Standard Template Library

pair

Pair is used to combine together two values which may be different in type. Pair provides a way to store two heterogeneous objects as a .single unit

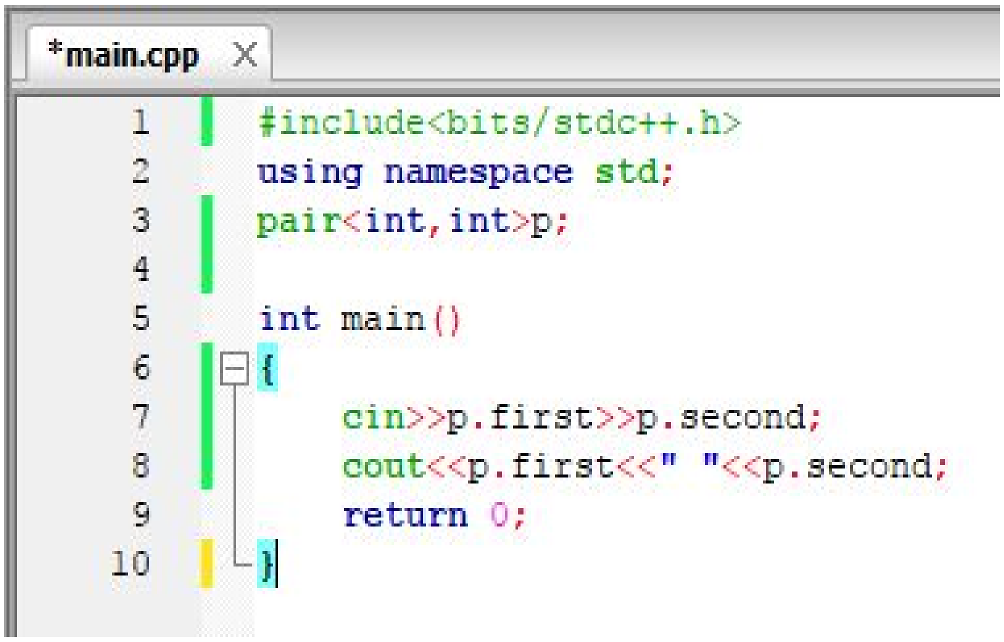
The first element is referenced as ‘first’ and the second element as .(‘second’ and the order is fixed (first, second

To access the elements, we use variable name followed by dot .operator followed by the keyword first or second

:Syntax

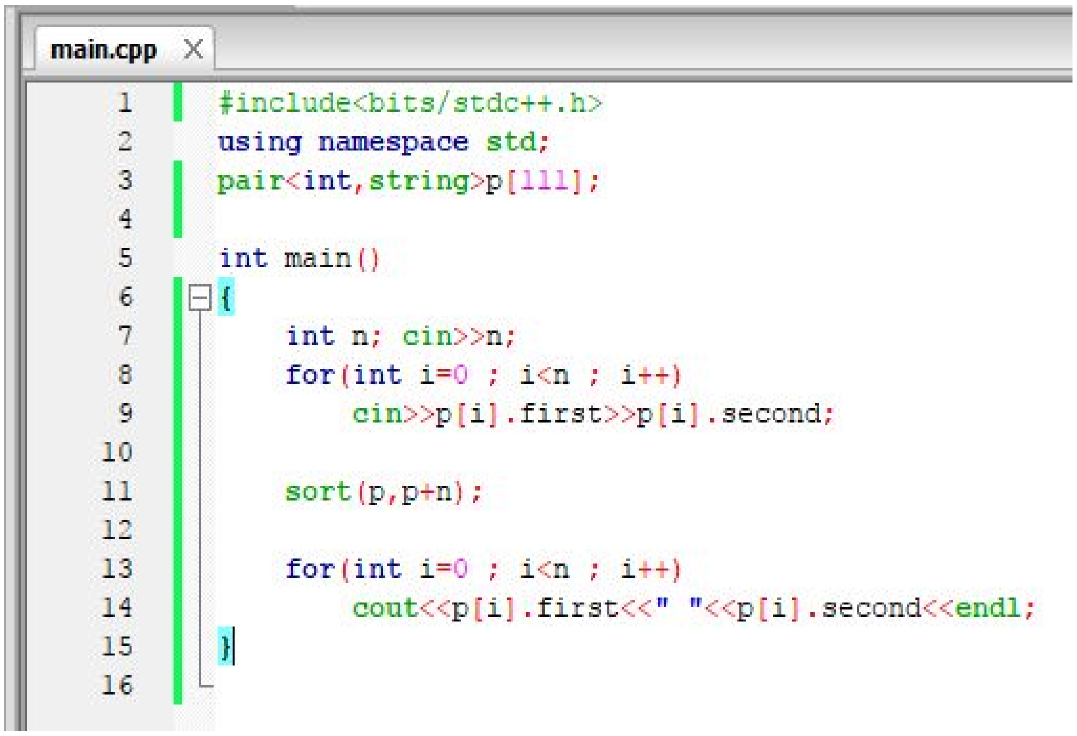
```
pair < data_type1, data_type2 > Pair_name;
```

