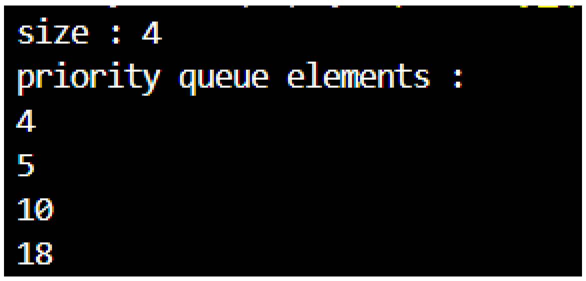
*** reverse korar jonno

- push, emplace
- top
- pop
- size
- empty

Reverse code:

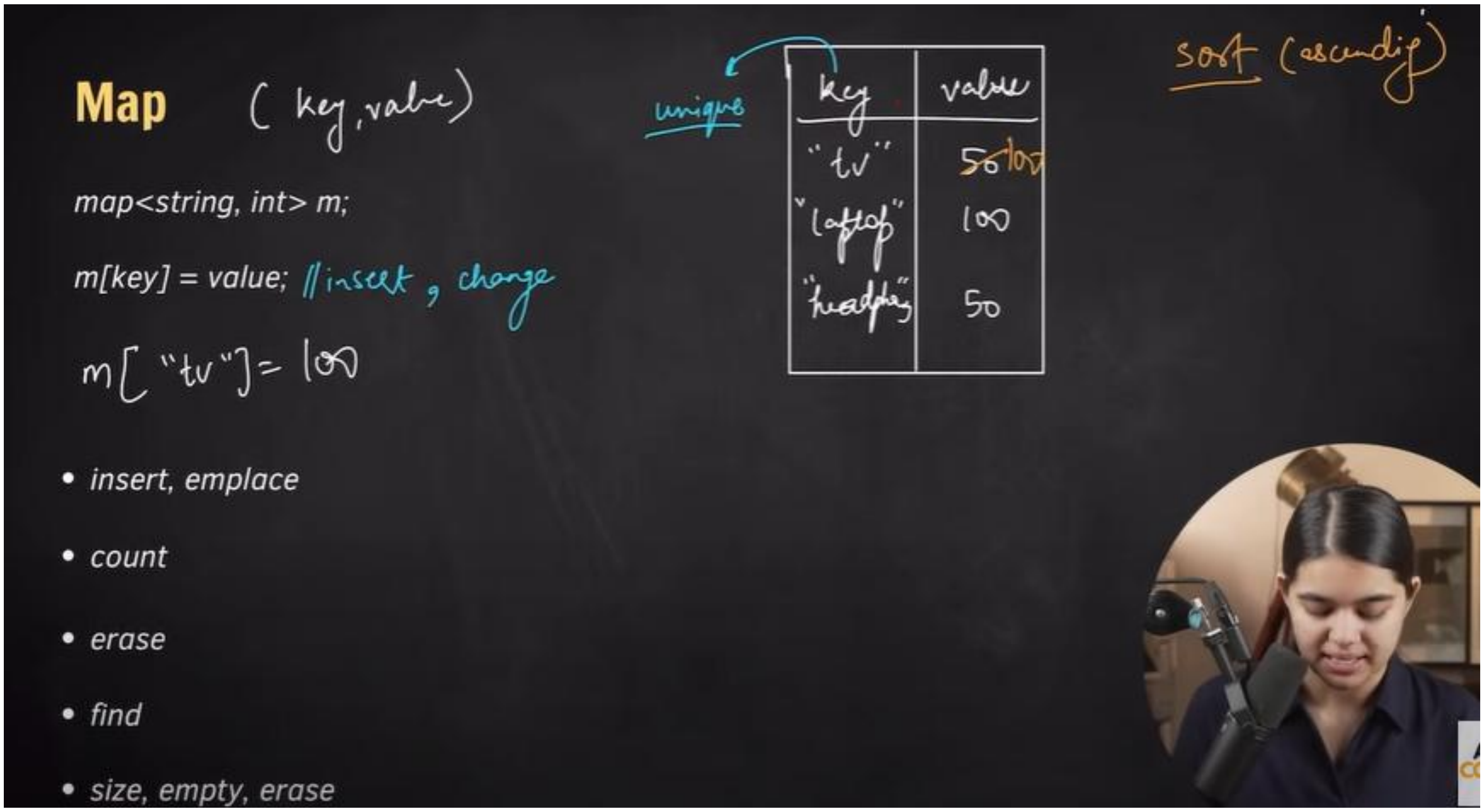
```cpp
#include<iostream>
#include<queue>
using namespace std;

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:



```
size : 4
priority queue elements :
4
5
10
18
```

Map



Map (key, value)

map<string, int> m;

m[key] = value; //insert, change

m["tv"]= 100

- insert, emplace
- count
- erase
- find
- size, empty, erase

unique

key	value
"tv"	~~50~~ 100
"laptop"	100
"headphones"	50

sort (ascending)

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:

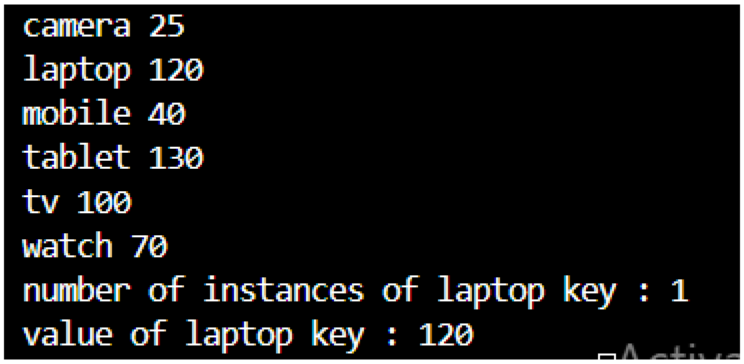
```
headphone 50
laptop 120
tablet 130
tv 99
watch 70
```

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

Code:

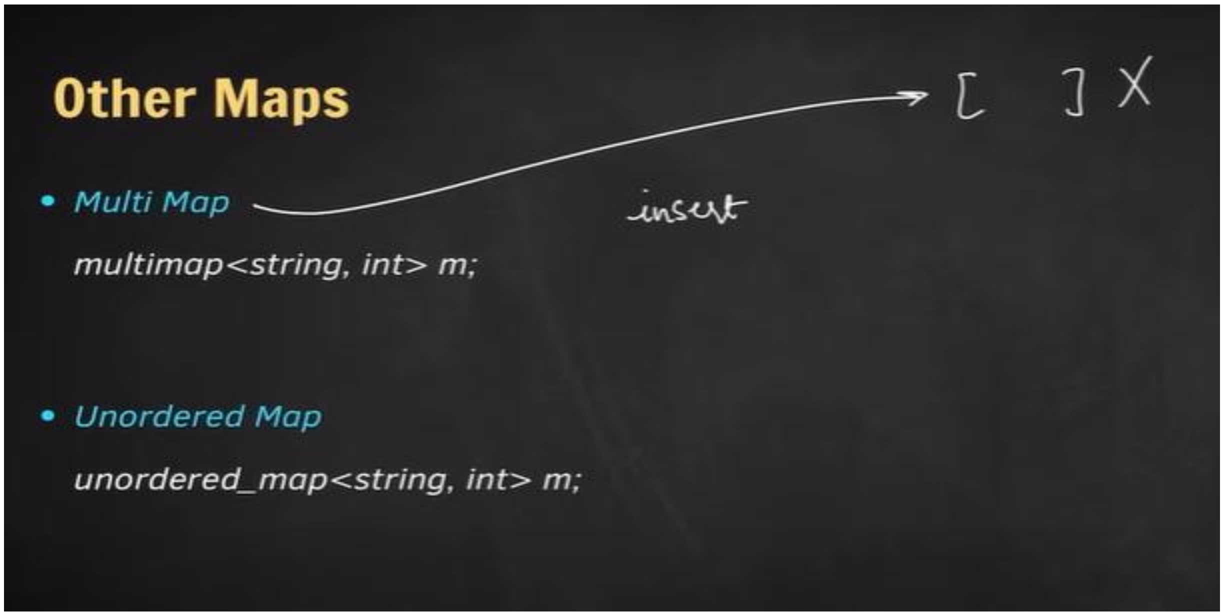
```
#include<iostream>
#include<map>
using namespace std;

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:



```
camera 25
laptop 120
mobile 40
tablet 130
tv 100
watch 70
number of instances of laptop key : 1
value of laptop key : 120
```

Multimap



Other Maps

[ ] X

- *Multi Map*

  insert

  multimap<string, int> m;

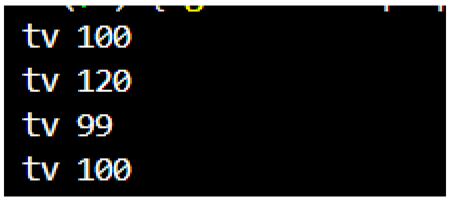
- *Unordered Map*

  unordered_map<string, int> m;

