**Name: M.Faizan**

**Roll no: L1S23BSCS0191**

**Data Structure and Alogorithm**

**Lab 3:**

**Task 1:**

## **Task 1**

Create a C++ generic abstract class named **List**, with the following:

**Attributes:**

Type \*arr;

int maxSize;

int currentSize;

**Functions:**

* 1. virtual void addElementAtFirstIndex(Type) = 0;

// Should add the element at the first position of the **List**

* 1. virtual void addElementAtLastIndex(Type) = 0;

// Should add the element at the last position of the **List**

* 1. virtual Type removeElementFromEnd() = 0;

// Should remove the element from the last position of the **List**

* 1. virtual void removeElementFromStart() = 0;

//Should remove the element from the first position of the **List**

* Write a **parameterized constructor with default arguments** for the above class.
* Write a **copy constructor** for the above class.
* Write **destructor** for the above class.

**Header file:**

template <typename type>

class list {

protected:

type\* arr;

int maxSize;

int currentSize;

public:

list(int size = 10);

list(const list& other);

virtual ~List();

virtual void addElementAtFirstIndex(type element) = 0;

virtual void addElementAtLastIndex(type element) = 0;

virtual type removeElementFromEnd() = 0;

virtual void removeElementFromStart() = 0;

};

**Source File:**

#include <iostream>

#include "list.h"

template <typename type>

list<type>::list(int size) : maxSize(size), currentSize(0) {

arr = new type[maxSize];

}

template <typename type>

list<type>::list(const list& other) : maxSize(other.maxSize), currentSize(other.currentSize) {

arr = new type[maxSize];

for (int i = 0; i < currentSize; ++i) {

arr[i] = other.arr[i];

}

}

template <typename type>

list<type>::~list() {

delete[] arr;

}

**Task 2:**

Create a menu-based program for the following functions, using the class made in task 1; make a class named as **MyList**, having the following additional functionalities:

* **bool empty()**: Returns whether the MyList is empty or not
* **bool full():** Returns whether the MyList is full or not  
  **int size():** Returns the current size of the MyList
* **bool insertAt(int index, T value):** Adds a value at the index passed to the function, returns true if the index is present and value is added else returns false.   
  **Type last():** Returns the last element of the MyList
* **bool search(Type):** Returns true if the searched value is present in the list else returns false
* Write a **parameterized constructor with default arguments** for the above class.
* Write a **copy constructor** for the above class.
* Write **destructor** for the above class.

**Header File:**

template <typename type>

class mylist : public List<type> {

public:

mylist(int size = 10);

mylist(const mylist& other);

~mylist();

bool empty();

bool full();

int size();

bool insertAt(int index, type value);

type last();

bool search(type value);

};

**Source File:**

#include <iostream>

#include "List.h"

template <typename type>

MyList<type>::MyList(int size) : List<type>(size) {}

template <typename type>

MyList<type>::MyList(const MyList& other) : List<type>(other) {}

template <typename type>

MyList<type>::~MyList() {}

template <typename type>

bool MyList<type>::empty() {

return currentSize == 0;

}

template <typename type>

bool MyList<type>::full() {

return currentSize >= maxSize;

}

template <typename type>

int MyList<type>::size() {

return currentSize;

}

template <typename type>

bool MyList<type>::insertAt(int index, type value) {

if (index < 0 || index > currentSize || full()) {

return false;

}

for (int i = currentSize; i > index; --i) {

arr[i] = arr[i - 1];

}

arr[index] = value;

currentSize++;

return true;

}

template <typename type>

type MyList<type>::last() {

if (currentSize == 0) {

throw std::out\_of\_range("List is empty.");

}

return arr[currentSize - 1];

}

template <typename type>

bool MyList<type>::search(type value) {

for (int i = 0; i < currentSize; ++i) {

if (arr[i] == value) {

return true;

}

}

return false;

}

**Task 3:**

**Header File:**