:Example



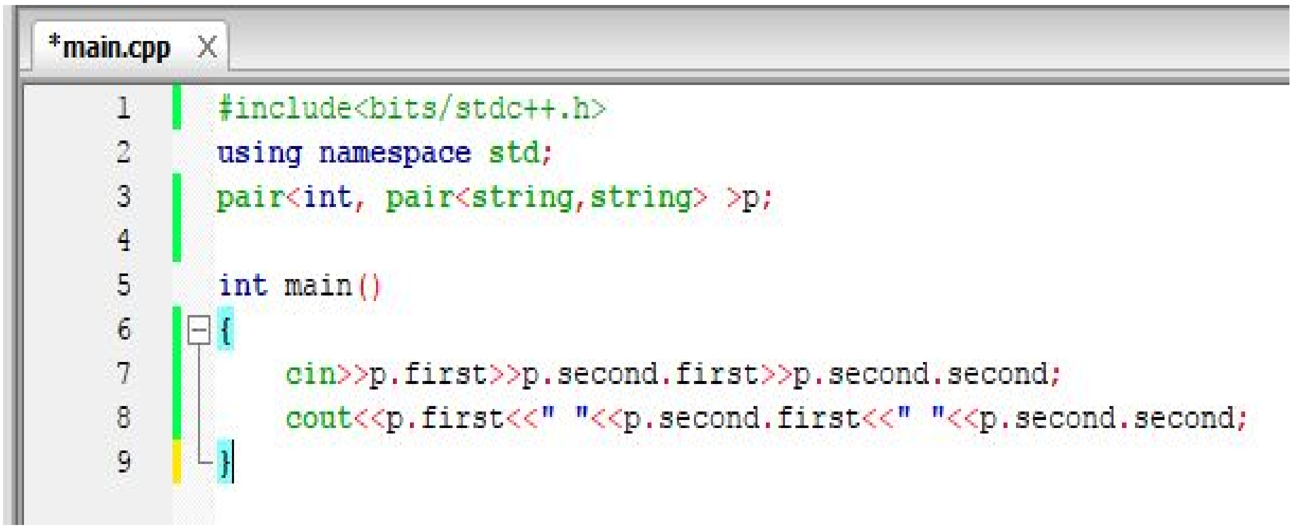
*main.cpp

```
#include<bits/stdc++.h>
using namespace std;
pair<int,int>p;

int main()
{
    cin>>p.first>>p.second;
    cout<<p.first<<" "<<p.second;
    return 0;
}
```



main.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
pair<int,string>p[111];

int main()
{
    int n; cin>>n;
    for(int i=0 ; i<n ; i++)
        cin>>p[i].first>>p[i].second;

    sort(p,p+n);

    for(int i=0 ; i<n ; i++)
        cout<<p[i].first<<" "<<p[i].second<<endl;
}
```



*main.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
pair<int, pair<string,string> >p;

int main()
{
    cin>>p.first>>p.second.first>>p.second.second;
    cout<<p.first<<" "<<p.second.first<<" "<<p.second.second;
}
```

vector

Vectors are same as dynamic arrays with the ability to resize .itself automatically when an element is inserted or deleted

:Syntax

```
vector< data_type > vector_name;
```

:Modifiers

push_back (value); Adds a new element at the end of the vector, after .its current last element

pop_back (); Removes the last element in the vector, effectively .reducing the container size by one

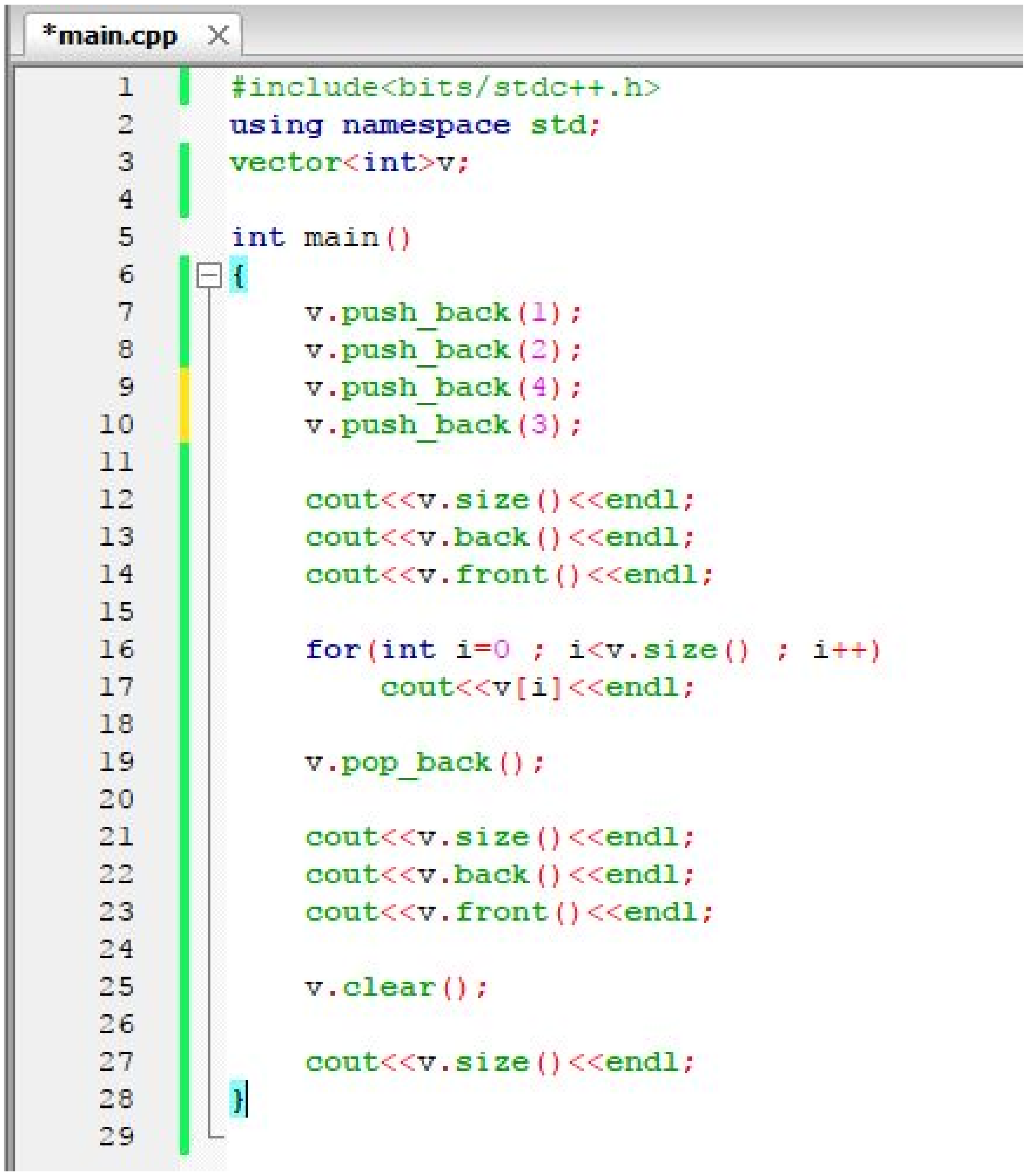
.size (); Returns the number of elements in the vector

