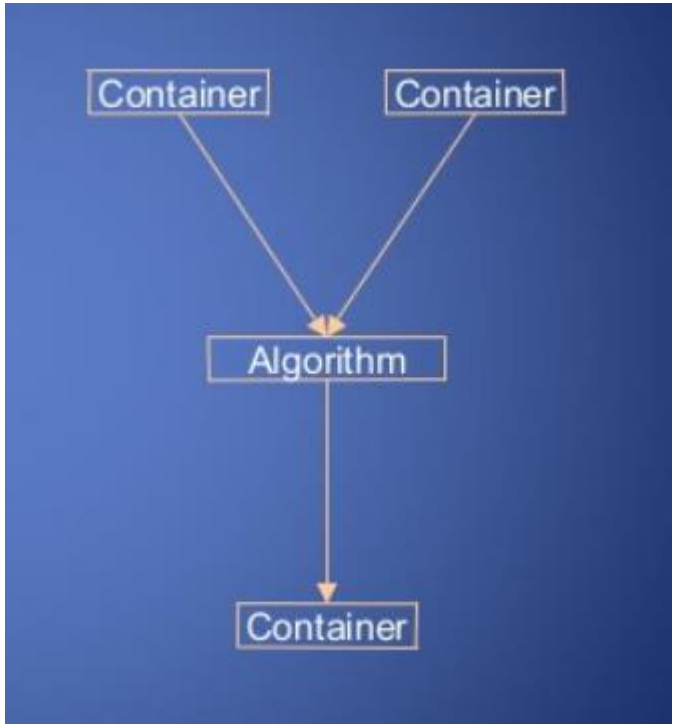
STL

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.

STL has four components

Algorithms
Containers
Functions
Iterators



Sort in C++ Standard Template Library (STL)

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

using namespace std;

void show(int a[])
{
    for(int i = 0; i < 10; ++i)
        cout << a[i] << " ";
}

int main()
{
    int a[10]= {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
    cout << "\n The array before sorting is : ";
    show(a);

    sort(a, a+10);

    cout << "\n\n The array after sorting is : ";
    show(a);

    return 0;

}
```

Binary Search in C++ Standard Template Library (STL)

```
#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 << "\n The array is : ";
    show(a, asize);

    cout << "\n\nLet's say we want to search for 2 in the array";
    cout << "\n So, we first sort the array";
    sort(a, a + asize);
    cout << "\n\n The array after sorting is : ";
    show(a, asize);
    cout << "\n\nNow, we do the binary search";
    if (binary_search(a, a + 10, 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";

    return 0;
}
```

Array algorithms in C++ STL

```
#include<iostream>
#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;

}
```

Array algorithms in C++ STL

```
#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;

}
```

Array algorithms in C++ STL

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

    // Checking if no element is negative
    none_of(ar, ar+6, [](int x){ return x<0; })?
         cout << "No negative elements" :
         cout << "There are negative elements";

    return 0;
}
```

Functors in C++

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

int increment(int x) {  return (x+1); }

int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Apply increment to all elements of
    // arr[] and store the modified elements
    // back in arr[]
    transform(arr, arr+n, arr, increment);

    for (int i=0; i<n; i++)
        cout << arr[i] << S" ";

    return 0;
}
```

Functors are objects that can be treated as though they are a function or function pointer. Functors are most commonly used along with STLs in a scenario like following

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.

```
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
int main()
{
    vector<int> ar = { 1, 2, 3, 4, 5 };

    // Declaring iterator to a vector
    vector<int>::iterator ptr;

    // Displaying vector elements using begin() and end()
    cout << "The vector elements are : ";
    for (ptr = ar.begin(); ptr < ar.end(); ptr++)
        cout << *ptr << " ";

    return 0;
}
```

Containers

- Three types of containers
  - Sequence containers:
    - linear data structures such as vectors and linked lists
  - Associative containers:
    - non-linear containers such as hash tables
  - Container adapters:
    - constrained sequence containers such as stacks and queues

Vector in C++ STL

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

Set in C++ Standard Template Library (STL)

- begin() – Returns an iterator to the first element in the set.
- end() – Returns an iterator to the theoretical element that follows last element in the set.
- size() – Returns the number of elements in the set.
- max_size() – Returns the maximum number of elements that the set can hold.
- empty() – Returns whether the set is empty.

Write a program to implement array with STL

```
int main()
{
array<int, 5> arr;
array<int, 5>::iterator it;
int choice, item;
arr.fill(0);
int count = 0;
while (1)
{
cout<<"\n----------------------"<<endl;
cout<<"Array Implementation in Stl"<<endl;
cout<<"\n----------------------"<<endl;
cout<<"1.Insert Element into the Array"<<endl;
cout<<"2.Size of the array"<<endl;
cout<<"3.Front Element of Array"<<endl;
cout<<"4.Back Element of Array"<<endl;
cout<<"5.Display elements of the Array"<<endl;
cout<<"6.Exit"<<endl;
cout<<"Enter your Choice: ";
cin>>choice;
switch(choice)
{
case 1:
cout<<"Enter value to be inserted: ";
cin>>item;
arr.at(count) = item;
count++;
break;
case 2:
cout<<"Size of the Array: ";
cout<<arr.size()<<endl;
break;
case 3:
cout<<"Front Element of the Array: ";
cout<<arr.front()<<endl;
break;
case 4:
cout<<"Back Element of the Stack: ";
cout<<arr.back()<<endl;
break;
case 5:
for (it = arr.begin(); it != arr.end(); ++it )
cout <<" "<< *it;
cout<<endl;
break;
case 6:
exit(1);
break;
default:
cout<<"Wrong Choice"<<endl;
}
}
return 0;
}
```