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;
}
```

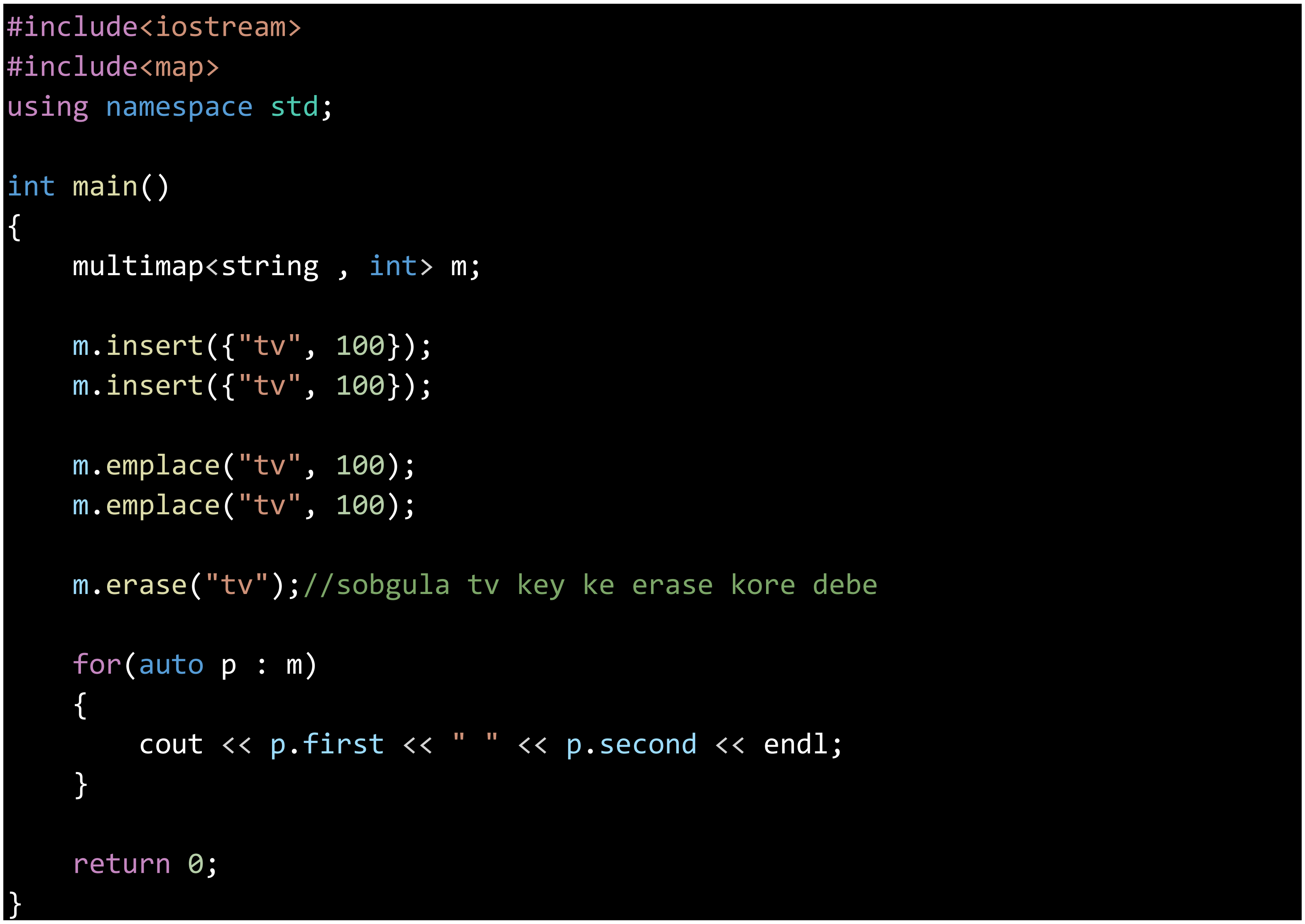
Result: (same key multiple times output debe, jei order e input dewa hoyeche shei order e)



```
tv 100
tv 120
tv 99
tv 100
```

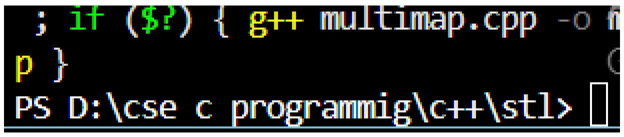
Erase code:

Sobgula same key erase korar jonno



```cpp
#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:



```
 ; if ($?) { g++ multimap.cpp -o
p }
PS D:\cse c programmig\c++\stl>
```

Erase code 2:

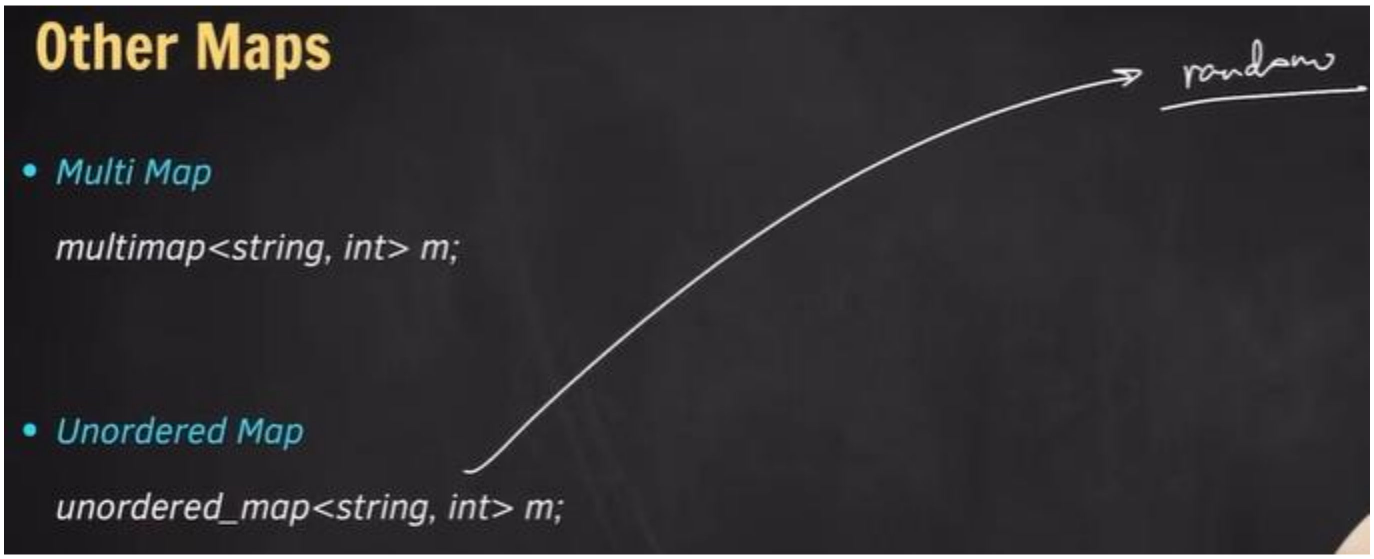
Sudhu first same key erase korar jonno find() function use korte hobe erase er moddhe. Ebhabe iterator use kore ekta key erase kora jai

```cpp
#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:

```
tv 100
tv 282
tv 737
```

Unordered map



Other Maps

- *Multi Map*

  multimap<string, int> m;

- *Unordered Map*

  unordered_map<string, int> m;

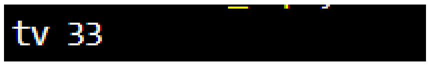
Code:

```
#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:

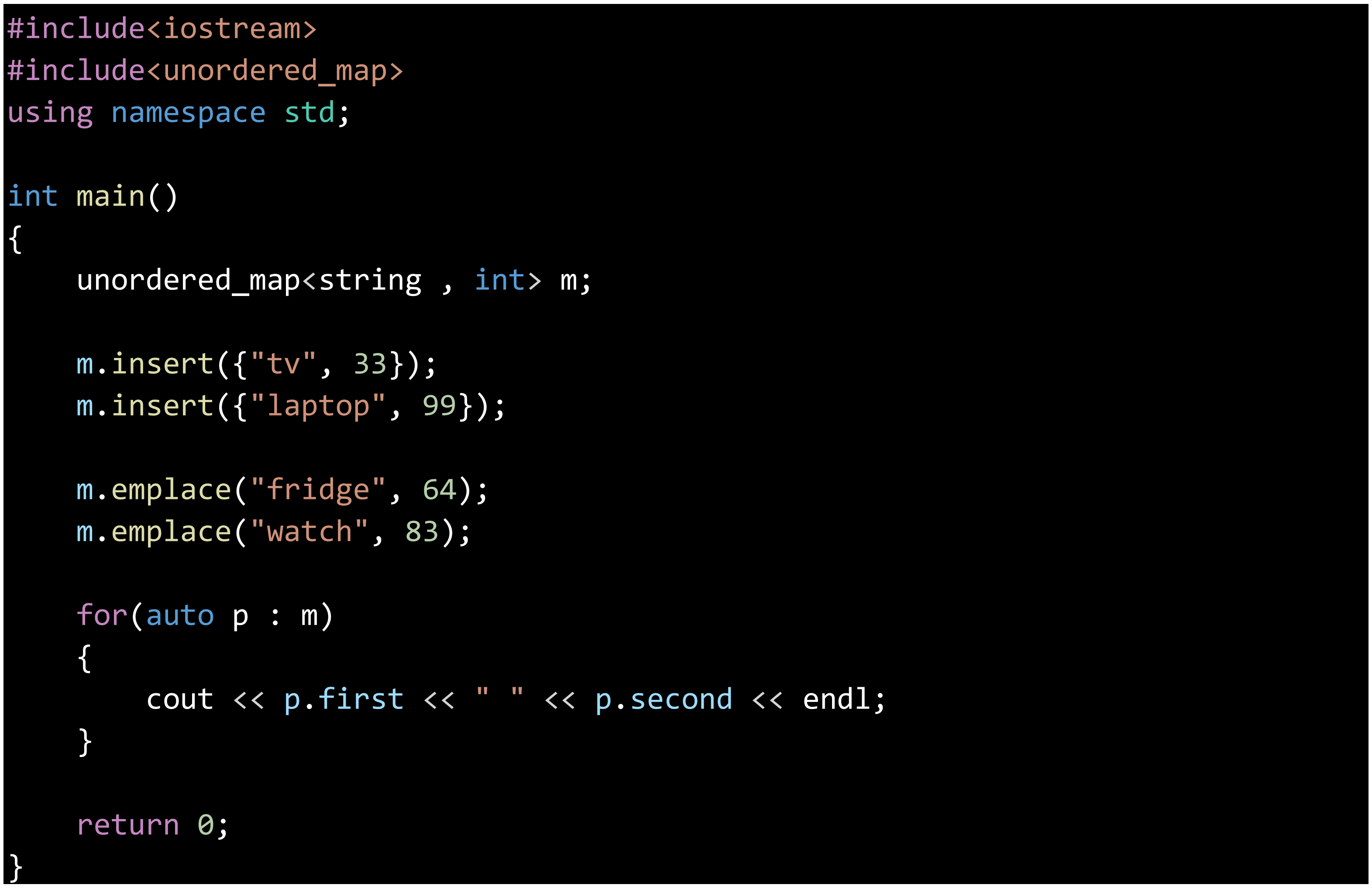
(same key multiple time insert korleo ekbar e output dibe