clear (); Removes all elements from the vector (which are destroyed), .leaving the container with a size of 0

.back (); Returns a reference to the last element in the vector

.front (); Returns a reference to the first element in the vector

:Example



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

int main()
{
    v.push_back(1);
    v.push_back(2);
    v.push_back(4);
    v.push_back(3);

    cout<<v.size()<<endl;
    cout<<v.back()<<endl;
    cout<<v.front()<<endl;

    for(int i=0 ; i<v.size() ; i++)
        cout<<v[i]<<endl;

    v.pop_back();

    cout<<v.size()<<endl;
    cout<<v.back()<<endl;
    cout<<v.front()<<endl;

    v.clear();

    cout<<v.size()<<endl;
}
```

queue

queues are a type of container adaptor, specifically designed to operate in a FIFO context (first-in first-out), where elements are inserted into one end of the .container and extracted from the other

:Syntax

```
queue< data_type > queue_name;
```

:Modifiers

push (value); Inserts a new element at the end of the queue, after its .current last element

pop (); Removes the next element in the queue, effectively reducing .its size by one

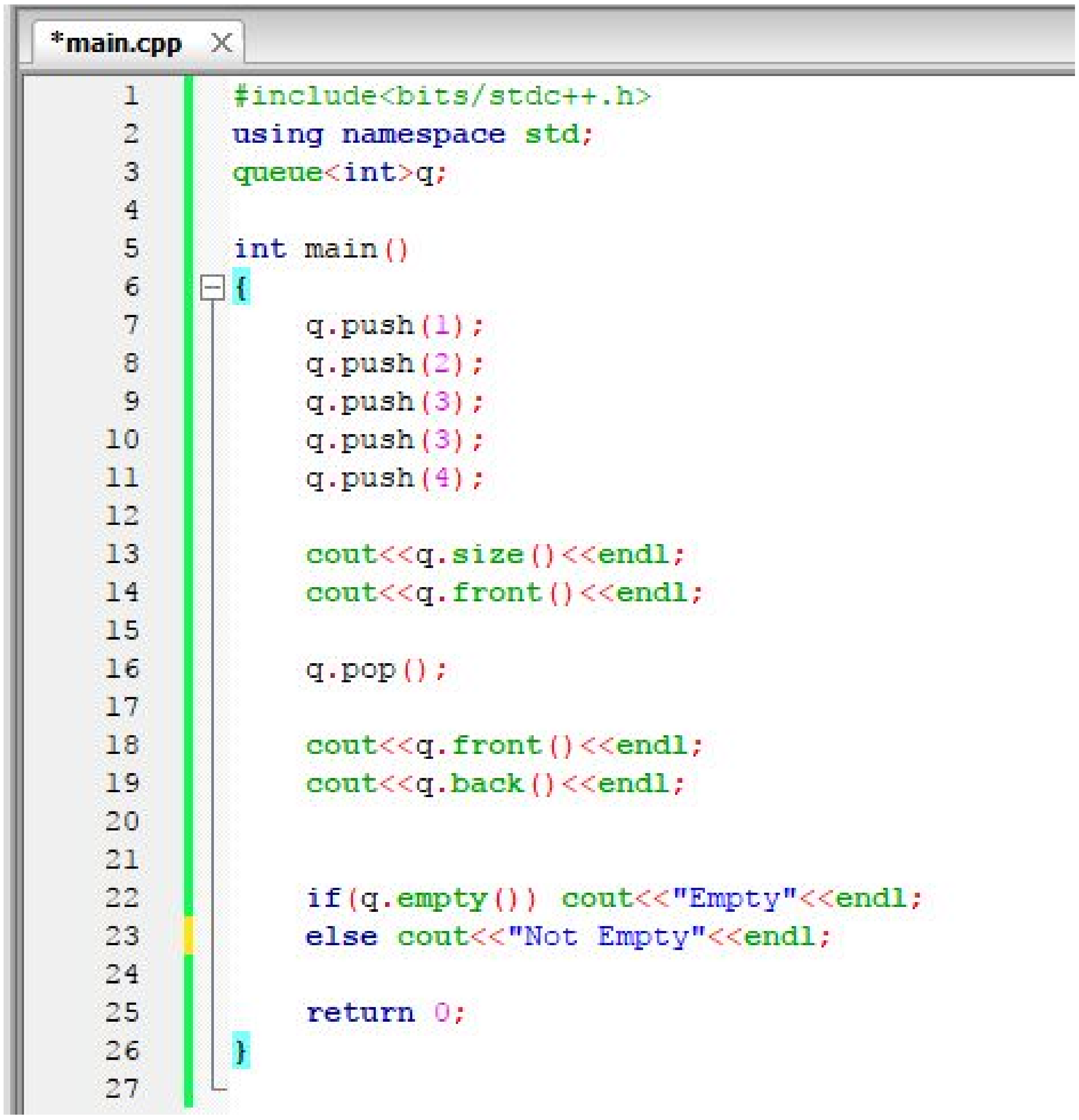
.front (); Returns a reference to the *next element* in the queue

.size (); Returns the number of elements in the queue

empty (); Returns whether the queue is empty: i.e. whether .its size is *zero*

back (); Returns a reference to the last element in the queue. This is the "newest" element in the queue (i.e. the last element pushed into .(the queue

:Example



*main.cpp

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

int main()
{
    q.push(1);
    q.push(2);
    q.push(3);
    q.push(3);
    q.push(4);

    cout<<q.size()<<endl;
    cout<<q.front()<<endl;

    q.pop();

    cout<<q.front()<<endl;
    cout<<q.back()<<endl;


    if(q.empty()) cout<<"Empty"<<endl;
    else cout<<"Not Empty"<<endl;

    return 0;
}

