Standard Template Library (STL)

INTRODUCTION

STL is a collection of:

- Classes (Data Structure)
- Functions (Algorithm)

Component of STL:

- Container
- Algorithm
- Iterators

CONTAINERS

Containers are data structure to store data.

Mostly used Containers:

- Vector
- Stack
- Queue
- Priority Queue
- Map

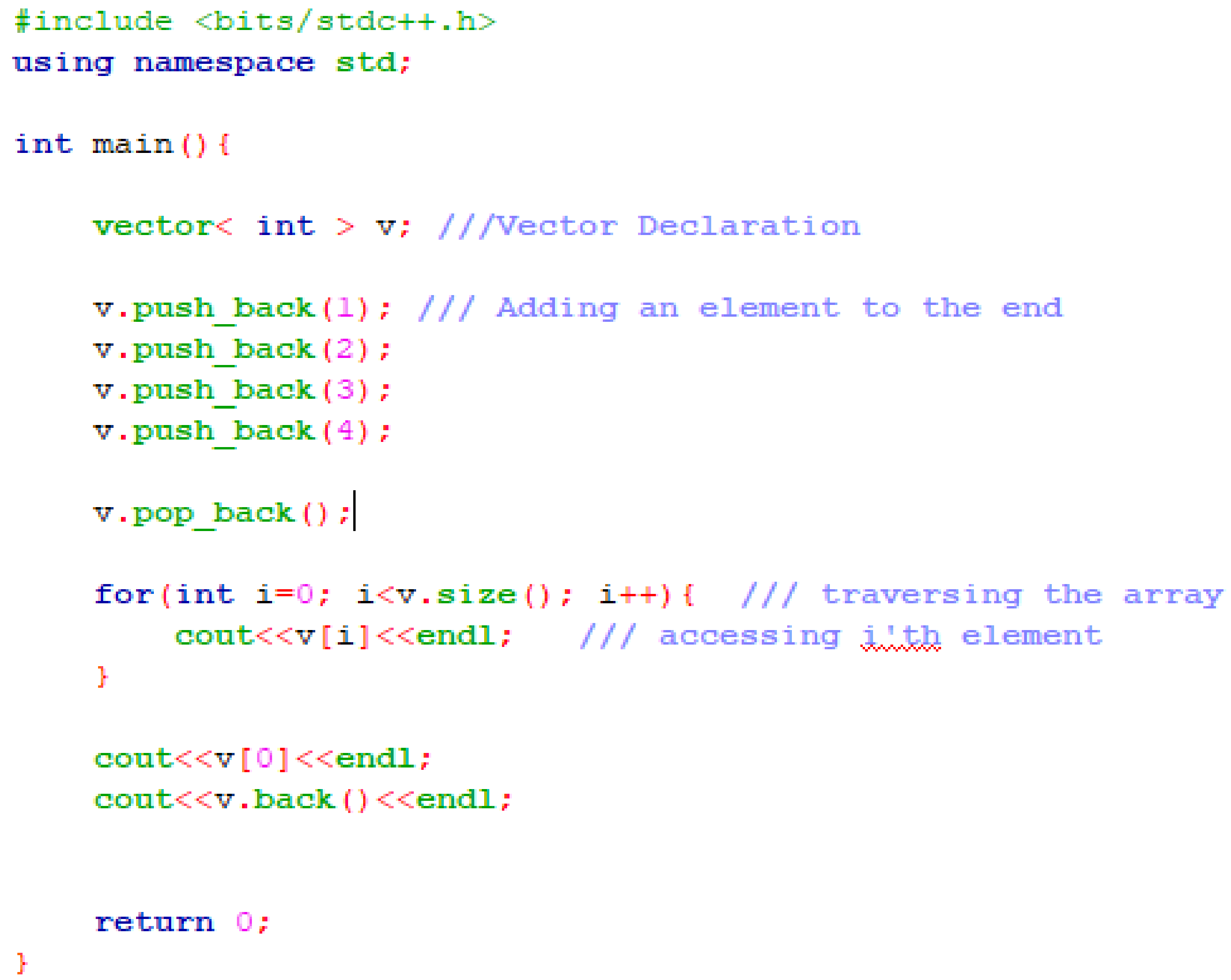
VECTOR

- Vector is most widely used container.
- Alternative of array
- Change its size dynamically and allocate memory as needed

Member Function:

- begin()
- end()
- size()
- back()
- push_back()
- pop_back()

