C++

- Bjarne Stroustrup (1979 - 1983, C with Classes)
- C++ literally means “increased C”

Structure

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

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

Output

```cpp
int age=27;
cout<<"Output line 1"<<endl;
cout<<120<<" "<<3.1416<<" Age = "<<age<<endl;
```

Input

```cpp
int num;
double pointnum;
char ch;
cin>>num>>pointnum>>ch;
cout<<num<<" "<<pointnum<<" "<<ch<<endl;
```

File I/O

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

int main()
{
    ///writing to a file
    ofstream myfile;
    myfile.open("testfile.txt", ios::out | ios::app);

    if(myfile.is_open()){
        myfile<<"First line"<<endl;
        myfile<<"Second line"<<endl;
        myfile.close();
    }
    else{
        cout<<"Unable to open file"<<endl;
    }

    ///reading from a file
    ifstream myfile1;
    myfile1.open("testfile.txt", ios::in);

    string line;
    if(myfile1.is_open()){
        while(!myfile1.eof()){
            getline(myfile1, line);
            cout<<line<<endl;
        }
        myfile1.close();
    }
    else{
        cout<<"Unable to open file"<<endl;
    }
    return 0;
}
```

C++ : STL – Standard Template Library

The Standard Template Library (STL) is **a set of C++ template classes** to provide common programming data structures and functions such as dynamic arrays (vector), queues (queue), stacks (stack), associative arrays (map), etc. It is a library of container classes, algorithms, and iterators.

STL Containers – A container is a holder of object that stores a collection of other objects. It manages the storage space for its elements and provides member functions to access them, either directly or through iterators.

Some of the major containers:

- Vector
- Stack
- Queue
- Map

1. **Vector –** It represents arrays that can change in size. It uses contiguous storage locations for their elements. Internally, vectors use a dynamically allocated array to store their elements. This array may need to be reallocated in order to grow in size when new elements are inserted, which implies allocating a new array and moving all elements to it.

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    ///----------------------------------------------------------------------

    int arr[]={1,2,3,4};
    int arrlen = sizeof(arr)/sizeof(int);

    ///Constructing an empty vector
    vector<int> v1;
    ///Constructing a vector from an array
    vector<int> v2 (arr, arr+arrlen);
    ///Constructing a vector from other vector
    vector<int> v3 (v1);

    ///----------------------------------------------------------------------

    ///Accessing vector elements using iterator (points to the vector elements)
    cout<<"Showing vector 2 - using Iterator"<<endl;
    for(vector<int>::iterator it=v2.begin();it!=v2.end();it++){
        cout<<*it<<endl;
    }
    ///or,
    cout<<"Showing vector 2 - using Index"<<endl;
    for(int ind=0;ind<v2.size();ind++){
        cout<<v2[ind]<<endl;
    }

    ///To check whether the vector is empty or not
    cout<<"Vector 1 empty check: "<< v1.empty() <<endl;
    ///---------------------------------------------------------------------

    ///Insertion - add single new element at the end of the vector, after its current last
element
    v1.push_back(100);
    ///Insertion - insert new elements (1 or more) before the element at the specified
position
    vector<int>::iterator it=v1.begin();
    v1.insert(it,v2.begin(),v2.begin()+3);

    cout<<"Showing vector 1 - after insertion"<<endl;
    for(int ind=0;ind<v1.size();ind++){
        cout<<v1[ind]<<endl;
    }

    ///---------------------------------------------------------------------

    ///Deletion - remove the last element from the vector
    v2.pop_back();

    cout<<"Showing vector 2 - after pop_back()"<<endl;
    for(int ind=0;ind<v2.size();ind++){
        cout<<v2[ind]<<endl;
    }

    ///Deletion - remove a single or range of elements from the vector
    v1.erase(v1.begin()+2); ///remove only the 3rd elements

    cout<<"Showing vector 1 - after erase(v1.begin()+2)"<<endl;
    for(int ind=0;ind<v1.size();ind++){
        cout<<v1[ind]<<endl;
    }


    v2.erase(v2.begin(),v2.begin()+2); ///remove the first 2 elements

    cout<<"Showing vector 2 - after erase(v2.begin(), v2.begin()+2)"<<endl;
    for(int ind=0;ind<v2.size();ind++){
        cout<<v2[ind]<<endl;
    }

    ///---------------------------------------------------------------------

    return 0;
}
```