```

set

Sets are containers that store unique elements following .a specific order

:Syntax

```
set < data_type > set_name;
```

:Modifiers

insert (value); Extends the container by inserting new elements, effectively increasing the container size by the number of elements .inserted

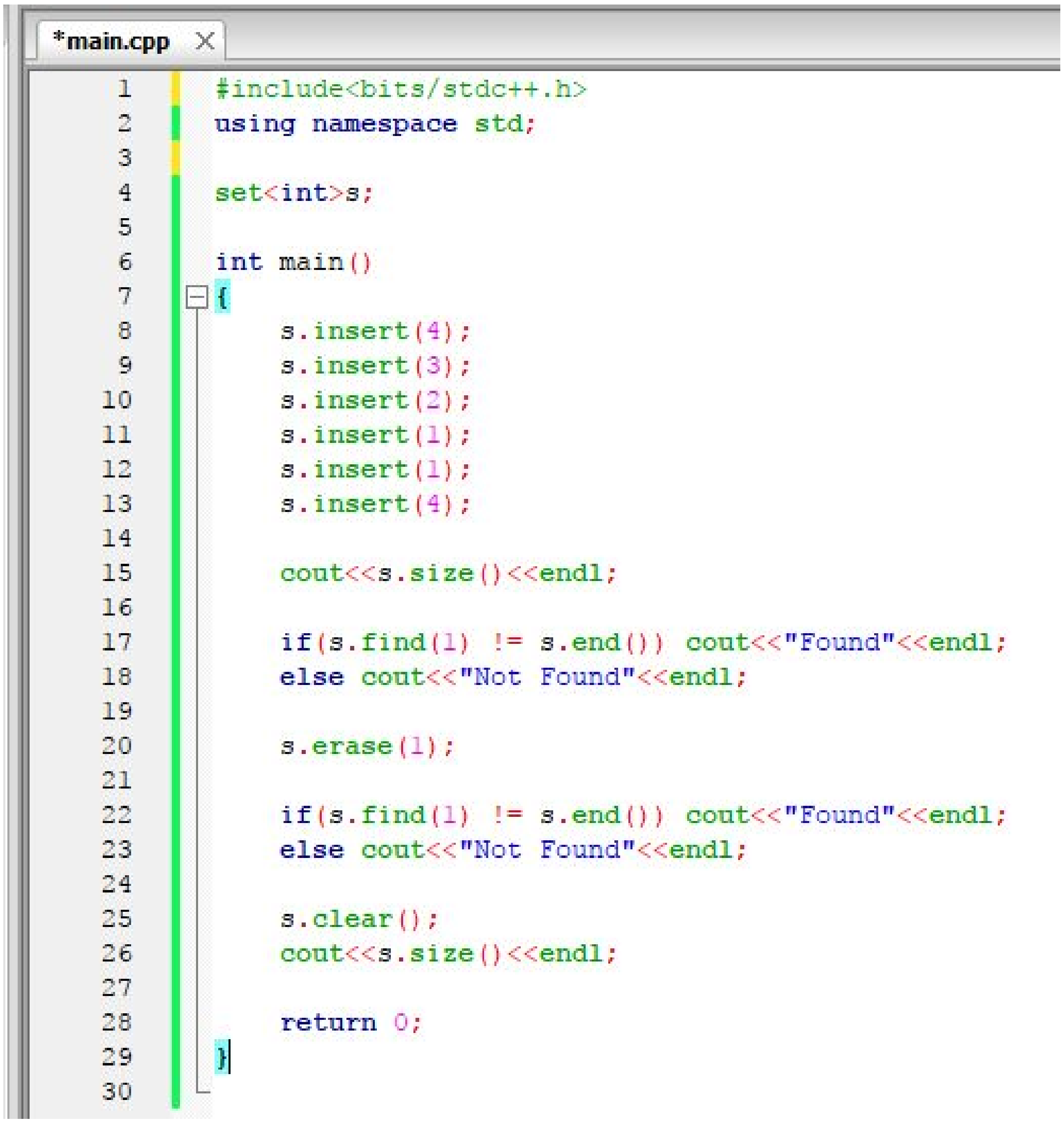
.**erase (value);** Removes from the set container a single element