VECTOR



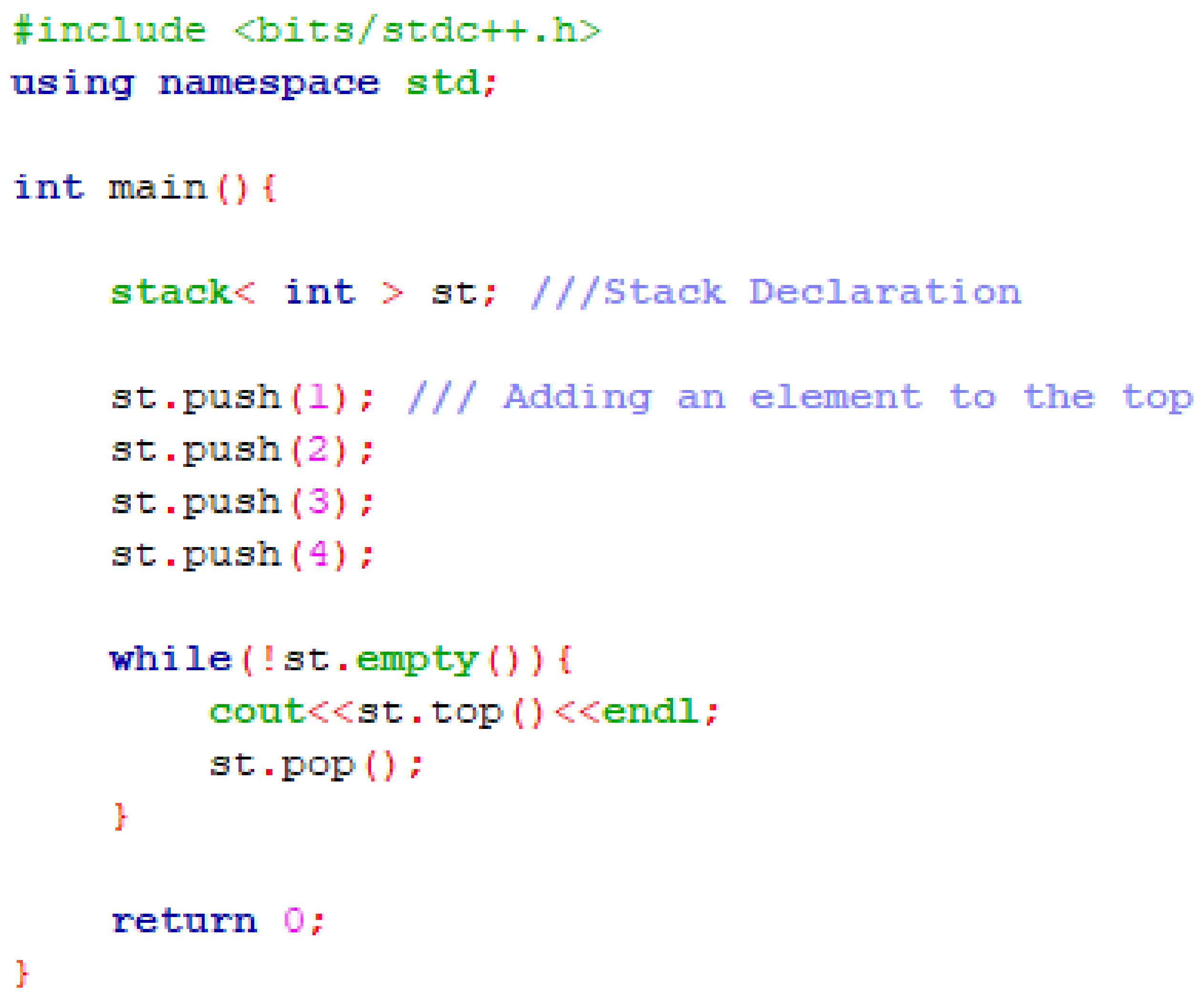
```
#include <bits/stdc++.h>
using namespace std;

int main(){

    vector< int > v; ///Vector Declaration

    v.push_back(1); /// Adding an element to the end
    v.push_back(2);
    v.push_back(3);
    v.push_back(4);

    v.pop_back();

    for(int i=0; i<v.size(); i++){   /// traversing the array
        cout<<v[i]<<endl;    /// accessing i'th element
    }

    cout<<v[0]<<endl;
    cout<<v.back()<<endl;


    return 0;
}
```

STACK

Member Function:

- push()
- pop()
- top()
- empty()

