

## 1] What is STL in C++? Write about containers in STL ?

Ans=

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. It is a generalized library and so, its components are parameterized.

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.

1. Sequence Containers: implement data structures that can be accessed in a sequential manner.

- vector
- list
- deque
- arrays
- forward\_list

2. Container Adaptors: provide a different interface for sequential containers.

- queue
- priority\_queue
- stack

3. Associative Containers: implement sorted data structures that can be quickly searched ( $O(\log n)$  complexity).

- set
- multiset

- map
- multimap

4. Unordered Associative Containers: implement unordered data structures that can be quickly searched

- unordered\_set
- unordered\_multiset
- unordered\_map
- unordered\_multimap

## 2] Explain about sequence, associative, adapter, and unordered in STL

### Sequence containers

Ans=

#### 1. Sequential Containers

In C++, sequential containers allow us to store elements that can be accessed in sequential order. Internally, sequential containers are implemented as arrays or linked lists data structures.

#### Types of Sequential Containers

1. vector:
  - Resizable array.
  - Fast random access to elements.
  - Amortized constant-time insertion and deletion at the end.
  - Linear time insertion/deletion elsewhere due to shifting.
2. deque:
  - Double-ended queue.
  - Fast insertion/deletion at both ends.

- Slightly slower access than vectors.
  - Internally uses a sequence of fixed-size arrays.
3. `list`:
    - Doubly-linked list.
    - Constant-time insertion/deletion anywhere.
    - Slow access by index.
    - Efficient for frequent insertion/deletion in the middle.
  4. `forward_list`:
    - Singly-linked list.
    - Similar to `std::list` but with less memory overhead.
    - Only supports forward iteration.
  5. `array`:
    - Fixed-size array.
    - Size determined at compile-time.
    - Fast access to elements.
    - Cannot be resized after declaration.

## 2.Container Adapter

Container adapters provide a different interface for sequential containers.

1. `stack`: Adapts a container to provide stack (LIFO data structure) (class template).
2. `queue`: Adapts a container to provide queue (FIFO data structure) (class template).
3. `priority_queue`: Adapts a container to provide priority queue (class template).

## 3. Associative Containers

In C++, associative containers allow us to store elements in sorted order.

The order doesn't depend upon when the element is inserted.

Internally, they are implemented as binary tree data structures.

## Types of Associative Containers

### 1. set:

- Container of unique, sorted elements.
- Implemented as a balanced binary search tree (usually red-black tree).
- Insertion, deletion, and search operations have a time complexity of  $O(\log n)$ .


### 2. map:

- Collection of key-value pairs where keys are unique and sorted.
- Operations like insertion, deletion, and search have  $O(\log n)$  time complexity.

### 3. multiset:

- Similar to `std::set` but allows duplicate elements.
- Insertion, deletion, and search have a time complexity of  $O(\log n)$ .

### 4. multimap:

- 
- Allows multiple elements with the same key.
  - Operations like insertion, deletion, and search have  $O(\log n)$  time complexity.

## 4. Unordered Associative Containers

Unordered associative containers implement unsorted (hashed) data structures that can be quickly searched ( $O(1)$  amortized,  $O(n)$  worst-case complexity).

## Types of Unordered Associative Containers

### 1. `unordered_set`:

- A collection of unique elements with no specific order.
- Implemented using a hash table.
- Insertion, deletion, and search operations have an average time complexity of  $O(1)$ , though it can degrade to  $O(n)$  in worst-case scenarios.

### 2. `unordered_map`:

- A collection of key-value pairs where keys are unique and there is no specific order.
- Implemented using a hash table.
- Average time complexity for insertion, deletion, and search operations is  $O(1)$ , with potential degradation to  $O(n)$  in worst-case scenarios.

### 3. `unordered_multiset`:

- Similar to `std::unordered_set` but allows duplicate elements.
- Implemented using a hash table.
- Average time complexity for insertion, deletion, and search is  $O(1)$ , with potential degradation to  $O(n)$  in worst-case scenarios.

### 4. `unordered_multimap`:

- Allows multiple elements with the same key and no specific order.
- Implemented using a hash table.
- Average time complexity for insertion, deletion, and search operations is  $O(1)$ , with potential degradation to  $O(n)$  in worst-case scenarios.

### 3] Implement Priority Queue in C++

A C++ priority queue is a type of [container adapter](#), specifically designed such that the first element of the queue is either the greatest or the smallest of all elements in the queue, and elements are in non-increasing or non-decreasing order (hence we can see that each element of the queue has a priority {fixed order}).

Syntax:

```
std::priority_queue<int> pq;
```

```
#include<iostream>

#include<queue>

using namespace std;

void showpq(

    priority_queue<int, vector<int>, greater<int> > g)
{

    while (!g.empty()) {

        cout << ' ' << g.top();

        g.pop();

    }

    cout << '\n';

}
```

```
void showArray(int* arr, int n)
{
    for (int i = 0; i < n; i++) {
        cout << arr[i] << ' ';
    }

    cout << endl;
}

int main()
{
    int arr[6] = { 11, 0, 7, 5, 7, 3};

    priority_queue<int, vector<int>, greater<int> > gquiz(
        arr, arr + 6);

    cout << "Array: ";

    showArray(arr, 6);

    cout << "Priority Queue : ";

    showpq(gquiz);

    return 0;
}
```

```
C:\Users\User\Desktop\ADA\priority_queue.exe
Array: 11 0 7 5 7 3
Priority Queue : 0 3 5 7 7 11

Process returned 0 (0x0)   execution time : 0.100 s
Press any key to continue.
```