.**size ();** Returns the number of elements in the set container

clear (); Removes all elements from the set container (which are .destroyed), leaving the container with a size of 0

find (value); Searches the container for an element equivalent to value and returns an iterator to it if found, otherwise it returns an iterator to .set::end

:Example


*main.cpp

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

set<int>s;

int main()
{
    s.insert(4);
    s.insert(3);
    s.insert(2);
    s.insert(1);
    s.insert(1);
    s.insert(4);

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

    if(s.find(1) != s.end()) cout<<"Found"<<endl;
    else cout<<"Not Found"<<endl;

    s.erase(1);

    if(s.find(1) != s.end()) cout<<"Found"<<endl;
    else cout<<"Not Found"<<endl;

    s.clear();
    cout<<s.size()<<endl;

    return 0;
}
```

map

Maps are associative containers that store elements formed by a combination of a key value and a mapped value, following a .specific order

In a map, the key values are generally used to sort and uniquely identify the elements, while the mapped values store the .content associated to this key

:Syntax

```
map< data_type1(key), data_type2(value) > map_name;
```

:Modifiers

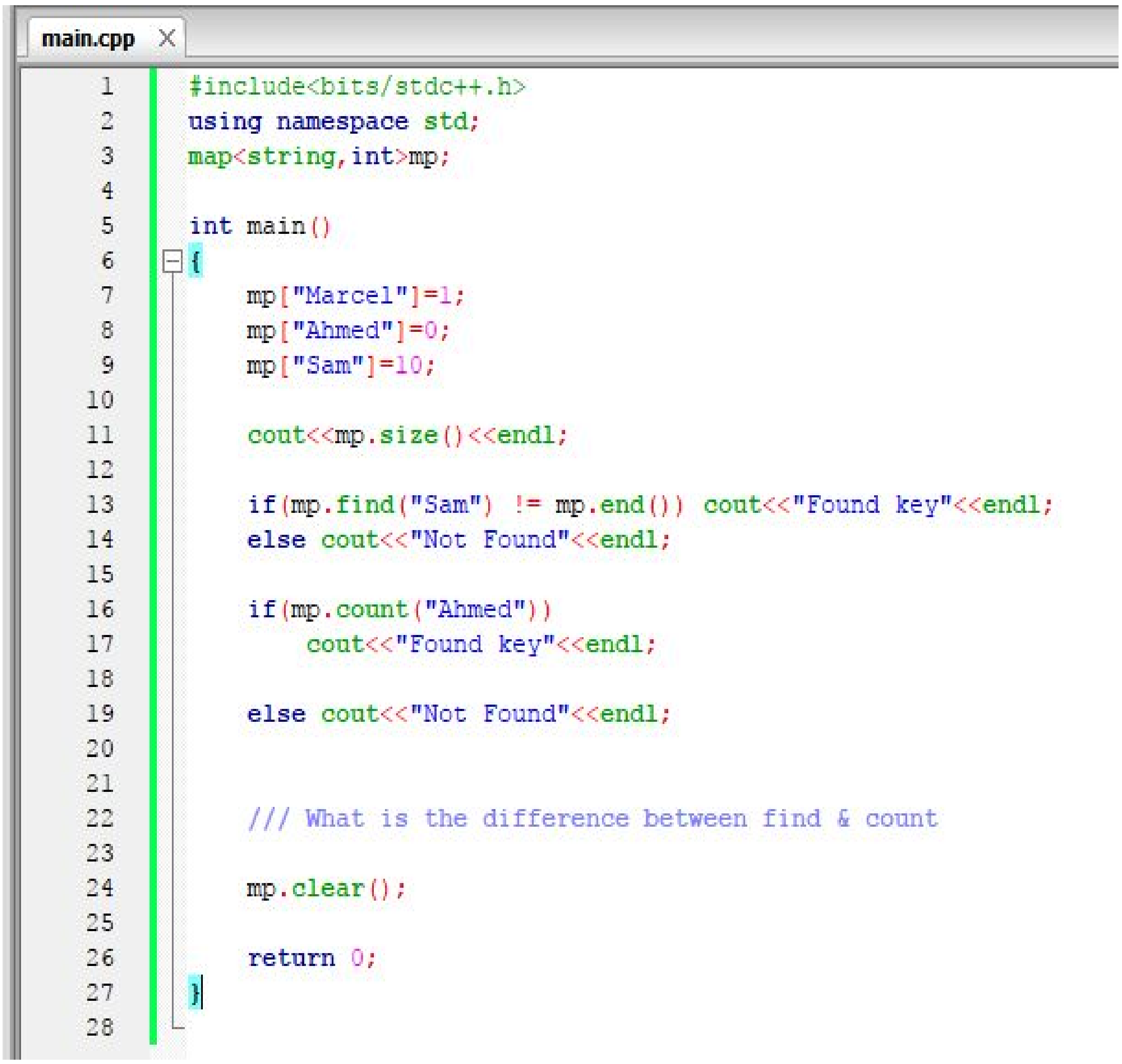
.size (); Returns the number of elements in the map container

clear (); Removes all elements from the map container (which are .destroyed), leaving the container with a size of 0

find (key); Searches the container for an element with a key equivalent to k and returns an iterator to it if found, otherwise it returns an .iterator to map::end

count (key); Searches the container for elements with a key equivalent .to *k* and returns the number of matches

:Example


main.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
map<string,int>mp;

int main()
{
    mp["Marcel"]=1;
    mp["Ahmed"]=0;
    mp["Sam"]=10;

    cout<<mp.size()<<endl;

    if(mp.find("Sam") != mp.end()) cout<<"Found key"<<endl;
    else cout<<"Not Found"<<endl;

    if(mp.count("Ahmed"))
        cout<<"Found key"<<endl;

    else cout<<"Not Found"<<endl;


    /// What is the difference between find & count

    mp.clear();

    return 0;
}