STACK



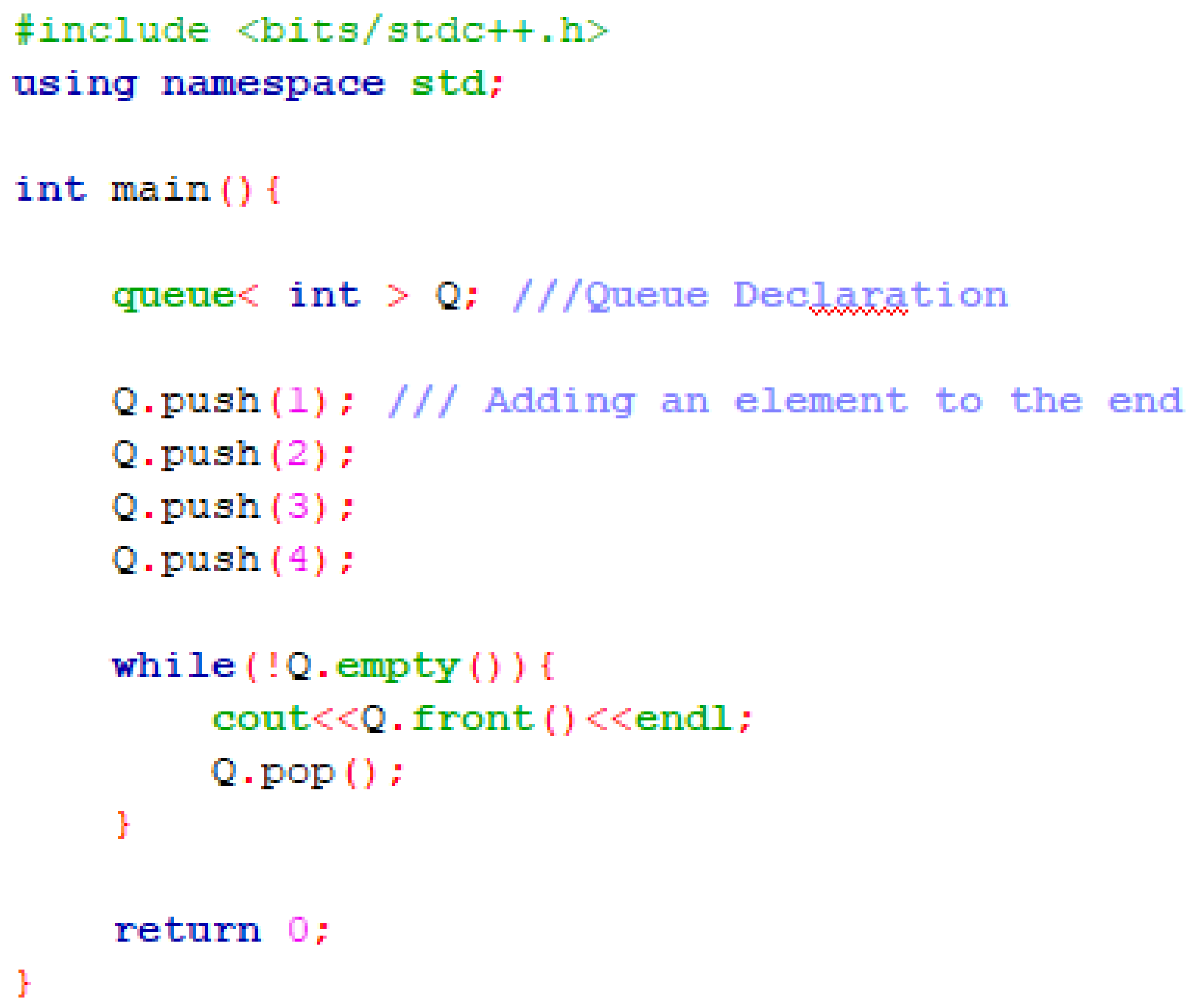
```
#include <bits/stdc++.h>
using namespace std;

int main(){

    stack< int > st; ///Stack Declaration

    st.push(1); /// Adding an element to the top
    st.push(2);
    st.push(3);
    st.push(4);

    while(!st.empty()){
        cout<<st.top()<<endl;
        st.pop();
    }

    return 0;
}
```

QUEUE

Member Function:

- push()
- pop()
- front()
- empty()