Sheta holo first key er value)



```
tv 33
```

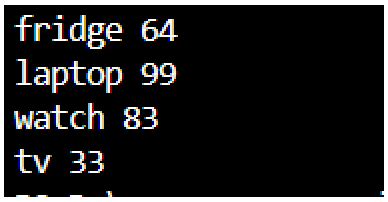
Code:



```
#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)

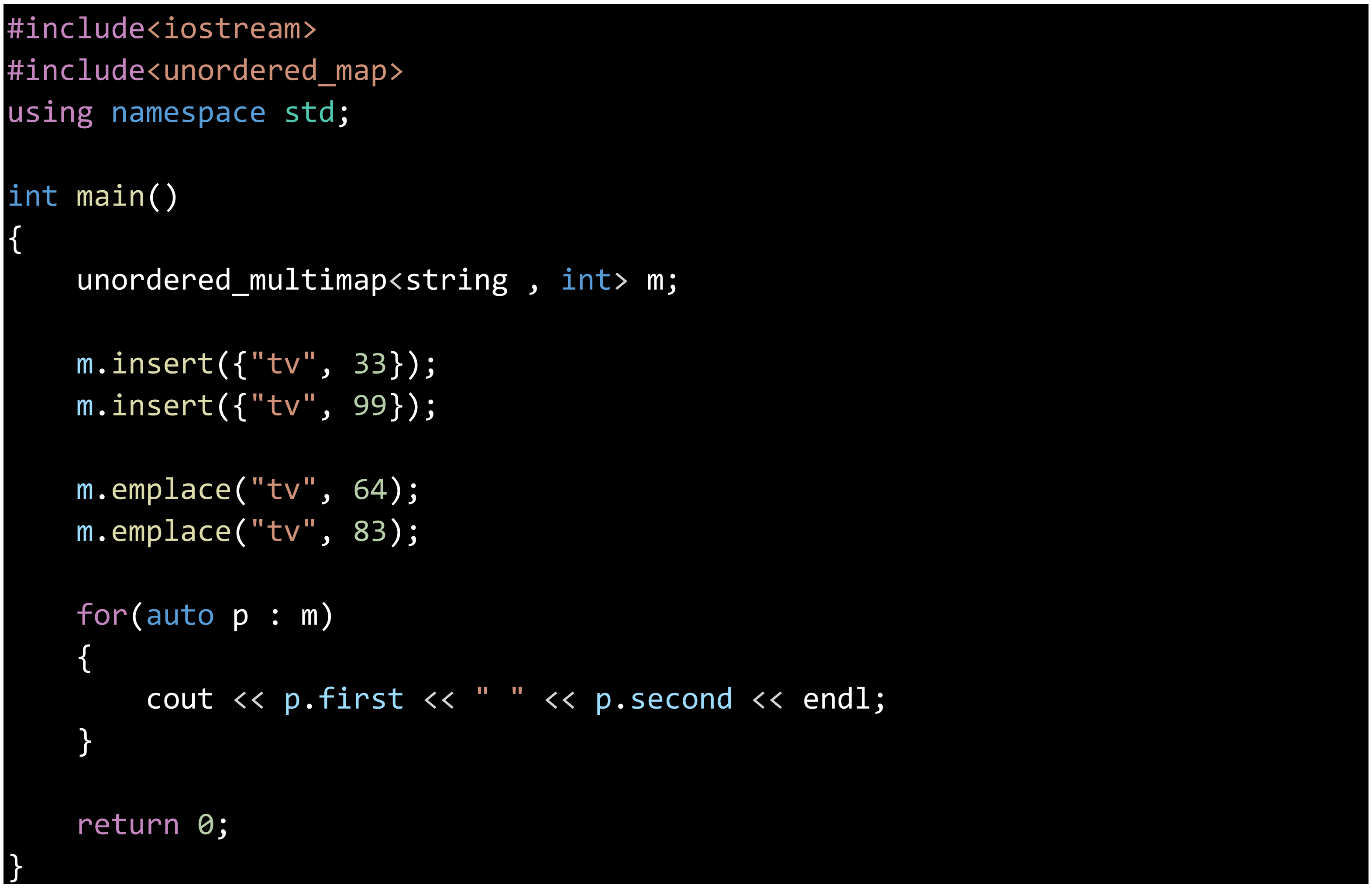


```
fridge 64
laptop 99
watch 83
tv 33
```

Unordered multimap

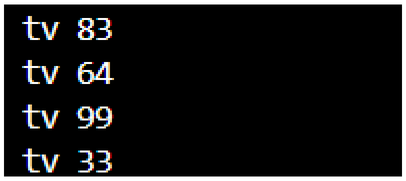
**same key unsorted bhabe multiple times output dei

Code:



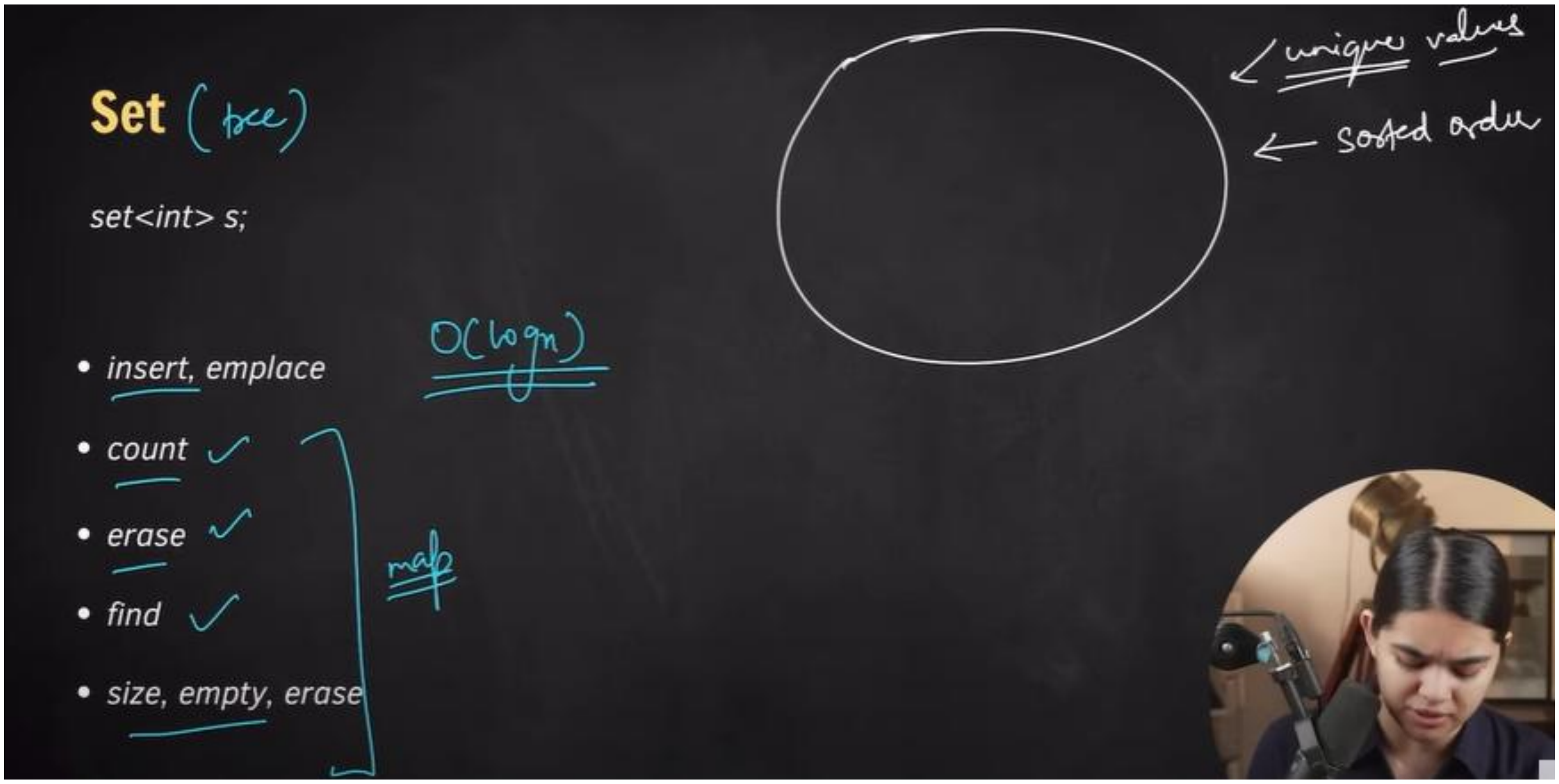
```
#include<iostream>
#include<unordered_map>
using namespace std;

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:



```
tv 83
tv 64
tv 99
tv 33
```

Set



Set (tree)

set<int> s;

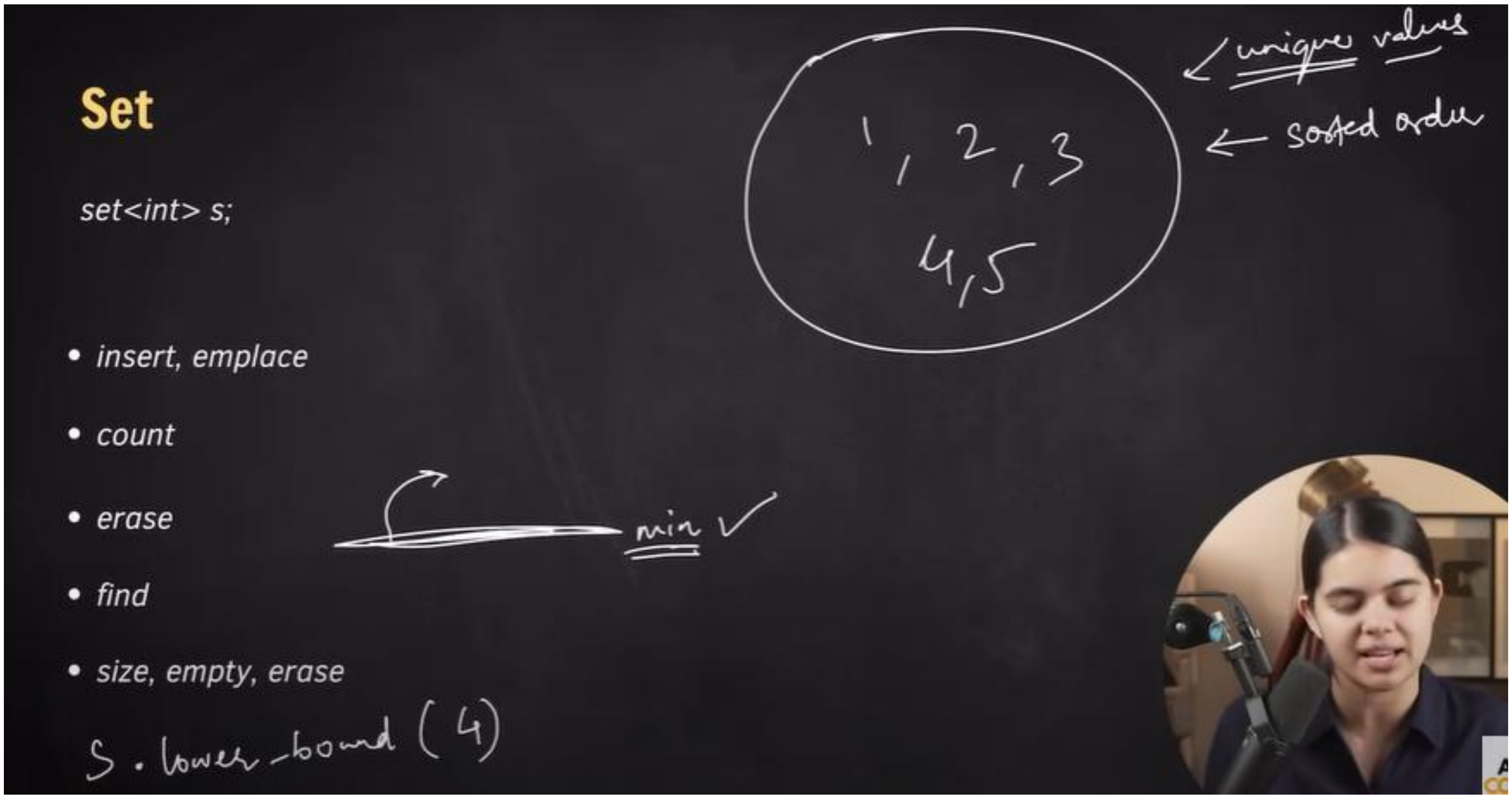
- insert, emplace
- count ✓
- erase ✓
- find ✓
- size, empty, erase

