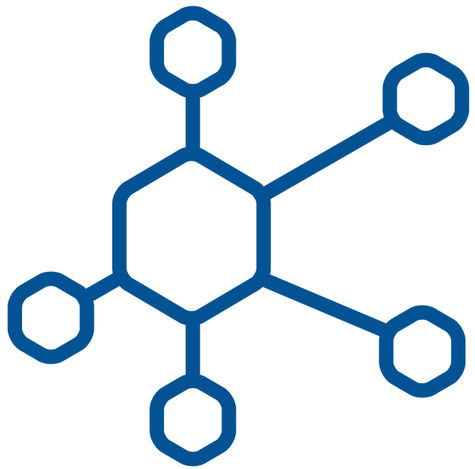
Association for
Computing Machinery
INSAT Student Chapter



Data Structures & STL

What is STL?

The **STL** is a collection of algorithms, data structures, and other components that can be used to simplify the development of C++ programs.

What is STL?

It provides a range of containers, such as vectors, lists, and maps, as well as algorithms for searching, sorting, and manipulating data.

Let’s discover some Data Structures

Dynamic Arrays

Dynamic arrays' sizes can be changed dynamically during execution.

C++'s dynamic arrays are called vectors.

We specify the type of elements that the dynamic array (vector) will hold during the declaration of the vector

```
vector<int> v1(10); // dynamic array of 10 ints (10 is the initial size)
vector<double> v2(5, 0.42); // vector of doubles of size 5 [0.42, 0.42, 0.42, 0.42, 0.42]
vector<long long> v3; // vector of long long elements that is initially empty
struct UserDefinedType {
  // ...
};
vector<UserDefinedType> v3;

```

Dynamic Arrays

The push_back method adds an element to the end of the array, potentially growing its size when it’s needed.

We access vector like we access ordinary arrays.

Insertion's Time complexity: **O(1)**

```
1 vector<int> v;
2 v.push_back(4); // [4]
3 v.push_back(2); // [4, 2]
4 v.push_back(0); // [4, 2, 0]
5
6 cout << v[0] << v[1] << v[2] << endl; // 420
```

Dynamic Arrays

The back method returns the last element of the vector

The pop_back method removes the last element of the vector

The size method returns the size of the vector

Deletion's Time complexity: O(1)

```
1 vector<int> v = {1, 2, 3};
2 cout << v.back() << endl; // 3
3 v.pop_back();
4 cout << v.back() << endl; // 2
```

Sets

Sets are a type of associative container in which each element has to be unique.

The value of the element cannot be modified once it is added to the set, though it is possible to remove and add the modified value of that element.

insert/erase: **log(n)**

Sets

```
set<int> s;
s.insert(2);
s.insert(1);
s.insert(3);
cout<<s.count(2)<<endl;// 1
cout<<s.count(4)<<endl;// 0
s.erase(2);
cout<<s.count(2)<<endl;// 0
s.insert(4);
cout<<s.count(4)<<endl;// 1
```

```
set<int> s={5,7,2,7,5,1,9};

cout<<s.size()<<endl; //5

for(auto i:s){
    cout<<i<<" "; //1 2 5 7 9
}
```

Pairs

Pair is a container defined in <utility> header used to combine together two values that may be different in type.

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

```
vector <pair<int, string>> v;
v.push_back({1, "one"});
v.push_back({4, "four"});

for (auto x : v) {
    cout << x.first << " " << x.second << endl;
} // 1 one
  // 4 four
```

Maps

Maps are associative containers that store elements in a mapped fashion. Each element has a key value and a mapped value. No two mapped values can have the same key values.

Key => Value

```
map<string, int> m;
m["a"] = 1;
m["b"] = 2;
m["c"] = 3;
m["d"] = 0;
cout << m["a"] << endl; // 1
cout << m["d"] << endl; // 0
```

Maps

Insertion : O(log n) if it is an ordered map & O(1) if it is an unordered map

Deletion : O(log n) if it is an ordered map & O(1) if it is an unordered map

Stacks

Stacks are a type of container adaptor with LIFO(Last In First Out) type of work, where a new element is added at one end (top) and an element is removed from that end only.

```
stack<int> s;
s.push(10);
s.push(20);
s.push(30);

cout<<s.size()<<endl; // 3
cout<<s.top()<<endl; // 30
s.pop();
cout<<s.top()<<endl; // 20
```