```

Iterators in C++ STL

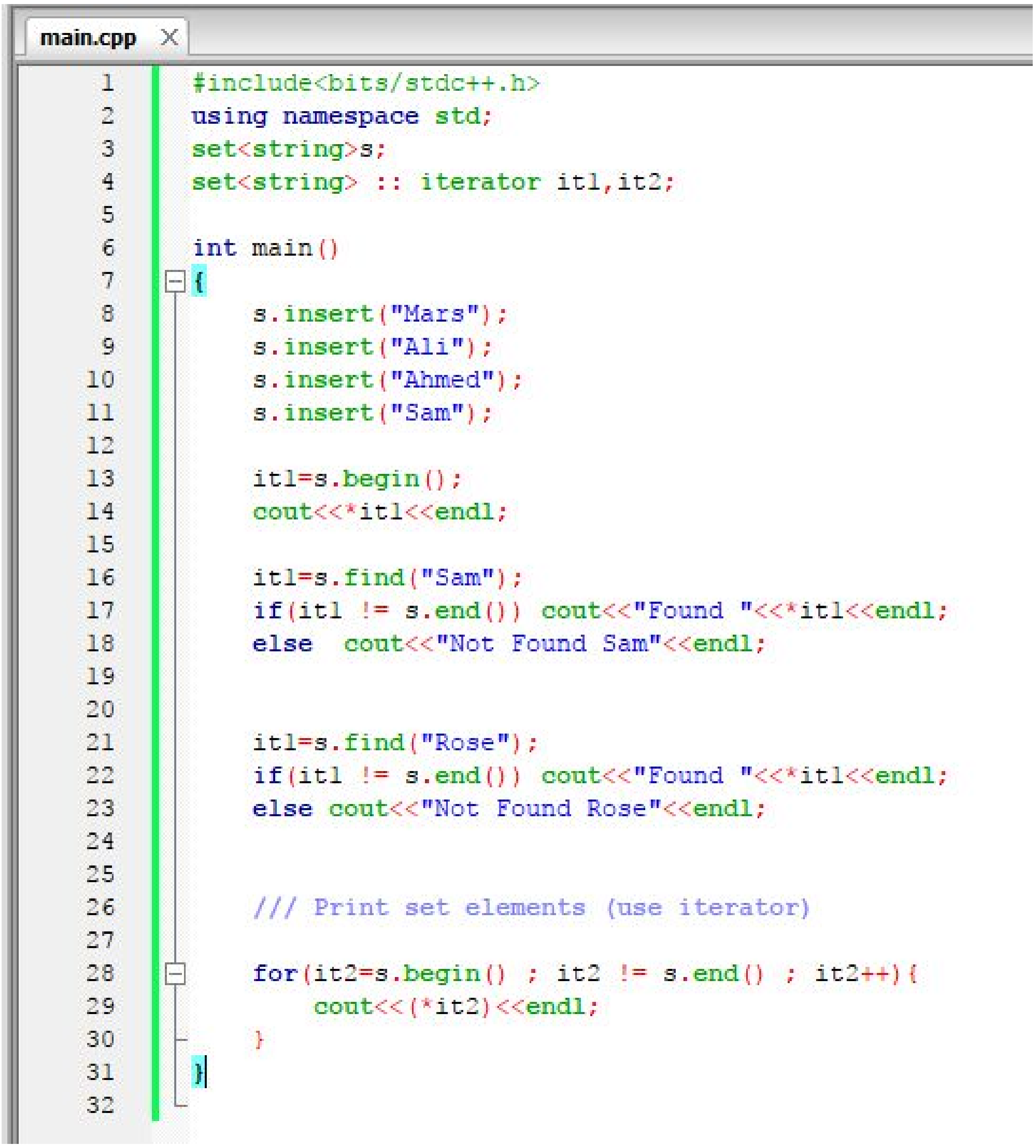
Iterators are used to point at the memory addresses of STL containers. They are primarily used in sequence of numbers, characters etc. They reduce the .complexity and execution time of program