2. **Stack –** A type of container that operates in a LIFO (Last-in First-out) context, where elements are inserted and extracted only from one end of the container.

```
#include <iostream>
#include <stack>

using namespace std;

int main()
{
    ///-------------------------------------------------------
    ///constructing an empty stack
    stack<int> stk;

    ///--------------------------------------------------------

    ///To check the current size of the stack
    cout<<"Stack size: "<< stk.size() <<endl;
    ///To check whether the stack is empty or not
    cout<<"Stack empty check: "<< stk.empty() <<endl;

    ///--------------------------------------------------------

    ///Insertion - to push an item in the stack
    cout<<"Inserting 100 and then 200"<<endl;
    stk.push(100);
    stk.push(200);

    ///--------------------------------------------------------

    ///Access - to access the topmost element of the stack
    cout<<"Top element: "<< stk.top() <<endl;

    ///--------------------------------------------------------

    ///Deletion - to pop an item from the stack top
    stk.pop();
    cout<<"After pop: "<< stk.top() <<endl;

    ///--------------------------------------------------------


    return 0;
}
```

3. **Queue –** A type of container that operates in FIFO (first-in first-out) context, where elements are inserted into one end of the container and extracted from the other.

```
#include <iostream>
#include <queue>

using namespace std;

int main()
{
    ///------------------------------------------------------------------

    ///constructing an empty queue
    queue<int> q;

    ///------------------------------------------------------------------

    ///To check the current size of the queue
    cout<<"Size of the queue: "<<q.size()<<endl;
    ///To check an empty queue
    cout<<"Queue empty check: "<<q.empty()<<endl;

    ///------------------------------------------------------------------

    ///Insertion - push an element at the end of the queue
    cout<<"Inserting 100 and then 200"<<endl;
    q.push(100);
    q.push(200);

    ///-----------------------------------------------------------------

    ///Accessing the front (oldest) element of the queue
    cout<<"Oldest element: "<<q.front()<<endl;

    ///-----------------------------------------------------------------

    ///Deletion - pop the oldest element from the queue
    q.pop();
    cout<<"After pop: "<<q.front()<<endl;

    ///-----------------------------------------------------------------

    return 0;
}
```

4. **Map –** It is an associative container that stores 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.

```
#include <iostream>
#include <map>

using namespace std;

int main()
{
    ///--------------------------------------------------------------

    ///constructing an empty map
    map<char,int> first;
    ///constructing a new map from other map
    map<char,int> second (first.begin(),first.end());
    map<char,int> third (second);

    ///--------------------------------------------------------------

    ///Insertion - if key exists then replaces the existing value otherwise insert a new
element with that key
    cout<<"Inserting a=>100 and b=>20"<<endl;
    first['a']=10;
    first['b']=20;
    first['a']=100; ///replace

    ///Accessing any map element
    cout<<"Accessing key a: "<< first['a']<<endl;

    ///--------------------------------------------------------------

    ///To check the size of the map
    cout<<"Map size: "<<first.size()<<endl;
    ///To check whether the map is empty
    cout<<"Map empty check: "<<first.empty()<<endl;

    ///--------------------------------------------------------------
    ///Deletion - to delete a specific map element
    first.erase('b');
    cout<<"After deletion key b, map size: "<<first.size()<<endl;

    ///---------------------------------------------------------------

    ///Search - to search for a key presence, if not found then returns map.end()
    map<char,int>::iterator it=first.find('a');
    if(it!=first.end()) cout<<"key a exists"<<endl;
    else cout<<"key a doesn't exist"<<endl;

    ///---------------------------------------------------------------

    ///Accessing all the elements of the map
    map<char,int> newmap;
    newmap['a']=100;
    newmap['b']=200;
    newmap['c']=300;

    for(map<char,int>::iterator it=newmap.begin();it!=newmap.end();it++){
        cout<<"Key: "<<it->first<< " => "<<it->second<<endl;
    }

    ///---------------------------------------------------------------

    return 0;
}
```

#include <string> – strings are objects that represent sequence of characters.