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:

```
size : 5
1 2 3 4 8
```



Set

set<int> s;

- *insert, emplace*
- *count*
- *erase*
- *find*
- *size, empty, erase*

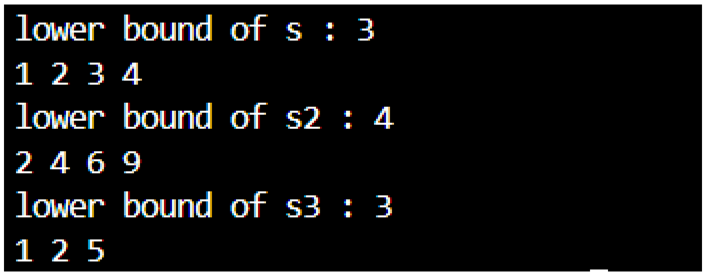
Lower_bound code:

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

int main()
```

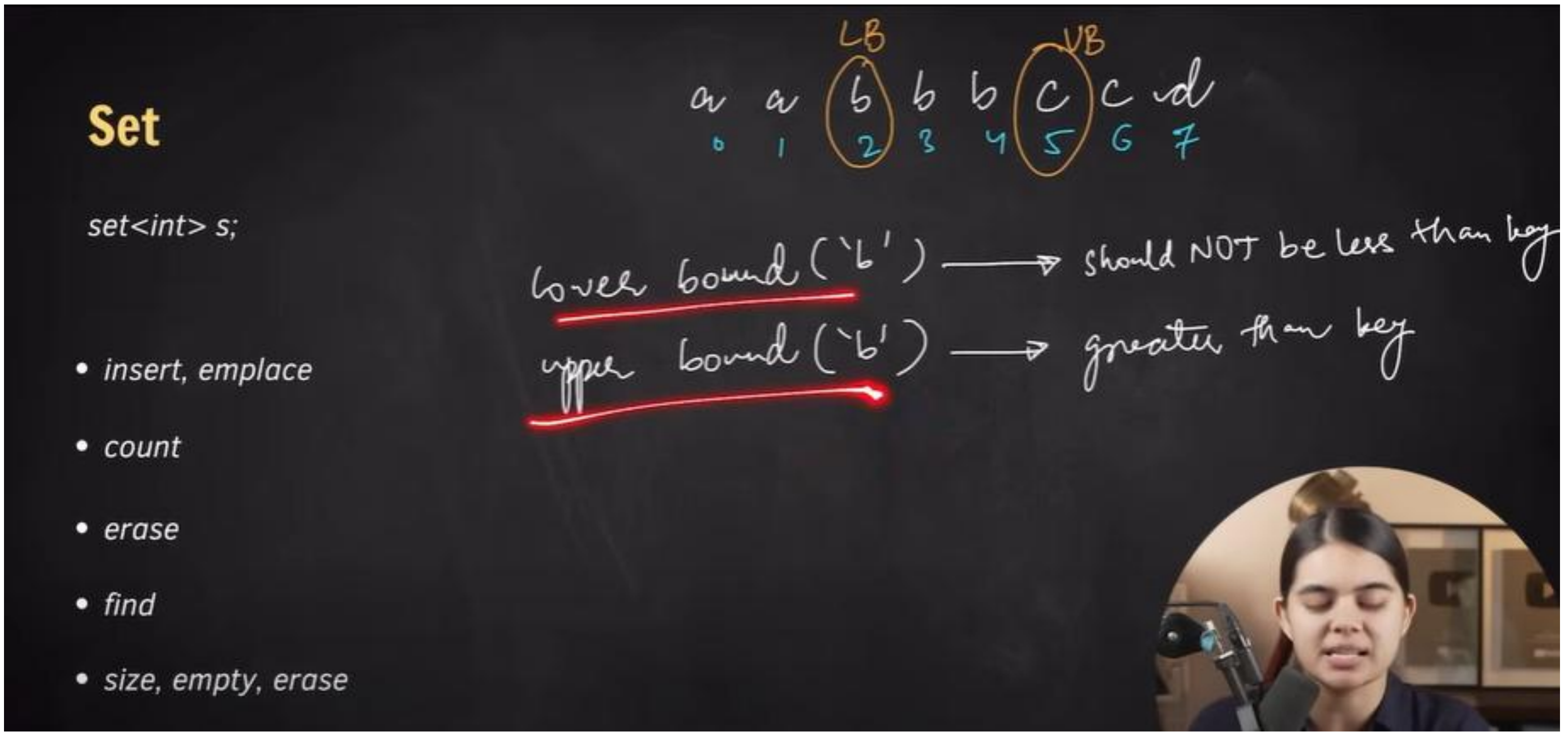
```
{
    //lower bound means minimmum
    set<int> s;

    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)



```
lower bound of s : 3
1 2 3 4
lower bound of s2 : 4
2 4 6 9
lower bound of s3 : 3
1 2 5
```



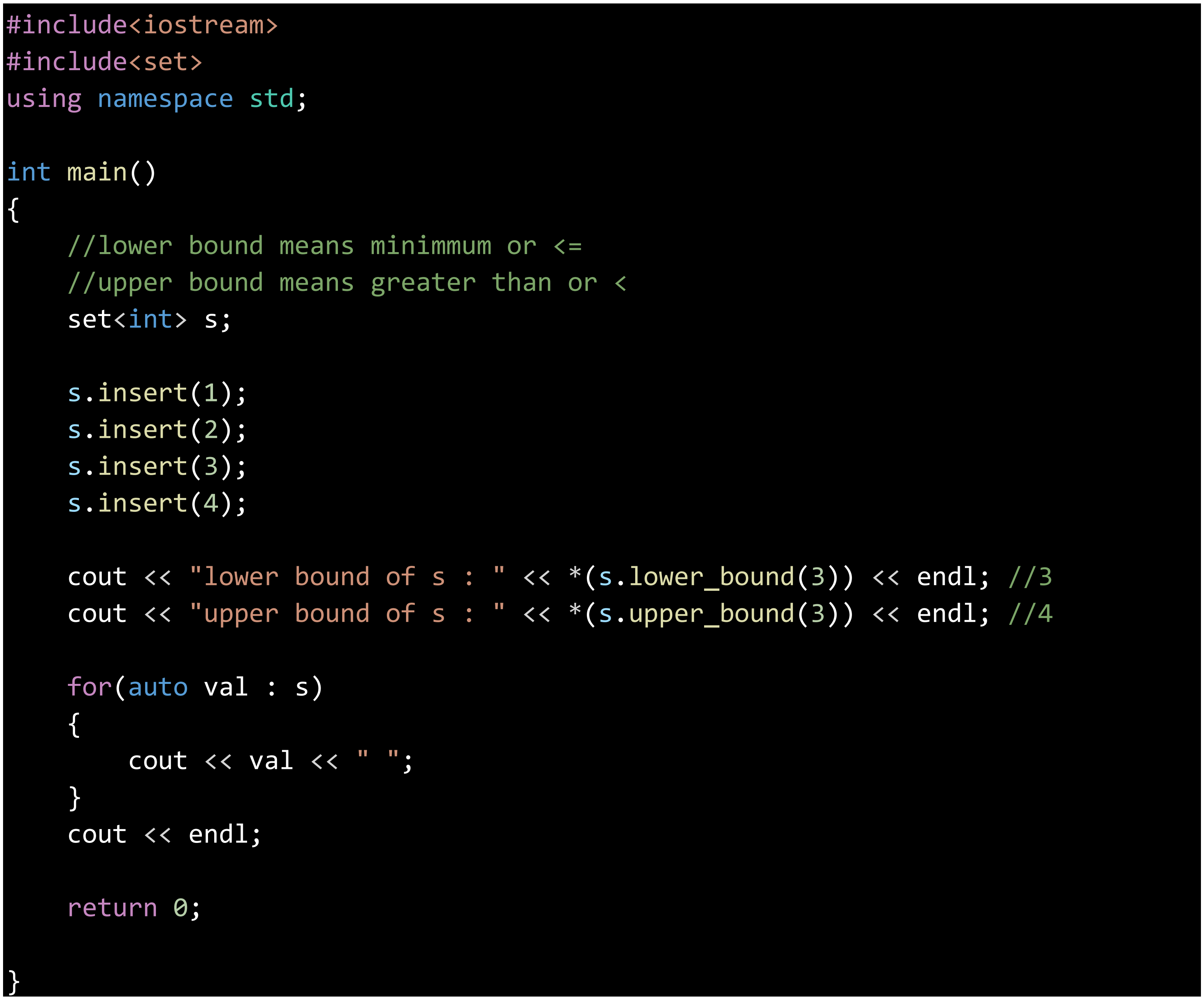
Set

set<int> s;

- insert, emplace
- count
- erase
- find
- size, empty, erase

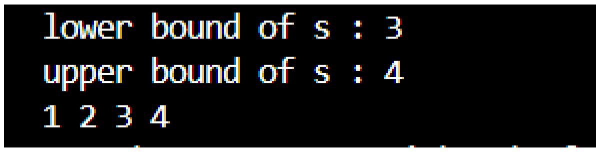
LB
UB
a a b b b c c d
0 1 2 3 4 5 6 7
lower bound ('b') → should NOT be less than key
upper bound ('b') → greater than key

