Introduction to the STL

STL

- 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.

Container

- A container is a way that stored data is organized in memory e.g. Stacks and linked lists. Another container, the array, is so common that it’s built into C++ (and most other computer languages). However, there are many other kinds of containers, and the STL includes the most useful. The STL containers are implemented by template classes, so they can be easily customized to hold different kinds of data.

Algorithms

- Algorithms in the STL are procedures that are applied to containers to process their data in various ways. For example, there are algorithms to sort, copy, search, and merge data. Algorithms are represented by template functions. These functions are not member functions of the container classes.

Iterators

- Iterators are a generalization of the concept of pointers: they point to elements in a container. You can increment an iterator, as you can a pointer, so it points in turn to each element in a container. Iterators are a key part of the STL because they connect algorithms with containers.

Algorithms

- Sorting
- Searching
- Important STL Algorithms
- Useful Array algorithms
- Partition Operations

Sorting

- Sorting is one of the most basic functions applied to data. It means arranging the data in a particular fashion, which can be increasing or decreasing. There is a builtin function in C++ STL by the name of sort().
- In more details it is implemented using hybrid of QuickSort, HeapSort and InsertionSort.By default, it uses QuickSort but if QuickSort is doing unfair partitioning and taking more than N*logN time, it switches to HeapSort and when the array size becomes really small, it switches to InsertionSort.

Sorting

```
#include <algorithm>
#include <iostream>
using namespace std;
void show(int a[], int array_size)
{
		for (int i = 0; i < array_size; ++i)
				cout << a[i] << " ";
}
int main()
{
		int a[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
		int asize = sizeof(a) / sizeof(a[0]);
		cout << "The array before sorting is : \n";
		show(a, asize);
		sort(a, a + asize);
		cout << "\n\nThe array after sorting is :\n";
		// print the array after sorting
		show(a, asize);
		return 0;
}
```

Searching- Binary Search

```
#include <algorithm>
#include <iostream>
using namespace std;
void show(int a[], int arraysize)
{
   for (int i = 0; i < arraysize; ++i)
      cout << a[i] << ",";
}
int main()
{   int a[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
   int asize = sizeof(a) / sizeof(a[0]);
   cout << "\nThe array is : \n";
   show(a, asize);
   sort(a, a + asize);
   if (binary_search(a, a + asize, 2))
      cout << "\nElement found in the array";
   else
      cout << "\nElement not found in the array";
   cout << "\n\nNow, say we want to search for 10";
   if (binary_search(a, a + 10, 10))
      cout << "\nElement found in the array";
   else
      cout << "\nElement not found in the array";
}
```

Array algorithms in C++ STL (all_of, any_of, none_of, copy_n and iota)

```
#include<algorithm> // for all_of()
using namespace std;
int main()
{
        // Initializing array
        int ar[6] = {1, 2, 3, 4, 5, -6};
        // Checking if all elements are positive
        all_of(ar, ar+6, [](int x) { return x>0; })?
                cout << "All are positive elements" :
                cout << "All are not positive elements";

        return 0;

}
```

```
// C++ code to demonstrate working of any_of()
#include<iostream>
#include<algorithm> // for any_of()
using namespace std;
int main()
{
        // Initializing array
        int ar[6] = {1, 2, 3, 4, 5, -6};

        // Checking if any element is negative
        any_of(ar, ar+6, [](int x){ return x<0; })?
                cout << "There exists a negative element" :
                cout << "All are positive elements";

        return 0;

}
```

Containers

- vector
- list
- deque
- arrays
- forward_list(Introduced in C++11)
- queue
- priority_queue
- Stack
- set
- multiset
- map
- Multimap
- unordered_set (Introduced in C++11)
- unordered_multiset (Introduced in C++11)
- unordered_map (Introduced in C++11)
- unordered_multimap (Introduced in C++11)

Vectors

Vectors are the same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators.

1. begin() – Returns an iterator pointing to the first element in the vector
2. end() – Returns an iterator pointing to the theoretical element that follows the last element in the vector
3. rbegin() – Returns a reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
4. rend() – Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)
5. cbegin() – Returns a constant iterator pointing to the first element in the vector.
6. cend() – Returns a constant iterator pointing to the theoretical element that follows the last element in the vector.
7. crbegin() – Returns a constant reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
8. crend() – Returns a constant reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end

https://www.geeksforgeeks.org/vector-in-cpp-stl/

Vectors

```
#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 << " ";
}
```

```
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
```

https://www.geeksforgeeks.org/vector-in-cpp-stl/

Vectors

```
#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();
        g1.resize(4);
        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";
        g1.shrink_to_fit();
        cout << "\nVector elements are: ";
        for (auto it = g1.begin(); it != g1.end(); it++)
                cout << *it << " ";
}
```

https://www.geeksforgeeks.org/vector-in-cpp-stl/