:Operations of iterators

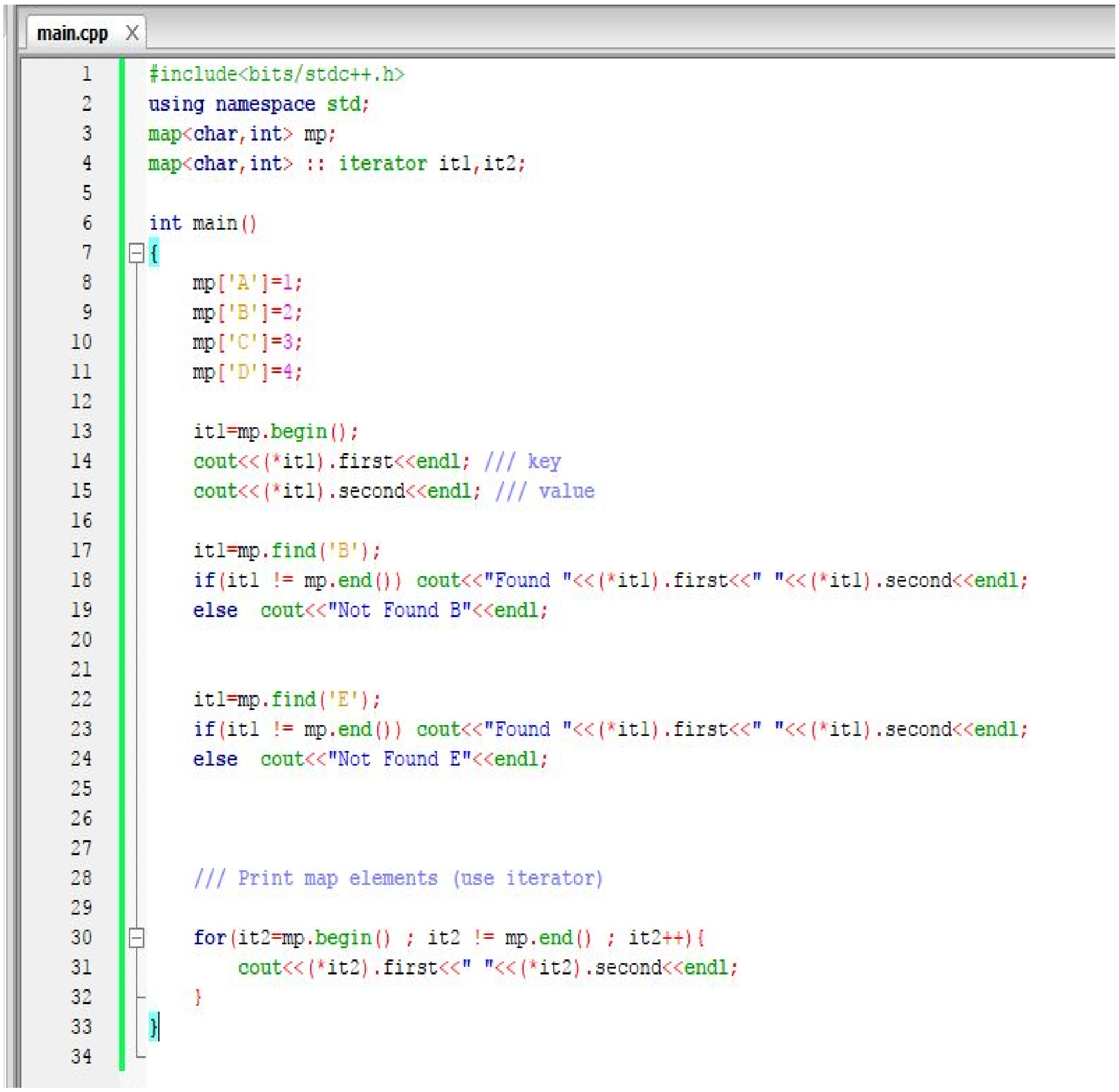
begin (): This function is used to return the **beginning position** of **.1** .the container

end (): This function is used to return the **end position** of the **.2** .container

:Example


main.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
set<string>s;
set<string> :: iterator it1,it2;

int main()
{
    s.insert("Mars");
    s.insert("Ali");
    s.insert("Ahmed");
    s.insert("Sam");

    it1=s.begin();
    cout<<*it1<<endl;

    it1=s.find("Sam");
    if(it1 != s.end()) cout<<"Found "<<*it1<<endl;
    else  cout<<"Not Found Sam"<<endl;


    it1=s.find("Rose");
    if(it1 != s.end()) cout<<"Found "<<*it1<<endl;
    else cout<<"Not Found Rose"<<endl;


