

# 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

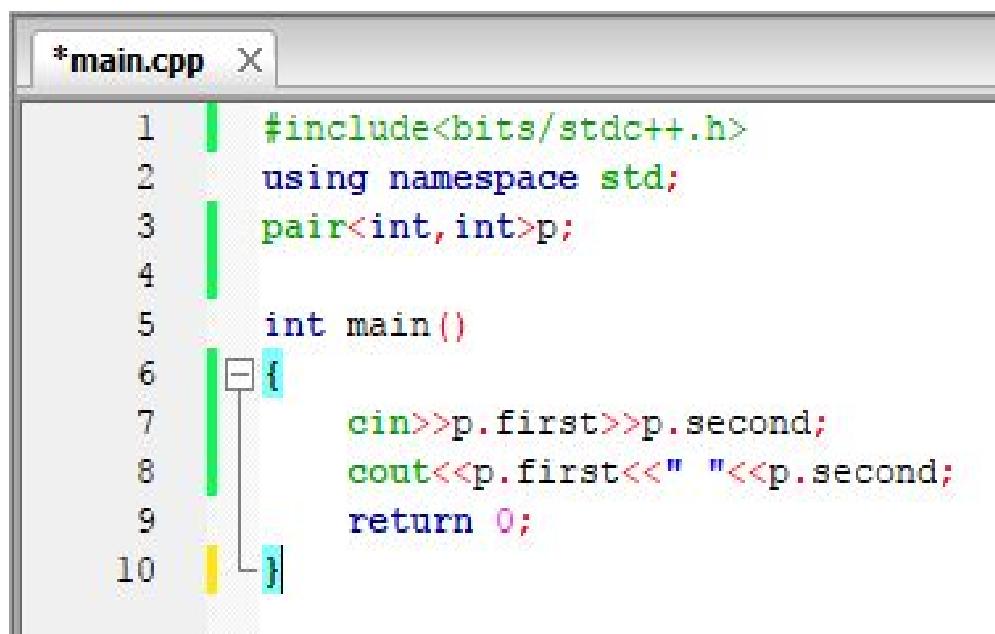
The first element is referenced as 'first' and the second element as .'second' and the order is fixed (first, second)

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 X
1 #include<bits/stdc++.h>
2 using namespace std;
3 pair<int,int>p;
4
5 int main()
6 {
7     cin>>p.first>>p.second;
8     cout<<p.first<<" "<<p.second;
9     return 0;
10 }
```

main.cpp X

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 pair<int, string>p[111];
4
5 int main()
6 {
7     int n; cin>>n;
8     for(int i=0 ; i<n ; i++)
9         cin>>p[i].first>>p[i].second;
10
11     sort(p,p+n);
12
13     for(int i=0 ; i<n ; i++)
14         cout<<p[i].first<<" "<<p[i].second<<endl;
15 }
16
```

\*main.cpp X

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 pair<int, pair<string, string> >p;
4
5 int main()
6 {
7     cin>>p.first>>p.second.first>>p.second.second;
8     cout<<p.first<<" "<<p.second.first<<" "<<p.second.second;
9 }
```

# 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

The screenshot shows a code editor window titled "\*main.cpp". The code is a C++ program demonstrating the use of a std::vector. It includes #include, using namespace std;, and defines a main() function that pushes back four integers (1, 2, 4, 3) into a vector v. It then prints the size, back, and front of the vector. A for loop prints each element of the vector. Finally, it pops back the last element, prints the size, back, and front again, and then clears the vector before printing its size once more.

```
#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

The screenshot shows a code editor window titled "\*main.cpp X". The code is a C++ program demonstrating the use of a queue. It includes #include, using namespace std, and queue<int>q declarations. The main function pushes integers 1 through 4 into the queue, then prints its size and front element. It then pops the first element and prints the front and back elements again. Finally, it checks if the queue is empty and prints "Empty" or "Not Empty". The code is color-coded for readability.