Upper_bound code:



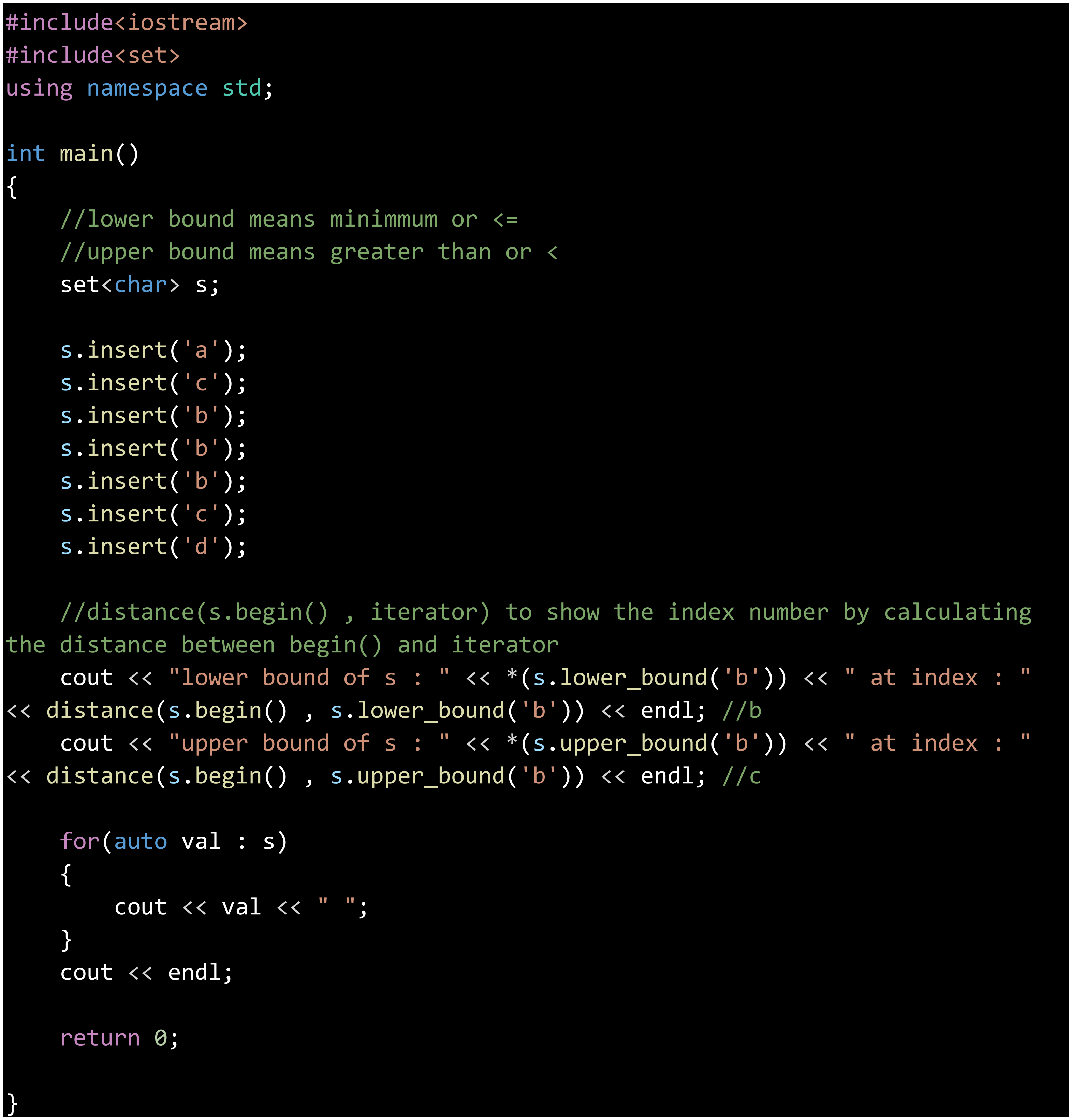
```
#include<iostream>
#include<set>
using namespace std;

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:



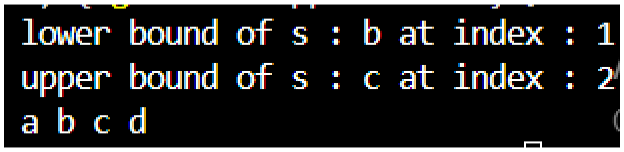
```
lower bound of s : 3
upper bound of s : 4
1 2 3 4
```

Char datatype lower_bound and upper_bound and index number code:



```cpp
#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:

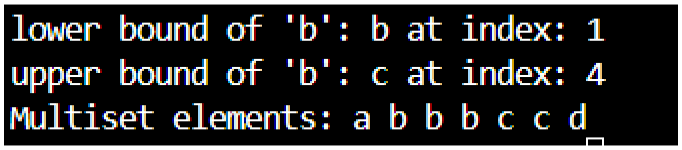


```
lower bound of s : b at index : 1
upper bound of s : c at index : 2
a b c d
```

Multiset index output code:

```cpp
#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:

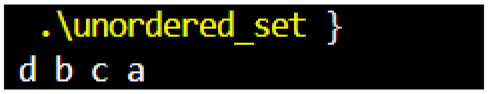


```
lower bound of 'b': b at index: 1
upper bound of 'b': c at index: 4
Multiset elements: a b b b c c d
```

Unordered set code:

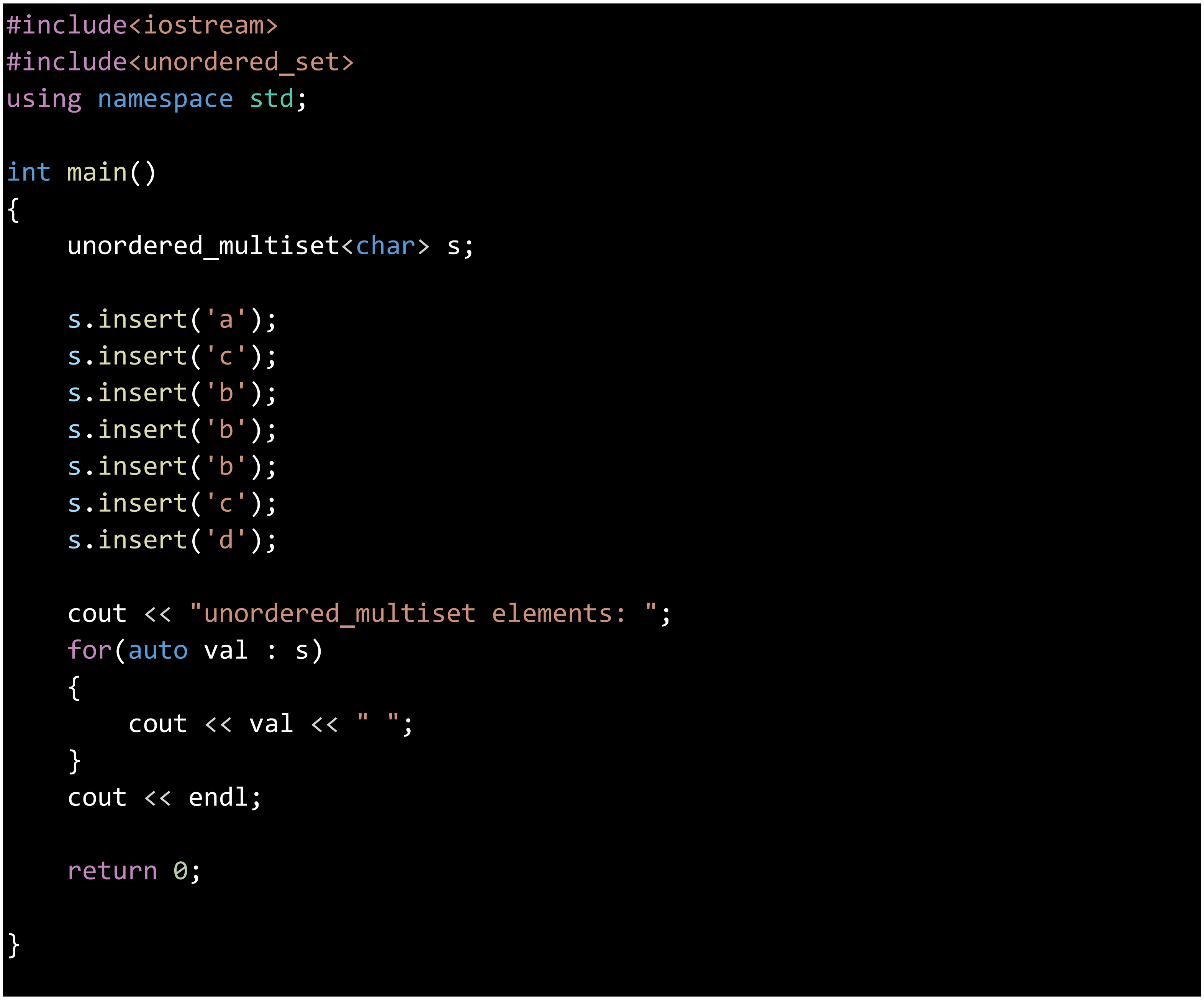
```cpp
#include<iostream>
#include<unordered_set>
using namespace std;

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:



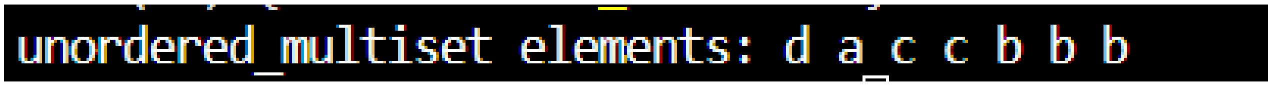
```
 .\unordered_set }
d b c a
```

Unordered multiset code:



```cpp
#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;

}
```