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    ///--------------------------------------------------------

    ///constructing strings
    string s0;
    string s1 ("Initial value");

    ///--------------------------------------------------------

    ///new string assignment
    s0 = "new string";

    ///string concatenation
    string s2=s0+s1;

    ///to append new string at the end
    s2+=" appended portion.";

    ///equality/greater than/less than checking
    string s3 = "abc";
    string s4 = "def";
    cout<<"abc == def "<< (s3==s4) <<endl;
    cout<<"abc != def "<< (s3!=s4) <<endl;
    cout<<"abc > def "<< (s3>s4) <<endl;
    cout<<"abc >= def "<< (s3>=s4) <<endl;
    cout<<"abc < def "<< (s3<s4) <<endl;
    cout<<"abc <= def "<< (s3<=s4) <<endl;

    ///--------------------------------------------------------

    ///To check the size of the string
    cout<<"String length: "<<s2.size()<<endl;
    ///To check whether the string is empty or not
    cout<<"String empty check: "<<s2.empty()<<endl;

    ///--------------------------------------------------------

    ///taking input from command prompt
    string input1, input2;
    getline(cin, input2); ///full line input
    cin>>input1; ///word input

    ///outputting a string
    cout<<input1<<endl;
    cout<<input2<<endl;

    ///--------------------------------------------------------
    string s5="abcdefghijghij";

    ///accessing specific characters
    cout<<s5[2]<<endl;

    ///extracting substrings
    cout<<s5.substr(3)<<endl; ///starting position
    cout<<s5.substr(3,5)<<endl; ///starting position, length

    ///finding substring, returns -1 if not found
    int find_index = s5.find("mno");
    if(find_index==-1) cout<<"Substring not found"<<endl;
    else cout<<"Substring found"<<endl;

    ///similarly we can use replace(), insert() and count()

    ///--------------------------------------------------------

    return 0;
}
```

#include <algorithm> – It defines a collection of functions especially designed to be used on ranges of elements (STL containers, array etc.)

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

///function for descending order sort
bool sortfn(int val1, int val2){
    if(val1>val2) return true;
    else return false;
}
```

```
int main()
{
    ///-----------------------------------------------------

    vector<int> v;
    v.push_back(100);
    v.push_back(40);
    v.push_back(150);
    v.push_back(128);

    vector<int> v1(v);

    ///to find out the minimum element
    vector<int>::iterator minit=min_element(v.begin(), v.end());
    cout<<*minit<<endl;

    ///to find out the maximum element
    vector<int>::iterator maxit=max_element(v.begin(), v.end());
    cout<<*maxit<<endl;

    ///To sort a vector
    sort(v.begin(),v.end()); ///default: ascending order sort
    cout<<"Output"<<endl;
    for(int ind=0;ind<v.size();ind++){
        cout<<v[ind]<<" ";
    }
    cout<<endl;

    ///To sort based on predefined rule
    sort(v.begin(),v.end(),sortfn);
    cout<<"Output"<<endl;
    for(int ind=0;ind<v.size();ind++){
        cout<<v[ind]<<" ";
    }
    cout<<endl;

    ///To reverse a vector
    reverse(v1.begin(),v1.end());
    cout<<"Output"<<endl;
    for(int ind=0;ind<v1.size();ind++){
        cout<<v1[ind]<<" ";
    }
    cout<<endl;

    ///Searching for an element
    vector<int>::iterator find_ind=find(v1.begin(),v1.end(),180);
    if(find_ind!=v1.end()) cout<<"Found"<<endl;
    else cout<<"Not Found"<<endl;

    ///To count for number of occurrences
    v1.push_back(99);
    v1.push_back(99);
    cout<<count(v1.begin(),v1.end(),99)<<endl;

    return 0;
}
```

Reference Link: https://www.cplusplus.com/reference/