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 queue<int>q;
4
5 int main()
6 {
7     q.push(1);
8     q.push(2);
9     q.push(3);
10    q.push(3);
11    q.push(4);
12
13    cout<<q.size()<<endl;
14    cout<<q.front()<<endl;
15
16    q.pop();
17
18    cout<<q.front()<<endl;
19    cout<<q.back()<<endl;
20
21
22    if(q.empty()) cout<<"Empty"<<endl;
23    else cout<<"Not Empty"<<endl;
24
25    return 0;
26
27 }
```

# 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

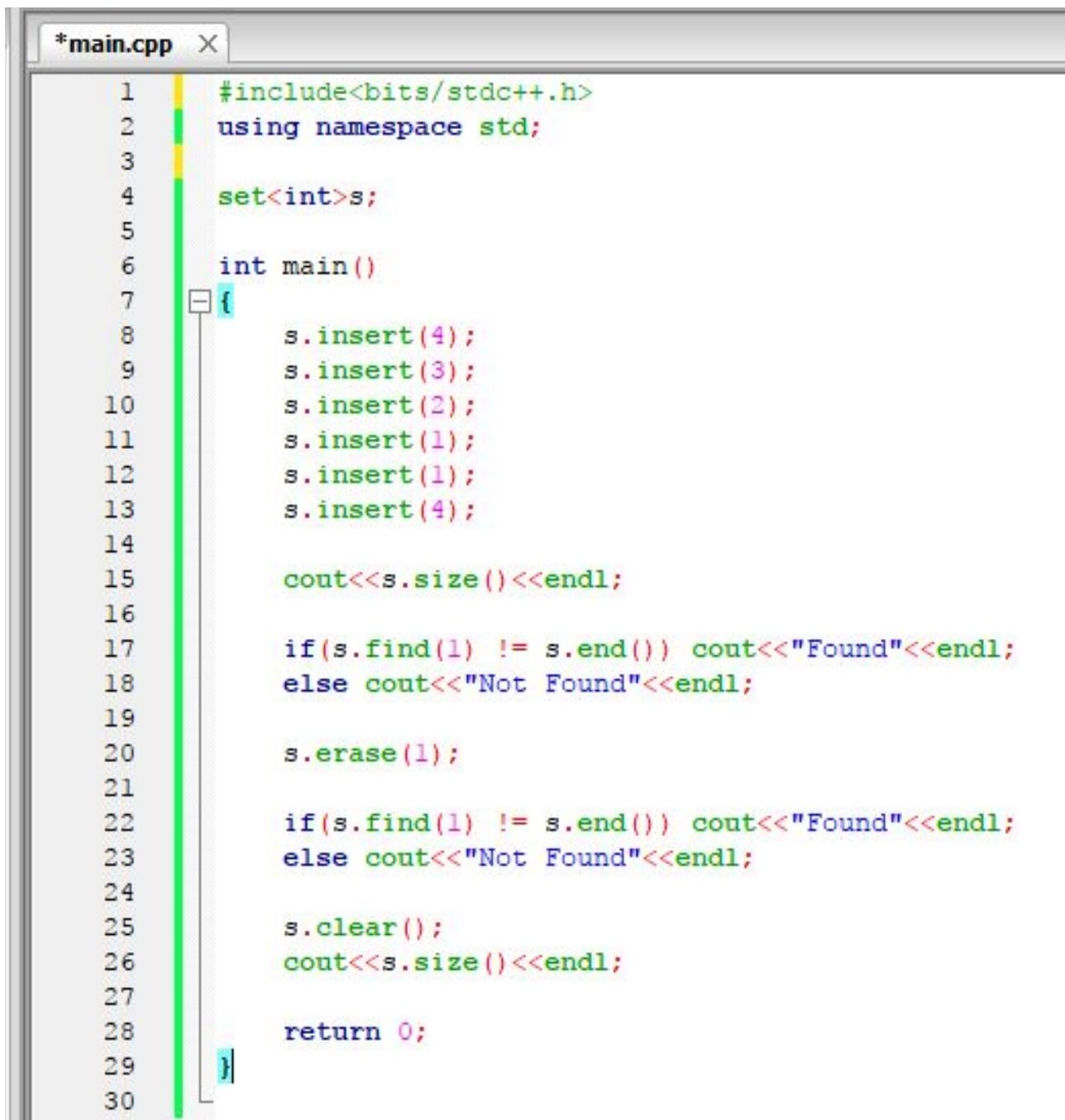
**.erase (value);** Removes from the set container a single element

**.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



The screenshot shows a code editor window with the file name "\*main.cpp" in the title bar. The code itself is as follows:

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 set<int>s;
5
6 int main()
7 {
8     s.insert(4);
9     s.insert(3);
10    s.insert(2);
11    s.insert(1);
12    s.insert(1);
13    s.insert(4);
14
15    cout<<s.size()<<endl;
16
17    if(s.find(1) != s.end()) cout<<"Found"<<endl;
18    else cout<<"Not Found"<<endl;
19
20    s.erase(1);
21
22    if(s.find(1) != s.end()) cout<<"Found"<<endl;
23    else cout<<"Not Found"<<endl;
24
25    s.clear();
26    cout<<s.size()<<endl;
27
28    return 0;
29
30 }
```

