STL In C++

- The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators

- **STL has three components**

- Containers
  - Containers are used to manage collections of objects of a certain kind. There are several different types of containers like deque, list, vector, map etc.
- Algorithms
  - Algorithms act on containers. They provide the means by which you will perform initialization, sorting, searching, and transforming of the contents of containers.

- Iterators
  - Iterators are used to step through the elements of collections of objects. These collections may be containers or subsets of containers.

STL Container

- Containers or container classes store objects and data. There are in total seven standard “first-class” container classes and three container adaptor classes and only seven header files that provide access to these containers or container adaptors. Sequence Containers: implement data structures which can be accessed in a sequential manner.
  - vector
  - list
  - deque
  - arrays
  - forward_list(Introduced in C++11)
- Container Adaptors : provide a different interface for sequential containers.
  - queue
  - priority_queue
  - stack

- Associative Containers : implement sorted data structures that can be quickly searched (O(log n) complexity).
  - set
  - multiset
  - map
  - multimap
- Unordered Associative Containers : implement unordered data structures that can be quickly searched
  - unordered_set (Introduced in C++11)
  - unordered_multiset (Introduced in C++11)
  - unordered_map (Introduced in C++11)
  - unordered_multimap

Vector Container

- A vector is a sequence container class that implements dynamic array, means size automatically changes when appending elements. A vector stores the elements in contiguous memory locations and allocates the memory as needed at run time.

- Difference between vector and array

- An array follows static approach, means its size cannot be changed during run time while vector implements dynamic array means it automatically resizes itself when appending elements.

- Syntax
- Consider a vector 'v1'.
- Syntax would be:
  - vector<object_type> v1;
- Example:
  - vector<int> iv; // create zero-length int vector
  - vector<char> cv(5); // create 5-element char vector
  - vector<char> cv(5, 'x'); // initialize a 5-element char vector
- vector<int> iv2(iv); // create int vector from an int vector

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

int main() {
   // create a vector to store int
   vector<int> vec;
   int i;
   // display the original size of vec
   cout << "vector size = " << vec.size() << endl;

   // push 5 values into the vector
   for(i = 0; i < 5; i++) {
      vec.push_back(i);
   }
   // display extended size of vec
   cout << "extended vector size = " << vec.size() << endl;
   // access 5 values from the vector
   for(i = 0; i < 5; i++) {
      cout << "value of vec [" << i << "] = " << vec[i] << endl;
   }
   // use iterator to access the values
   vector<int>::iterator v = vec.begin();
   while( v != vec.end()) {
      cout << "value of v = " << *v << endl;
    v++;
   }

   return 0;
}
```

```
vector size = 0
extended vector size = 5
value of vec [0] = 0
value of vec [1] = 1
value of vec [2] = 2
value of vec [3] = 3
value of vec [4] = 4
value of v = 0
value of v = 1
value of v = 2
value of v = 3
value of v = 4
```

- The push_back( ) member function inserts value at the end of the vector, expanding its size as needed.
- The size( ) function displays the size of the vector.
- The function begin( ) returns an iterator to the start of the vector.
- The function end( ) returns an iterator to the end of the vector.

C++ Vector Functions

- at() It provides a reference to an element.
- back() It gives a reference to the last element.
- front() It gives a reference to the first element.
- swap() It exchanges the elements between two vectors.
- push_back() It adds a new element at the end.
- pop_back() It removes a last element from the vector.
- empty() It determines whether the vector is empty or not.
- insert() It inserts new element at the specified position.

C++ Vector Functions

- erase() It deletes the specified element.
- resize() It modifies the size of the vector.
- clear() It removes all the elements from the vector.
- size() It determines a number of elements in the vector.
- capacity() It determines the current capacity of the vector.
- assign() It assigns new values to the vector.

- operator=() It assigns new values to the vector container
- .
- operator[]() It access a specified element.

- end() It refers to the past-lats-element in the vector.

C++ Vector Functions

- emplace() It inserts a new element just before the position pos.
- emplace_back() It inserts a new element at the end.
- rend() It points the element preceding the first element of the vector.
- rbegin() It points the last element of the vector.
- begin() It points the first element of the vector.
- max_size() It determines the maximum size that vector can hold.

C++ Vector Functions

- cend() It refers to the past-last-element in the vector.
- cbegin() It refers to the first element of the vector.
- crbegin() It refers to the last character of the vector.
- crend() It refers to the element preceding the first element of the vector.
- data() It writes the data of the vector into an array.
- shrink_to_fit() It reduces the capacity and makes it equal to the size of the vector.