QUEUE



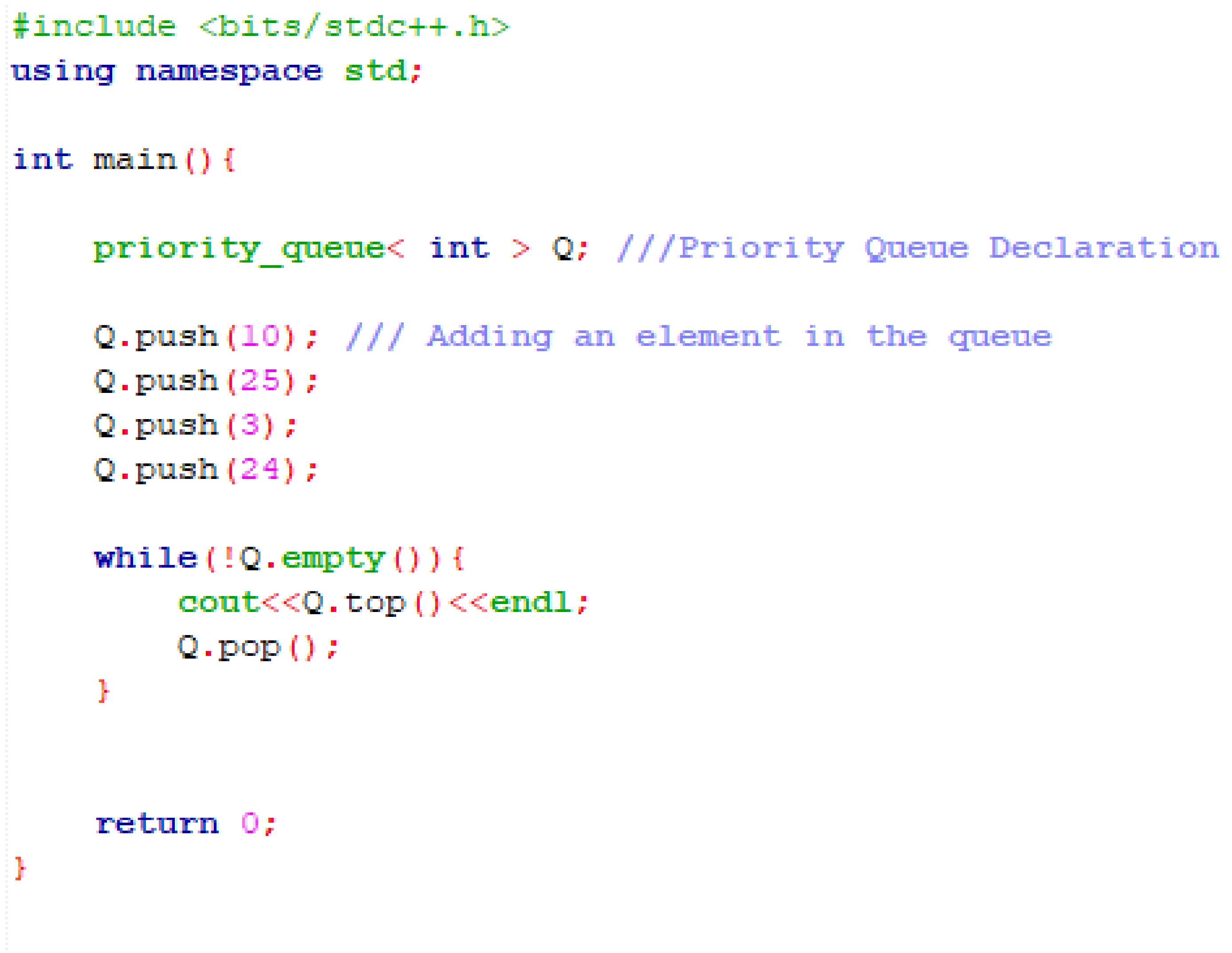
```
#include <bits/stdc++.h>
using namespace std;

int main(){

    queue< int > Q; ///Queue Declaration

    Q.push(1); /// Adding an element to the end
    Q.push(2);
    Q.push(3);
    Q.push(4);

    while(!Q.empty()){
        cout<<Q.front()<<endl;
        Q.pop();
    }

    return 0;
}
```

PRIORITY QUEUE

Member Function:

- push()
- pop()
- top()
- empty()