Stack Operations

faceprep

Queues

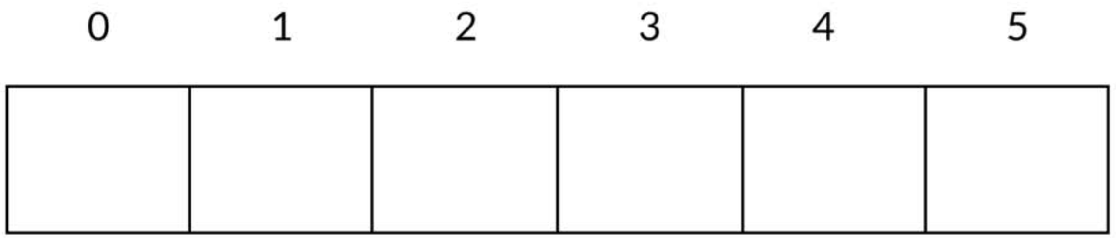
The queue is a type of container which operates in a First In First Out (FIFO) type of arrangement. Elements are inserted at the back(end) and are deleted from the front of the queue.

```
queue<int> q;
q.push(1);
q.push(25);
q.push(31);

cout<<q.size()<<endl; // 3
cout<<q.front()<<endl; // 1
cout<<q.back()<<endl; // 31
q.pop();
cout<<q.front()<<endl; // 25
```

Queue Operations

Front = Rear = -1



Empty Queue

faceprep

Numeric Librabry

This library consists of basic mathematical functions and types, as well as optimized numeric arrays and support for random number generation.

Numeric Librabry

Accumulate

Returns the sum of all the values lying in a range between [first, last) with the variable sum. (O(N))

```
vector<int> v = {1,2,3,4};
int sum = accumulate(v.begin(),v.end(),0);
cout<<sum<<endl; // 10
```

Numeric Librabry

gcd

calculates the greatest common divisor of two integers. (O(log2n))

```
int a=20,b=30;
cout<<__gcd(a,b)<<endl; // 10
```

Numeric Librabry

iota

assigns a value to the elements in the range [first, last) of the array which is incremented at each step by val++. (O(n))

```
int arr[5] = {0};
iota(arr, arr+5, 20);
for(int i=0; i<5; i++){
    cout << arr[i] << " "; // 20 21 22 23 24
}
```

Algorithms Librabry

Reverse

- The reverse method reverses the elements. (O(n))

```
vector <int> v={5,2,7,1,9,3,4};

reverse(v.begin(),v.end());

for(auto i:v){
    cout<<i<<" "; // 4 3 9 1 7 2 5
}
```

Algorithms Librabry

Sort/is_sorted

Sort method takes a range of elements and sorts it. (O(nlogn))
Is_sorted checks if the elements in range [first, last] are sorted in ascending order.

```
string s="bcdagfe";
sort(s.begin(),s.end());
cout<<s; // bcdfgv
```

```
vector <int> v={5,2,7,1,9,3,4};

sort(v.begin(),v.end());
for (int i=0;i<v.size();i++){
    cout<<v[i]<<" "; // 1 2 3 4 5 7 9
}

sort(v.begin(),v.end(),greater<int>());
for (int i=0;i<v.size();i++){
    cout<<v[i]<<" "; // 9 7 5 4 3 2 1
}
```

Algorithms Librabry

Find

search for a specific value within a range (O(n))

```
string s1="Association of computing machinery INSAT Student chapter";
string s2="INSAT";

auto it2=s1.find(s2);

if (it2≠string::npos)
    cout << it2 << endl; // 35
```

```
vector <int> v={5,2,7,1,9,3,4};

auto it=find(v.begin(),v.end(),3);

if (it≠v.end())
    cout << it-v.begin() << endl; // 5
```

Algorithms Librabry

for_each

It’s a loop that accepts a function which executes over each of the container elements.

```
void print(int x){
    cout<<x+1<<" ";
}

int main(){

vector<int> vect = {1,2,3,4,5};

for_each(vect.begin(), vect.end(), print);

}
```

Algorithms Librabry

all_of / any_of / none_of

It checks for a given property on every element and returns true when each/none/any element in range satisfies specified property, else returns false. (O(n))

```
bool negative(int x){
    return x<0;
}

int main(){
vector<int> vect = {1,2,3,4,5};

all_of(vect.begin(),vect.end(),negative) ?
    cout<<"All are positive" :
    cout<<"All are not positive";
}
```

Algorithms Librabry

Swap

Swaps the value of two elements

```
int a=5,b=6;
swap(a,b);

cout<<a<<" "<<b<<endl; //6 5
```

Thank you for your attention!