    /// Print set elements (use iterator)

    for(it2=s.begin() ; it2 != s.end() ; it2++){
        cout<<(*it2)<<endl;
    }
}
```

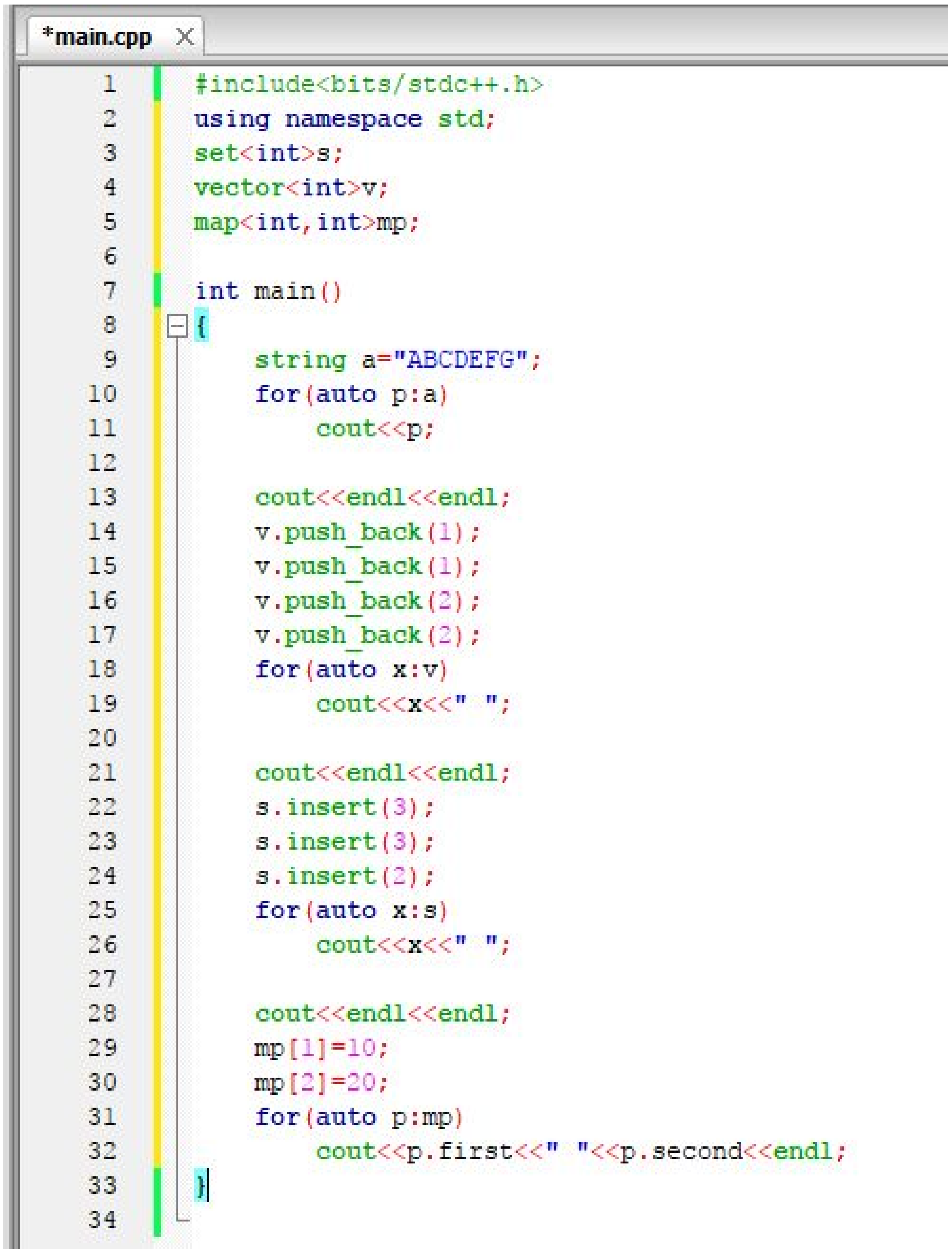

main.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
map<char,int> mp;
map<char,int> :: iterator it1,it2;

int main()
{
    mp['A']=1;
    mp['B']=2;
    mp['C']=3;
    mp['D']=4;

    it1=mp.begin();
    cout<<(*it1).first<<endl; /// key
    cout<<(*it1).second<<endl; /// value

    it1=mp.find('B');
    if(it1 != mp.end()) cout<<"Found "<<(*it1).first<<" "<<(*it1).second<<endl;
    else  cout<<"Not Found B"<<endl;


    it1=mp.find('E');
    if(it1 != mp.end()) cout<<"Found "<<(*it1).first<<" "<<(*it1).second<<endl;
    else  cout<<"Not Found E"<<endl;



    /// Print map elements (use iterator)

    for(it2=mp.begin() ; it2 != mp.end() ; it2++){
        cout<<(*it2).first<<" "<<(*it2).second<<endl;
    }
}
```

for each in C++11

For each loop is used to access elements of an array quickly without performing initialization, testing and increment/decrement. The working of for each loops is to do something for every element rather than doing .something n times

:Example



*main.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
set<int>s;
vector<int>v;
map<int,int>mp;

int main()
{
    string a="ABCDEFG";
    for(auto p:a)
        cout<<p;

    cout<<endl<<endl;
    v.push_back(1);
    v.push_back(1);
    v.push_back(2);
    v.push_back(2);
    for(auto x:v)
        cout<<x<<" ";

    cout<<endl<<endl;
    s.insert(3);
    s.insert(3);
    s.insert(2);
    for(auto x:s)
        cout<<x<<" ";

    cout<<endl<<endl;
    mp[1]=10;
    mp[2]=20;
    for(auto p:mp)
        cout<<p.first<<" "<<p.second<<endl;
}
```

End