PRIORITY QUEUE



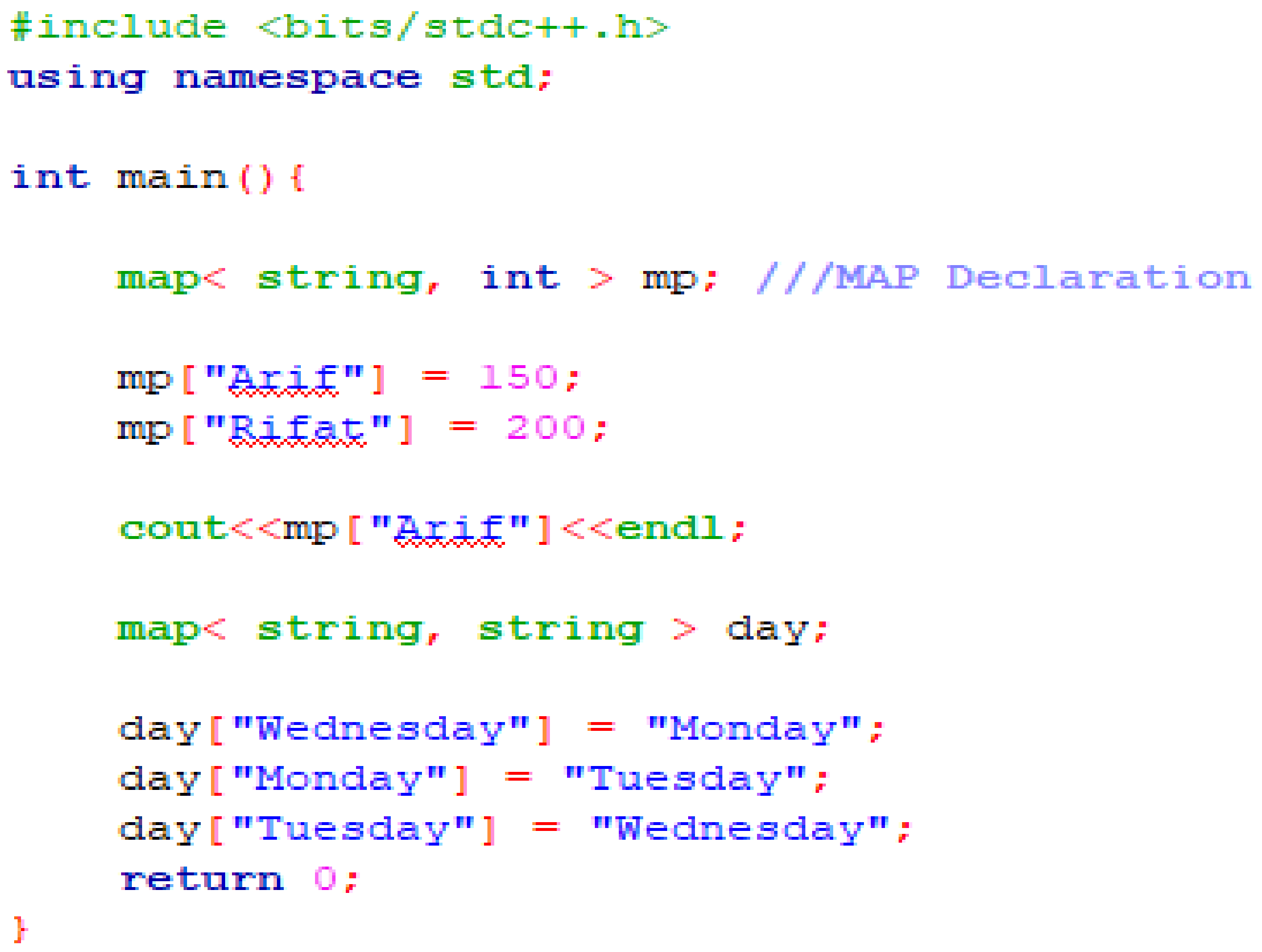
```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){

    priority_queue< int > Q; ///Priority Queue Declaration

    Q.push(10); /// Adding an element in the queue
    Q.push(25);
    Q.push(3);
    Q.push(24);

    while(!Q.empty()){
        cout<<Q.top()<<endl;
        Q.pop();
    }


    return 0;
}
```

MAP

Member Function:

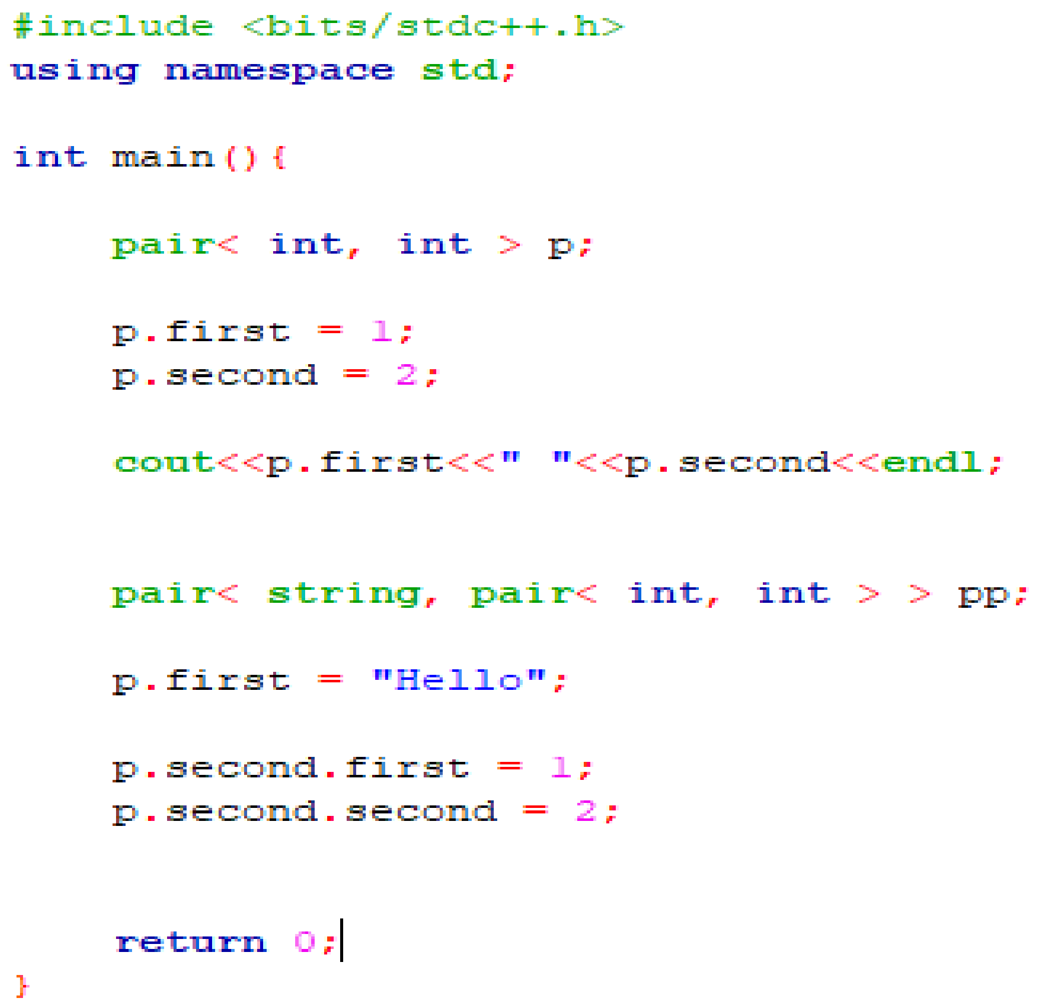
- begin()
- end()
- size()
- empty()
- clear()