```cpp
#include<iostream>
#include<vector>
using namespace std;
int main()
{
vector<int> v1{1,2,3,4};
for(int i=0;i<v1.size();i++)
cout<<v1.at(i);
return 0;
}
```

O/P

1234

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main()
{   vector<int> g1;

    for (int i = 1; i <= 5; i++)
        g1.push_back(i);

    cout << "Output of begin and end: ";
    for (auto i = g1.begin(); i != g1.end(); ++i)
        cout << *i << " ";

    cout << "\nOutput of cbegin and cend: ";
    for (auto i = g1.cbegin(); i != g1.cend(); ++i)
        cout << *i << " ";

    cout << "\nOutput of rbegin and rend: ";
    for (auto ir = g1.rbegin(); ir != g1.rend(); ++ir)
        cout << *ir << " ";

    cout << "\nOutput of crbegin and crend : ";
    for (auto ir = g1.crbegin(); ir != g1.crend(); ++ir)
        cout << *ir << " ";

    return 0;
}
```

Output:
```
Output of begin and end: 1 2 3 4 5
Output of cbegin and cend: 1 2 3 4 5
Output of rbegin and rend: 5 4 3 2 1
Output of crbegin and crend : 5 4 3 2 1
```

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> g1;
    for (int i = 1; i <= 5; i++)
        g1.push_back(i);
    cout << "Size : " << g1.size();
    cout << "\nCapacity : " << g1.capacity();
    cout << "\nMax_Size : " << g1.max_size();
    // resizes the vector size to 4
    g1.resize(4);
    // prints the vector size after resize()
    cout << "\nSize : " << g1.size();
    // checks if the vector is empty or not
    if (g1.empty() == false)
        cout << "\nVector is not empty";
    else
        cout << "\nVector is empty";
    // Shrinks the vector
    g1.shrink_to_fit();
    cout << "\nVector elements are: ";
    for (auto it = g1.begin(); it != g1.end(); it++)
        cout << *it << " ";
    return 0;
}
```

```
Size : 5
Capacity : 8
Max_Size : 2305843009213693951
Size : 4
Vector is not empty
Vector elements are: 1 2 3 4
```

MAP in C++ STL

- The map class supports an associative container in which unique keys are mapped with values.
- In essence, a key is simply a name that you give to a value. Once a value has been stored, you can retrieve it by using its key.
- Thus, in its most general sense, a map is a list of key/value pairs.
- The power of a map is that you can look up a value given its key.

- For example, you could define a map that uses a person's name as its key and stores that person's telephone number as its value. Associative containers are becoming more popular in programming.
- Map can hold only unique keys. Duplicate keys are not allowed.

Consider the example:

Keys(Roll NO)	Names
16030141001	Akash
16030141002	Yuvraj
16030141003	Ashish
16030141004	Arun

The above example shows a key and value pair. The roll number is the key and each student has a different roll number, hence unique key representing them.

- **Syntax:**

- map<key_type , value_type> map_name;

- This will create a map with key of type **Key_type** and value of type **value_type**.
- One thing which is to remembered is that key of a map and corresponding values are always inserted as a pair, you cannot insert only key or just a value in a map.

Some basic functions associated with Map:

- **begin():** Returns an iterator to the first element in the map.
- **size():** Returns the number of elements in the map.
- **empty():** Returns a boolean value indicating whether the map is empty.
- **insert(pair(key, value)):** Adds a new key-value pair to the map.
- **find(val):** Gives the iterator to the element *val*, if it is found otherwise it returns m.end()
- **erase(iterator position):** Removes the element at the position pointed by the iterator.
- **erase(const g):** Removes the key value *g* from the map.
- **clear():** Removes all the elements from the map.

Program

```
// A simple map demonstration.
#include <iostream>
#include <map>
using namespace std;
int main()
{
map<char, int> m;
int i;
// put pairs into map
for(i=0; i<26; i++) {
m.insert(pair<char, int>('A'+i, 65+i));
}
char ch;
cout << "Enter key: ";
cin >> ch;
map<char, int>::iterator p;
// find value given key
p = m.find(ch);
if(p != m.end())
cout << "Its ASCII value is " << p->second;
else
cout << "Key not in map.\n";
return 0;
}
```

```
Enter key: A
Its ASCII value is 65
```

Program

```cpp
#include <iostream>
#include <iterator>
#include <map>
using namespace std;
int main()
{
    map<int, int> marks;
    marks.insert(pair<int, int>(160, 42));
    marks.insert(pair<int, int>(161, 30));
    marks.insert(pair<int, int>(162, 40));
    marks.insert(pair<int, int>(163, 50));
    marks.insert(pair<int, int>(164, 31));
    marks.insert(pair<int, int>(165, 12));
    marks.insert(pair<int, int>(166, 34));

    map<int, int>::iterator itr;
    cout << "The map marks is : \n";
    cout << "ROLL NO"<<"\t  \t"<<"Marks"<<endl;
    for (itr =  marks.begin(); itr !=  marks.end(); ++itr) {
       cout  << itr->first
            << "\t   \t" << itr->second << "\n";
    }
    cout << endl;
    return 0;
}
```

```
The map marks is :
ROLL NO          Marks
160              42
161              30
162              40
163              50
164              31
165              12
166              34
```

References

- C++: The Complete Reference, 4th Edition Herbert Schildt
- https://www.javatpoint.com/cpp-vector
- https://www.geeksforgeeks.org/vector-in-cpp-stl/
- https://www.tutorialspoint.com/cplusplus/cpp_standard_library.htm