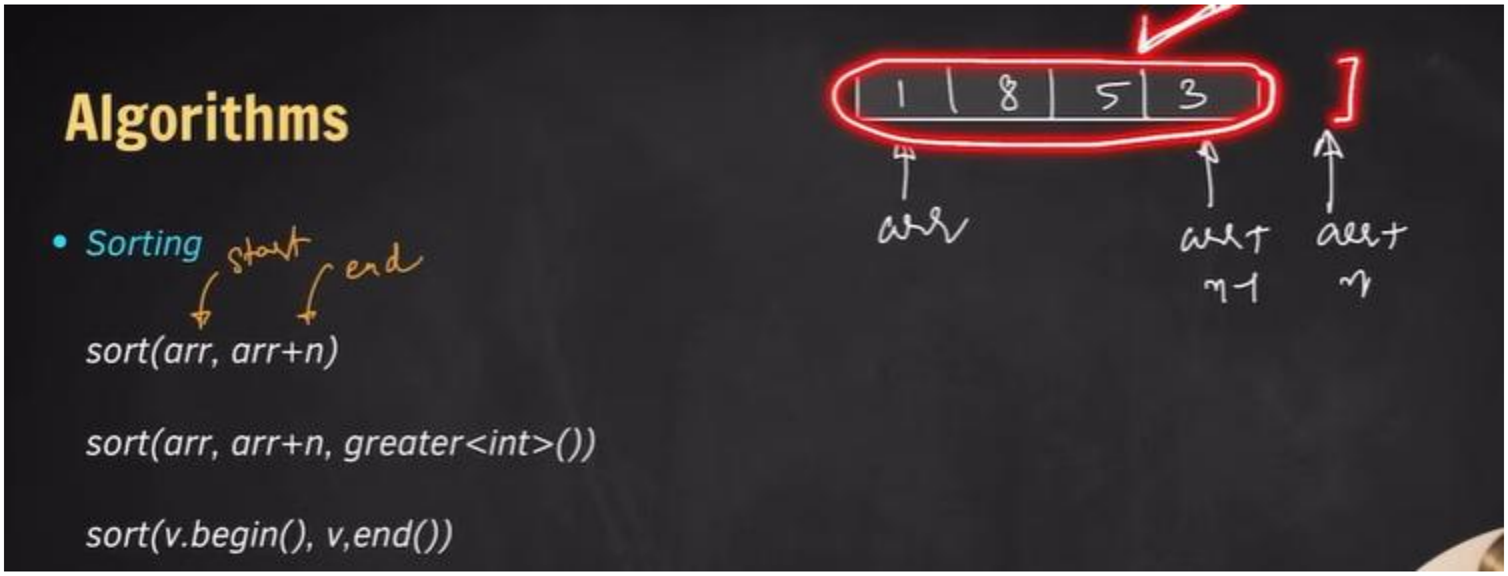
Result:



```
unordered_multiset elements: d a c c b b b
```

Algorithms

Sorting



Algorithms

- Sorting

sort(arr, arr+n)

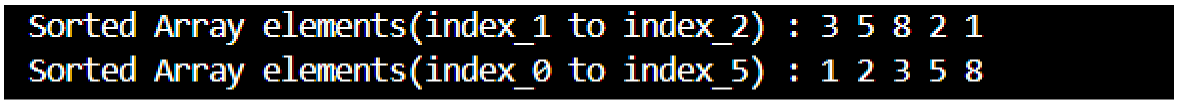
sort(arr, arr+n, greater<int>())

sort(v.begin(), v,end())

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:

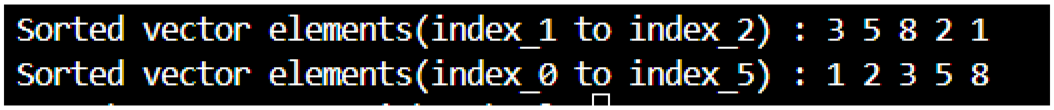


```
Sorted Array elements(index_1 to index_2) : 3 5 8 2 1
Sorted Array elements(index_0 to index_5) : 1 2 3 5 8
```

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;
}
```

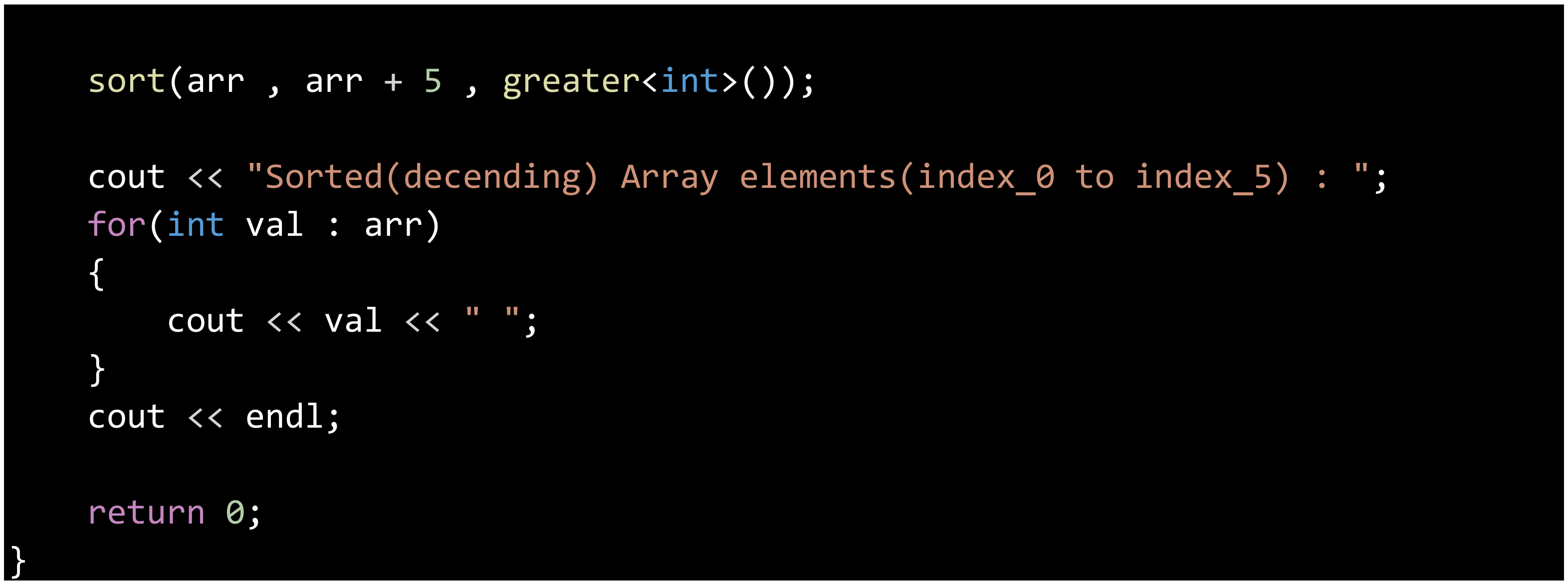
Result:



```
Sorted vector elements(index_1 to index_2) : 3 5 8 2 1
Sorted vector elements(index_0 to index_5) : 1 2 3 5 8
```

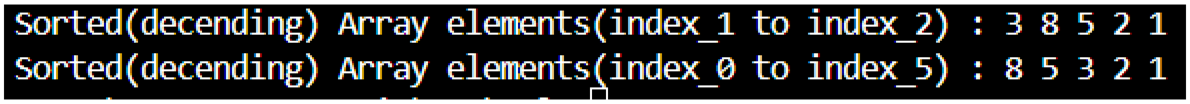
Sorted array(decending) code:

```
#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;
```



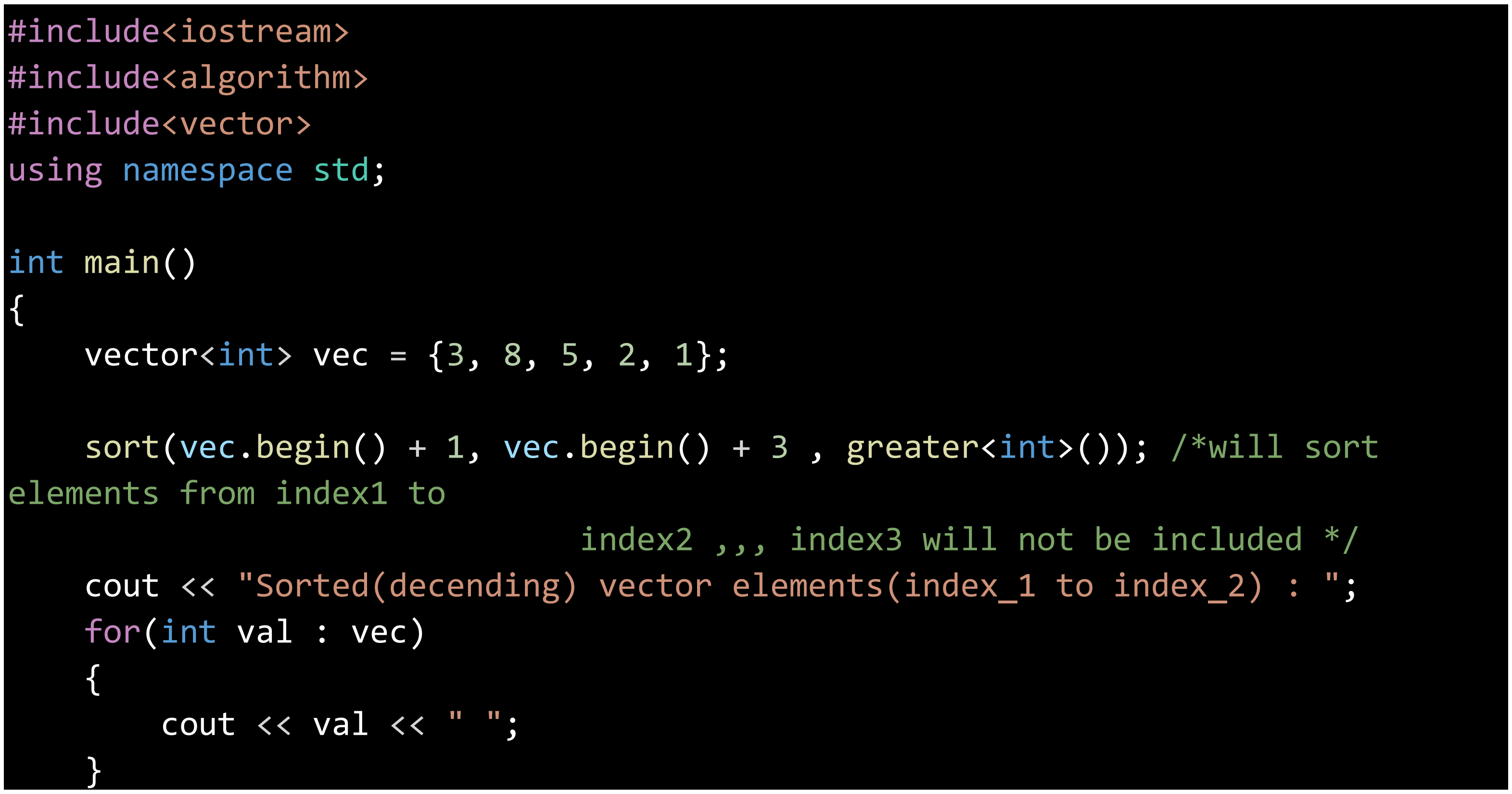
```
    sort(arr , arr + 5 , greater<int>());