MAP



```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){

    map< string, int > mp; ///MAP Declaration

    mp["Arif"] = 150;
    mp["Rifat"] = 200;

    cout<<mp["Arif"]<<endl;

    map< string, string > day;

    day["Wednesday"] = "Monday";
    day["Monday"] = "Tuesday";
    day["Tuesday"] = "Wednesday";
    return 0;
}
```

PAIR



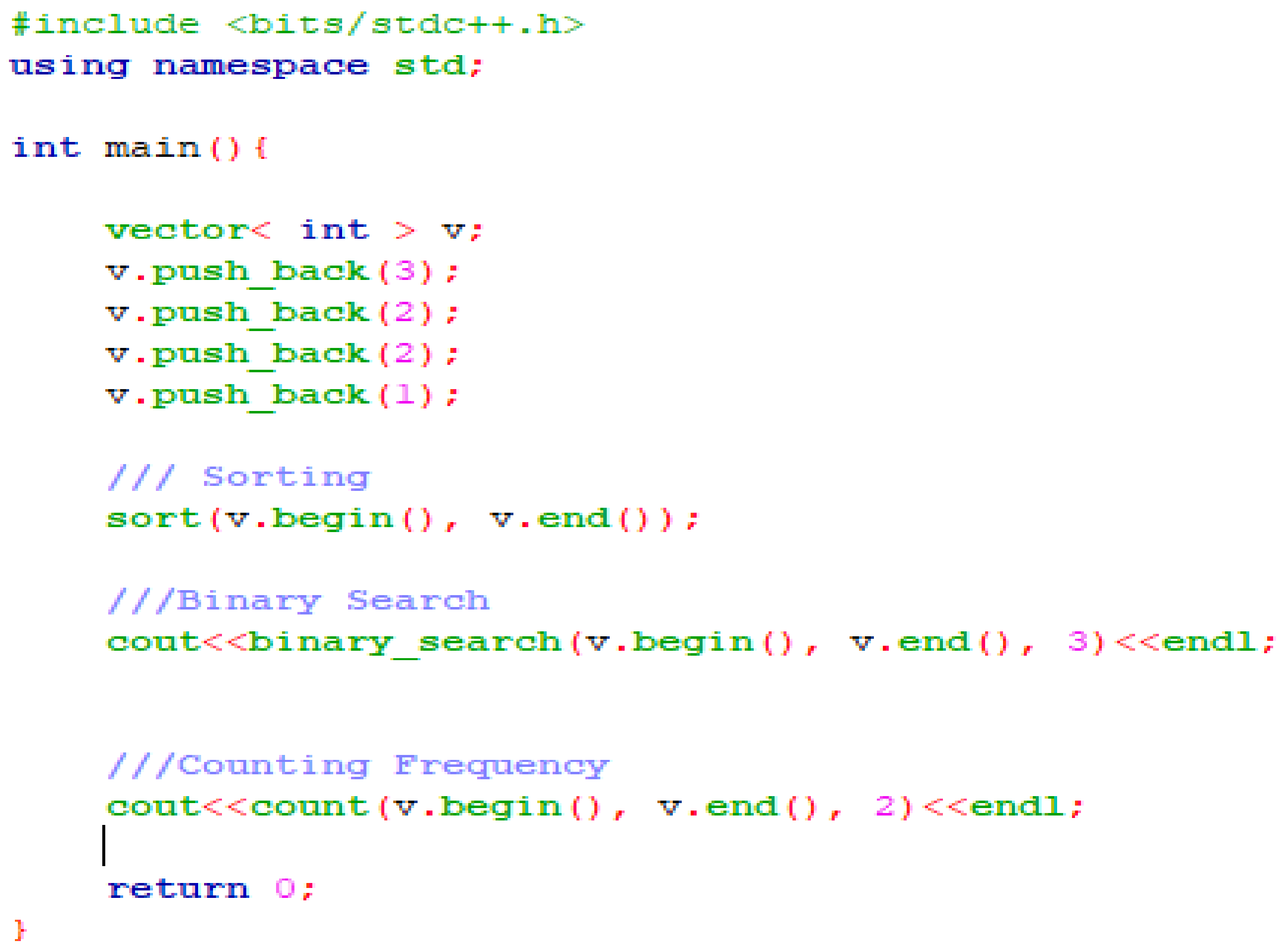
```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){

    pair< int, int > p;

    p.first = 1;
    p.second = 2;

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


    pair< string, pair< int, int > > pp;

    p.first = "Hello";

    p.second.first = 1;
    p.second.second = 2;


    return 0;
}
```