# 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 X
1 #include<bits/stdc++.h>
2 using namespace std;
3 map<string,int>mp;
4
5 int main()
6 {
7     mp["Marcel"]=1;
8     mp["Ahmed"]=0;
9     mp["Sam"]=10;
10
11     cout<<mp.size()<<endl;
12
13     if(mp.find("Sam") != mp.end()) cout<<"Found key"<<endl;
14     else cout<<"Not Found"<<endl;
15
16     if(mp.count("Ahmed"))
17         cout<<"Found key"<<endl;
18
19     else cout<<"Not Found"<<endl;
20
21
22     // What is the difference between find & count
23
24     mp.clear();
25
26     return 0;
27 }
28
```

# 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 X

1 #include<bits/stdc++.h>
2 using namespace std;
3 set<string>s;
4 set<string> :: iterator it1,it2;
5
6 int main()
7 {
8     s.insert("Mars");
9     s.insert("Ali");
10    s.insert("Ahmed");
11    s.insert("Sam");
12
13    it1=s.begin();
14    cout<<*it1<<endl;
15
16    it1=s.find("Sam");
17    if(it1 != s.end()) cout<<"Found "<<*it1<<endl;
18    else cout<<"Not Found Sam"<<endl;
19
20
21    it1=s.find("Rose");
22    if(it1 != s.end()) cout<<"Found "<<*it1<<endl;
23    else cout<<"Not Found Rose"<<endl;
24
25
26    /// Print set elements (use iterator)
27
28    for(it2=s.begin() ; it2 != s.end() ; it2++){
29        cout<<(*it2)<<endl;
30    }
31
32 }
```

```
main.cpp X

1 #include<bits/stdc++.h>
2 using namespace std;
3 map<char,int> mp;
4 map<char,int> :: iterator it1,it2;
5
6 int main()
7 {
8     mp['A']=1;
9     mp['B']=2;
10    mp['C']=3;
11    mp['D']=4;
12
13    it1=mp.begin();
14    cout<<(*it1).first<<endl; // key
15    cout<<(*it1).second<<endl; // value
16
17    it1=mp.find('B');
18    if(it1 != mp.end()) cout<<"Found "<<(*it1).first<<" "<<(*it1).second<<endl;
19    else cout<<"Not Found B"<<endl;
20
21
22    it1=mp.find('E');
23    if(it1 != mp.end()) cout<<"Found "<<(*it1).first<<" "<<(*it1).second<<endl;
24    else cout<<"Not Found E"<<endl;
25
26
27
28     /// Print map elements (use iterator)
29
30     for(it2=mp.begin() ; it2 != mp.end() ; it2++){
31         cout<<(*it2).first<<" "<<(*it2).second<<endl;
32     }
33 }
34
```

# **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 X
1 #include<bits/stdc++.h>
2 using namespace std;
3 set<int>s;
4 vector<int>v;
5 map<int, int>mp;
6
7 int main()
8 {
9     string a="ABCDEFG";
10    for(auto p:a)
11        cout<<p;
12
13    cout<<endl<<endl;
14    v.push_back(1);
15    v.push_back(1);
16    v.push_back(2);
17    v.push_back(2);
18    for(auto x:v)
19        cout<<x<<" ";
20
21    cout<<endl<<endl;
22    s.insert(3);
23    s.insert(3);
24    s.insert(2);
25    for(auto x:s)
26        cout<<x<<" ";
27
28    cout<<endl<<endl;
29    mp[1]=10;
30    mp[2]=20;
31    for(auto p:mp)
32        cout<<p.first<<" "<<p.second<<endl;
33
34 }
```

End