    cout << "Sorted(decending) Array elements(index_0 to index_5) : ";
    for(int val : arr)
    {
        cout << val << " ";
    }
    cout << endl;

    return 0;
}
```

Result:



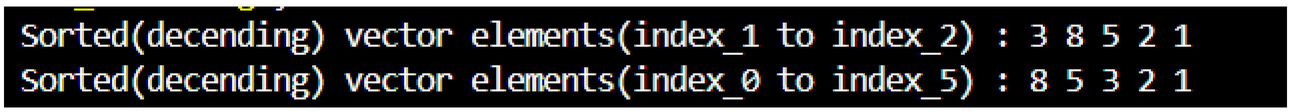
```
Sorted(decending) Array elements(index_1 to index_2) : 3 8 5 2 1
Sorted(decending) Array elements(index_0 to index_5) : 8 5 3 2 1
```

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:

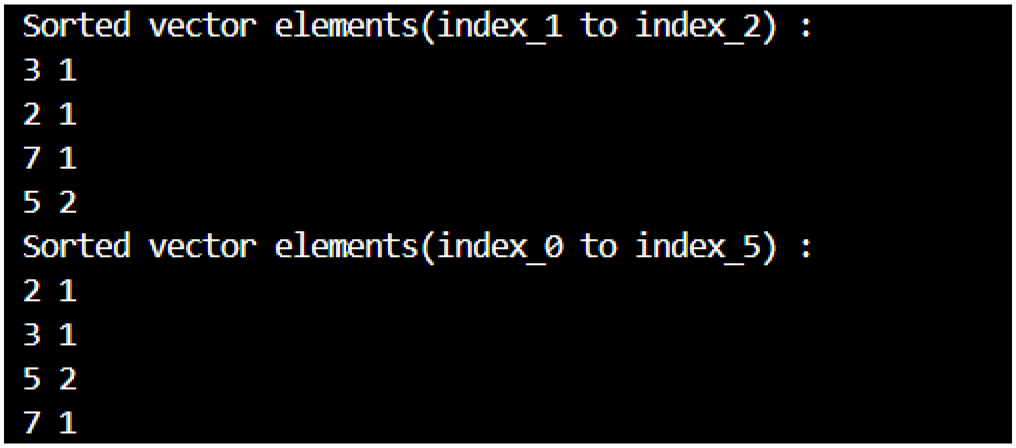


```
Sorted(decending) vector elements(index_1 to index_2) : 3 8 5 2 1
Sorted(decending) vector elements(index_0 to index_5) : 8 5 3 2 1
```

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:



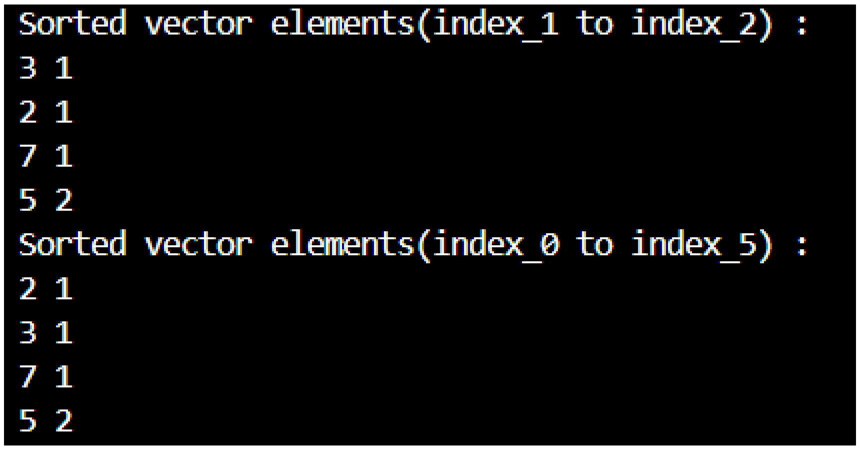
```
Sorted vector elements(index_1 to index_2) :
3 1
2 1
7 1
5 2
Sorted vector elements(index_0 to index_5) :
2 1
3 1
5 2
7 1
```

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()
```

```cpp
{
    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:



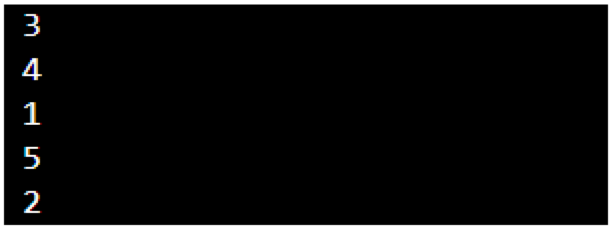
```
Sorted vector elements(index_1 to index_2) :
3 1
2 1
7 1
5 2
Sorted vector elements(index_0 to index_5) :
2 1
3 1
7 1
5 2
```

Reverse

Code:

```cpp
#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;

}
```

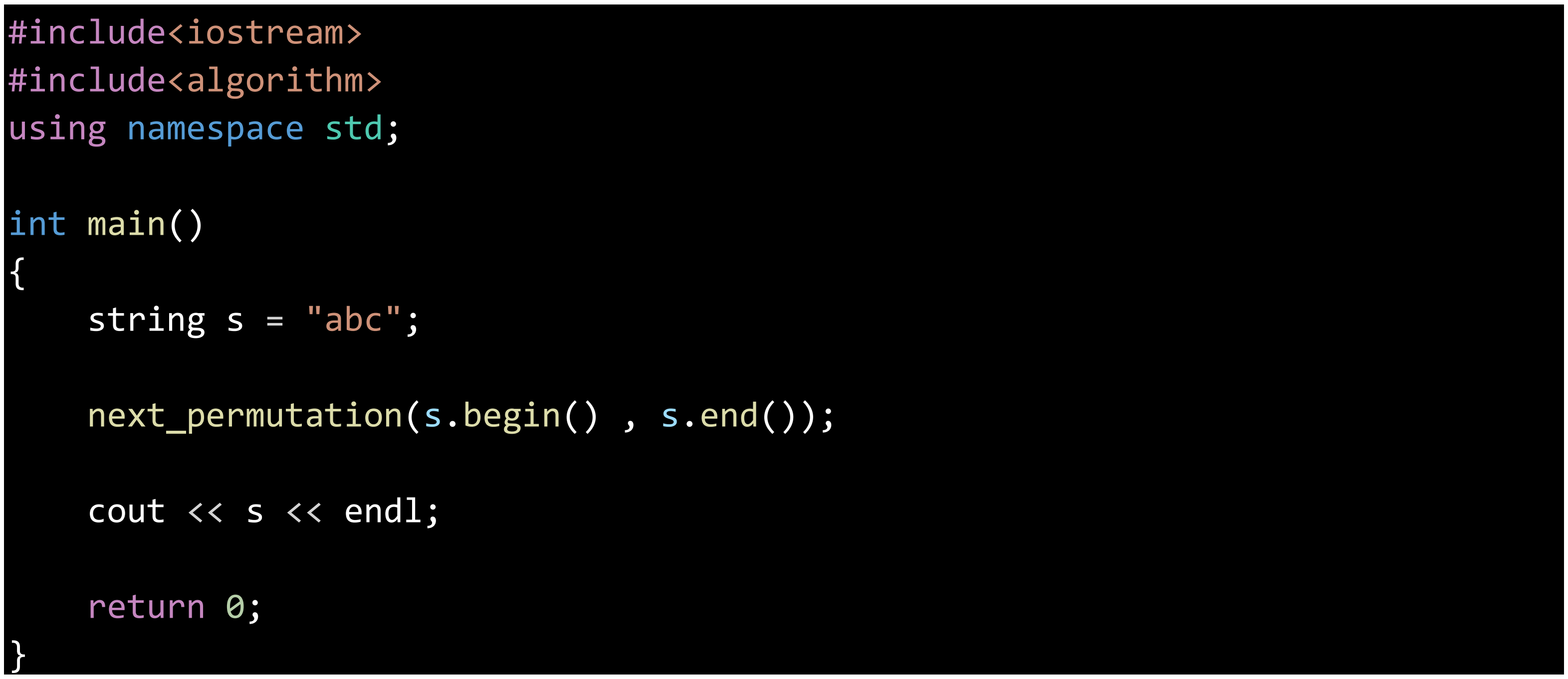
Result:



```
3
4
1
5
2
```

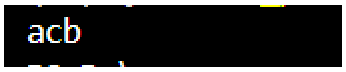
Next permutation

Code:

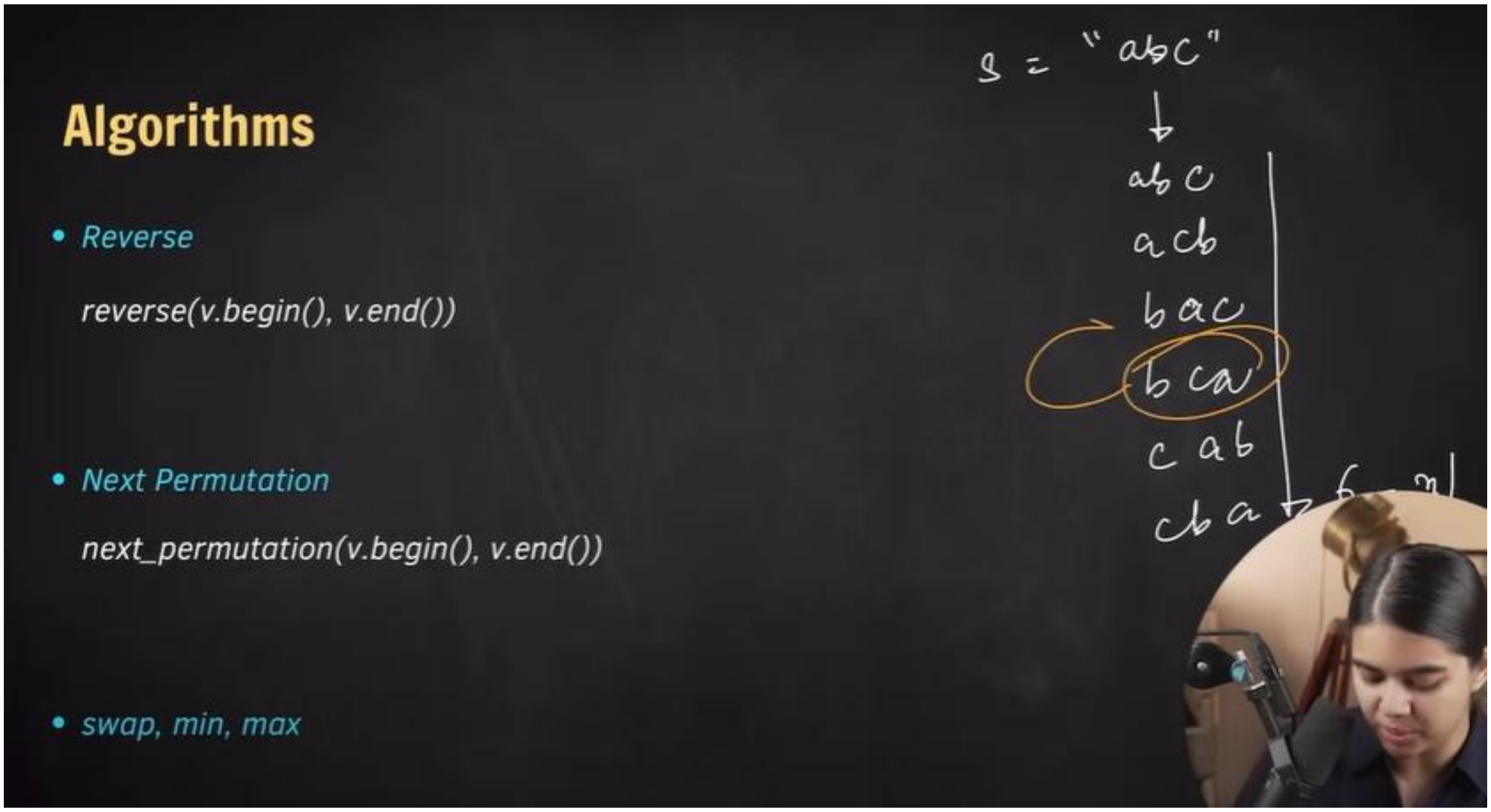


```cpp
#include<iostream>
#include<algorithm>
using namespace std;

int main()
{
    string s = "abc";

    next_permutation(s.begin() , s.end());

    cout << s << endl;

    return 0;
}
```

Result:



```
acb
```



Algorithms

- Reverse

  reverse(v.begin(), v.end())

- Next Permutation

  next_permutation(v.begin(), v.end())

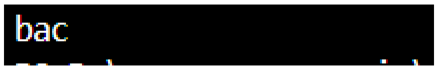
- swap, min, max

Previous permutation

Code:

```cpp
#include<iostream>
#include<algorithm>
using namespace std;

int main()
{
    string s = "bca";

    prev_permutation(s.begin() , s.end());

    cout << s << endl;

    return 0;
}
```

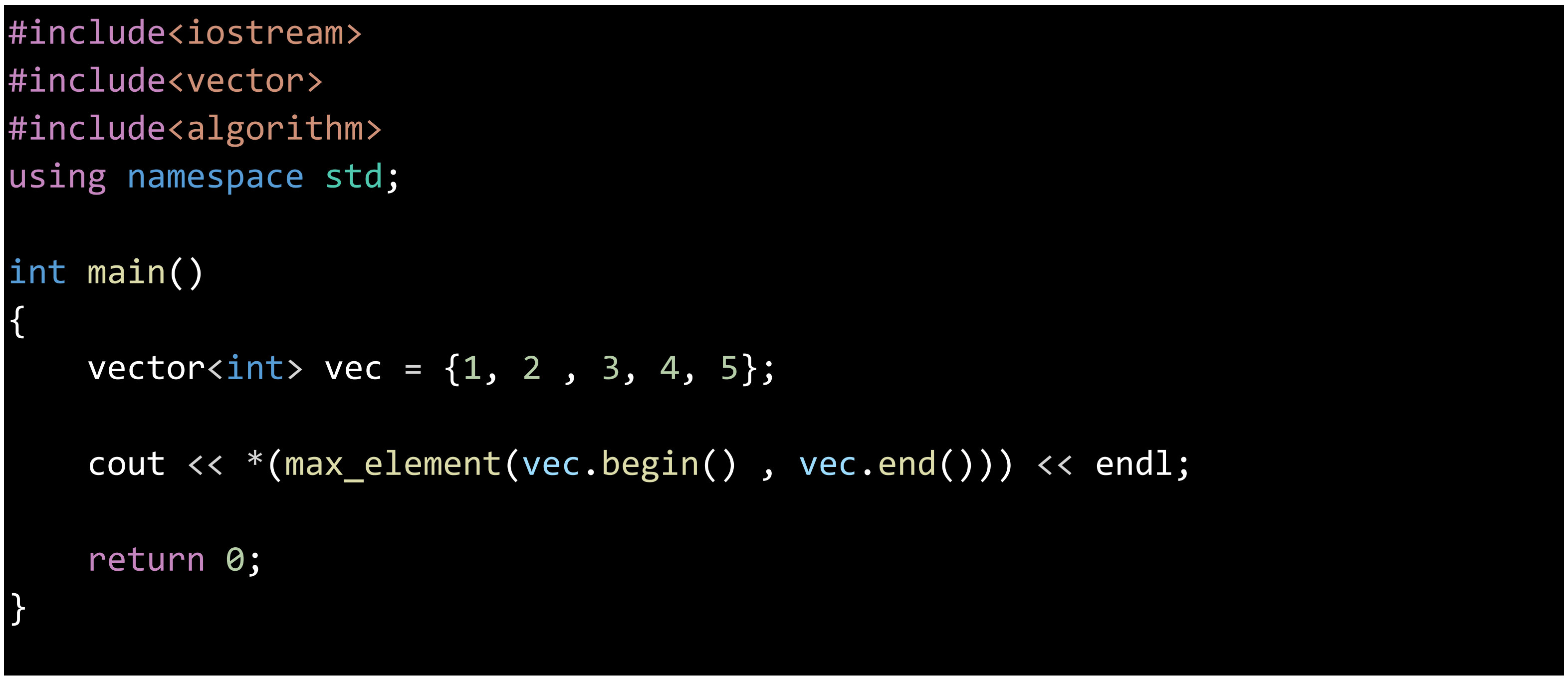
Result:



```
bac
```

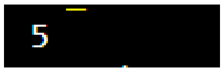
Max element

Code:



```cpp
#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:



```
5
```

Min element

Code:

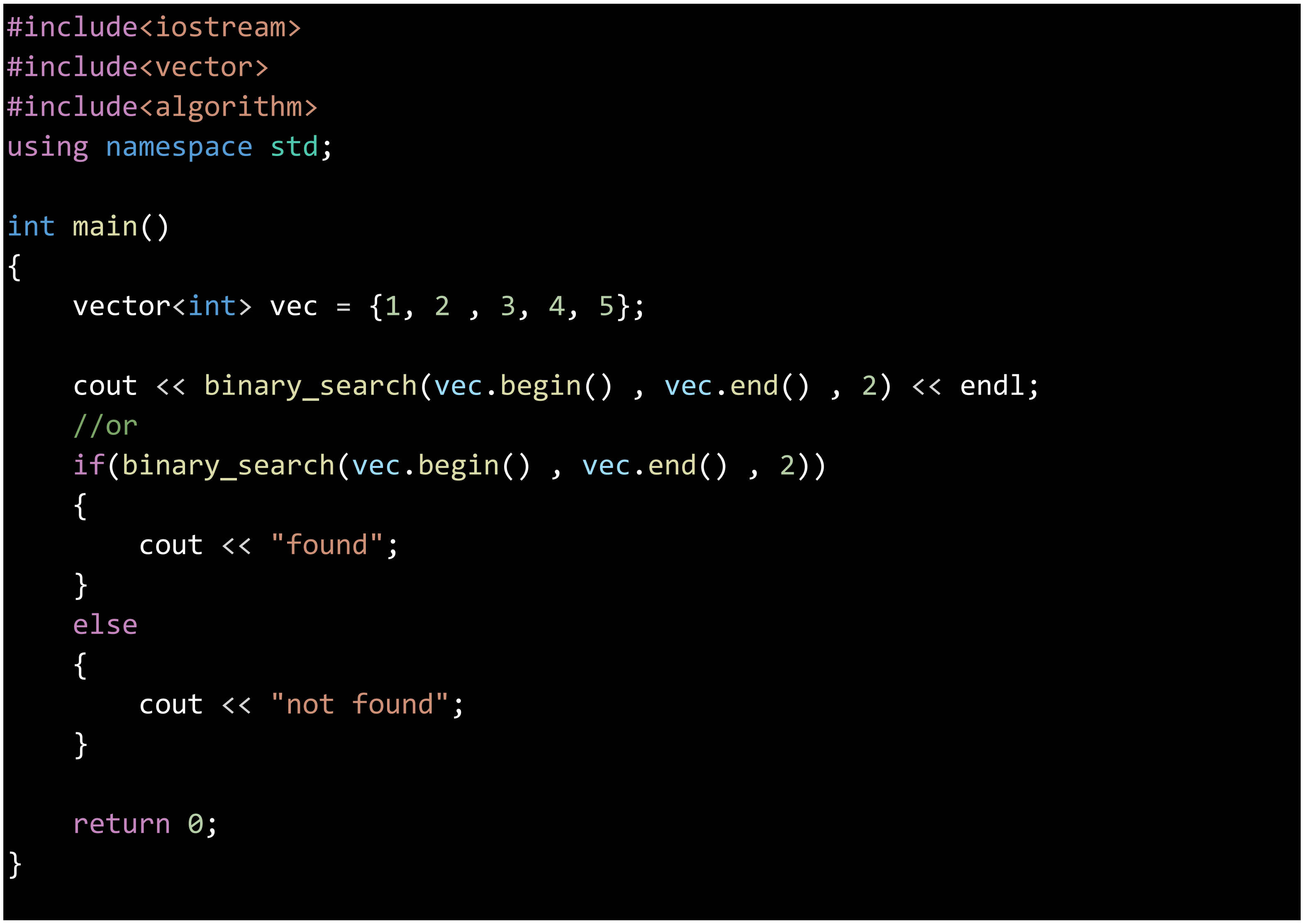
```cpp
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

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

    cout << *(min_element(vec.begin() , vec.end())) << endl;

    return 0;
}
```

Result:

```
1
```

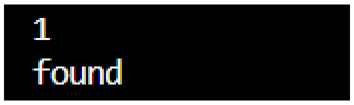
Binary search

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:



```
1
found
```