ALGORITHM

Function:

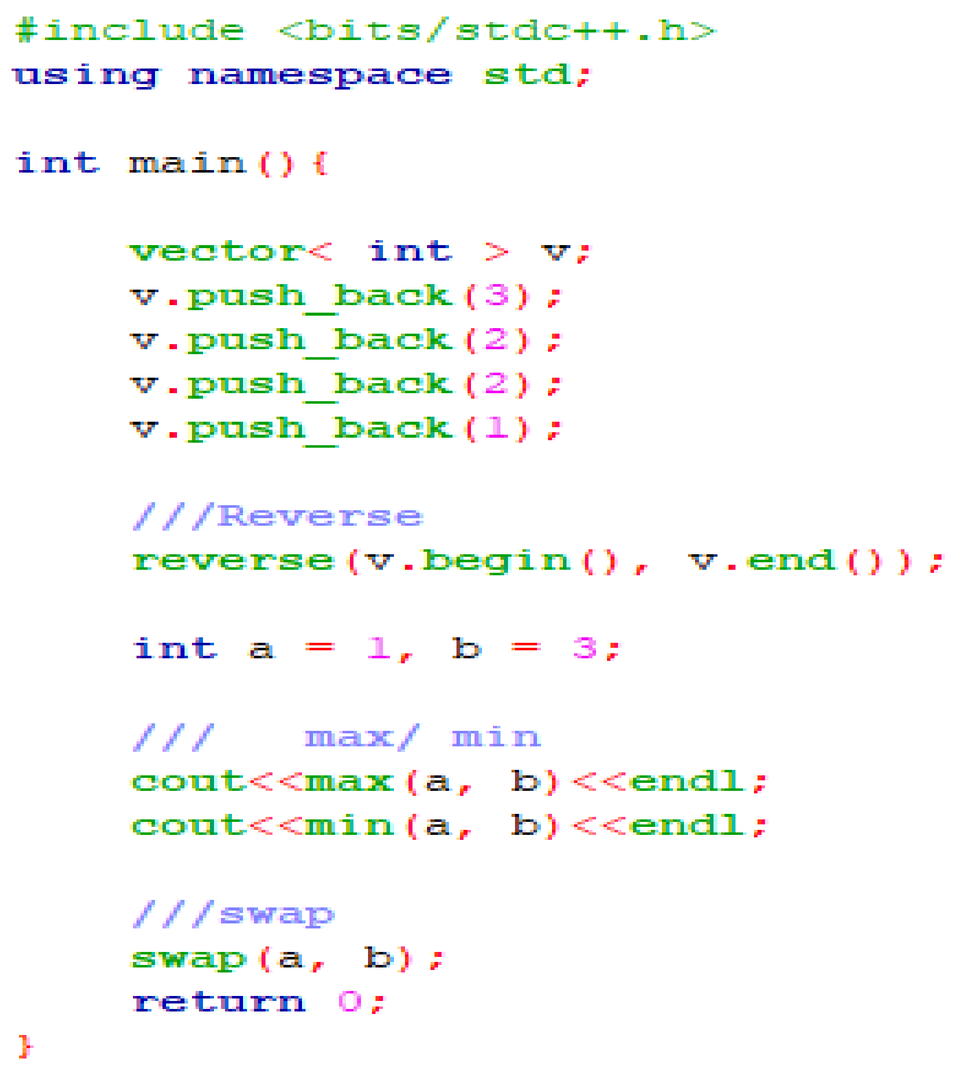
- sort()
- binary_search()
- lower_bound()
- upper_bound()
- count()
- swap()
- reverse()
- max() / min()
- max_element() / min_element()
- accumulate()