## 4] Implement Stack using array and linked list in C++

### Stack using array:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    struct Node *next;
};
struct Node* top = NULL;
void push(int val) {
    struct Node* newnode = (struct Node*) malloc(sizeof(struct Node));
    newnode->data = val;
    newnode->next = top;
    top = newnode;
}
void pop() {
    if(top==NULL)
        cout<<"Stack Underflow"<<endl;
    else {
        cout<<"The popped element is "<< top->data <<endl;
        top = top->next;
    }
}
```



```

}

void display() {
    struct Node* ptr;
    if(top==NULL)
        cout<<"stack is empty";
    else {
        ptr = top;
        cout<<"Stack elements are: ";
        while (ptr != NULL) {
            cout<< ptr->data <<" ";
            ptr = ptr->next;
        }
    }
    cout<<endl;
}

int main() {
    int ch, val;
    cout<<"1) Push in stack"<<endl;
    cout<<"2) Pop from stack"<<endl;
    cout<<"3) Display stack"<<endl;
    cout<<"4) Exit"<<endl;
    do {
        cout<<"Enter choice: "<<endl;
        cin>>ch;
        switch(ch) {
            case 1: {
                cout<<"Enter value to be pushed:"<<endl;
                cin>>val;
                push(val);
                break;
            }
            case 2: {
                pop();
                break;
            }
            case 3: {
                display();
                break;
            }
            case 4: {

```

```

        cout<<"Exit"<<endl;
        break;
    }
    default: {
        cout<<"Invalid Choice"<<endl;
    }
}
}while(ch!=4);
return 0;
}

```

Output:

```

Select C:\Users\User\Desktop\ADA\stack_11.exe
1) Push in stack
2) Pop from stack
3) Display stack
4) Exit
Enter choice:
2
Stack Underflow
Enter choice:
1
Enter value to be pushed:
3
Enter choice:
1
Enter value to be pushed:
3
Enter choice:
1
Enter value to be pushed:
4
Enter choice:
1
Enter value to be pushed:
6
Enter choice:
1
Enter value to be pushed:
7
Enter choice:
6
Invalid Choice
Enter choice:

```

Stack using linked list:

```

#include <iostream>
using namespace std;
struct Node {
    int data;
    struct Node *next;
};

```

```

struct Node* top = NULL;
void push(int val) {
    struct Node* newnode = (struct Node*) malloc(sizeof(struct Node));
    newnode->data = val;
    newnode->next = top;
    top = newnode;
}
void pop() {
    if(top==NULL)
        cout<<"Stack Underflow"<<endl;
    else {
        cout<<"The popped element is "<< top->data <<endl;
        top = top->next;
    }
}
void display() {
    struct Node* ptr;
    if(top==NULL)
        cout<<"stack is empty";
    else {
        ptr = top;
        cout<<"Stack elements are: ";
        while (ptr != NULL) {
            cout<< ptr->data <<" ";
            ptr = ptr->next;
        }
    }
    cout<<endl;
}
int main() {
    int ch, val;
    cout<<"1) Push in stack"<<endl;
    cout<<"2) Pop from stack"<<endl;
    cout<<"3) Display stack"<<endl;
    cout<<"4) Exit"<<endl;
    do {
        cout<<"Enter choice: "<<endl;
        cin>>ch;
        switch(ch) {
            case 1: {

```

```
        cout<<"Enter value to be pushed:"<<endl;
        cin>>val;
        push(val);
        break;
    }
    case 2: {
        pop();
        break;
    }
    case 3: {
        display();
        break;
    }
    case 4: {
        cout<<"Exit"<<endl;
        break;
    }
    default: {
        cout<<"Invalid Choice"<<endl;
    }
}
}while(ch!=4);
return 0;
}
```

**Output:**

```
C:\Users\User\Desktop\ADA\stack_ll.exe
1) Push in stack
2) Pop from stack
3) Display stack
4) Exit
Enter choice:
1
Enter value to be pushed:
3
Enter choice:
1
Enter value to be pushed:
5
Enter choice:
1
Enter value to be pushed:
5
Enter choice:
8
Invalid Choice
Enter choice:
```

## 5]Implement Binary Search trees in C++.

```
#include <iostream>

using namespace std;

struct node {
    int key;
    struct node *left, *right;
};

// A utility function to create a new BST node
struct node* newNode(int item)
{
    struct node* temp
        = new struct node;
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to insert
// a new node with given key in BST
struct node* insert(struct node* node, int key)
```

```

{
    // If the tree is empty, return a new node
    if (node == NULL)
        return newNode(key);

    // Otherwise, recur down the tree
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    // Return the (unchanged) node pointer
    return node;
}

// Utility function to search a key in a BST
struct node* search(struct node* root, int key)
{
    // Base Cases: root is null or key is present at root
    if (root == NULL || root->key == key)
        return root;

    // Key is greater than root's key
    if (root->key < key)
        return search(root->right, key);

    // Key is smaller than root's key
    return search(root->left, key);
}

// Driver Code
int main()
{
    struct node* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
}

```

```
insert(root, 80);


// Key to be found
int key = 30;

// Searching in a BST
if (search(root, key) == NULL)
    cout << key << " not found" << endl;
else
    cout << key << " found" << endl;

key = 60;

// Searching in a BST
if (search(root, key) == NULL)
    cout << key << " not found" << endl;
else
    cout << key << " found" << endl;
return 0;
}
```

## Output:

 C:\Users\User\Desktop\ADA\BinaryST.exe

```
30 found
60 found
```

```
Process returned 0 (0x0)   execution time : 0.119 s
Press any key to continue.
```

## Conclusion

Hence, from this lab we learnt about Standard Template Library along with its containers. Here, we also implemented stack using array and linked list, Binary Search tree and priority queue.

Astha Thapa