ALGORITHM



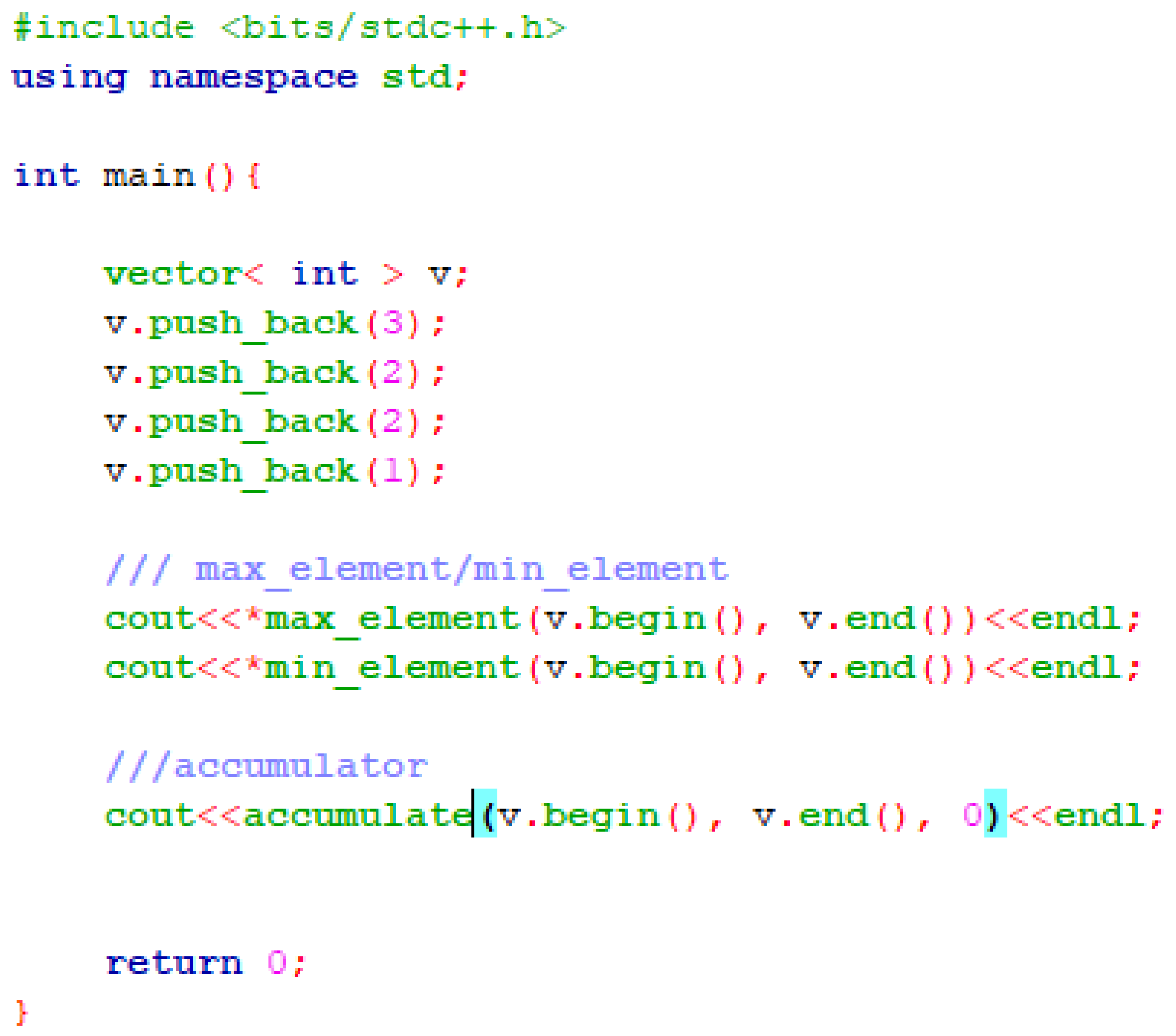
```
#include <bits/stdc++.h>
using namespace std;

int main(){

    vector< int > v;
    v.push_back(3);
    v.push_back(2);
    v.push_back(2);
    v.push_back(1);

    /// Sorting
    sort(v.begin(), v.end());

    ///Binary Search
    cout<<binary_search(v.begin(), v.end(), 3)<<endl;


    ///Counting Frequency
    cout<<count(v.begin(), v.end(), 2)<<endl;

    return 0;
}
```

ALGORITHM



```
#include <bits/stdc++.h>
using namespace std;

int main(){

    vector< int > v;
    v.push_back(3);
    v.push_back(2);
    v.push_back(2);
    v.push_back(1);

    ///Reverse
    reverse(v.begin(), v.end());

    int a = 1, b = 3;

    ///   max/ min
    cout<<max(a, b)<<endl;
    cout<<min(a, b)<<endl;

    ///swap
    swap(a, b);
    return 0;
}
```

ALGORITHM



```
#include <bits/stdc++.h>
using namespace std;

int main(){

    vector< int > v;
    v.push_back(3);
    v.push_back(2);
    v.push_back(2);
    v.push_back(1);

    /// max_element/min_element
    cout<<*max_element(v.begin(), v.end())<<endl;
    cout<<*min_element(v.begin(), v.end())<<endl;

    ///accumulator
    cout<<accumulate(v.begin(), v.end(), 0)<<endl;


    return 0;
}
```

?????