National University of Computer and Emerging Sciences, Lahore Campus

	Course:	Data Structures	Course Code:	CS218
	Program:	BS(Computer Science)	Semester:	Spring 2021
	Due Date:	29-April-2021	Total Marks:	10
	Section:	4G and 4H	Weight (Tentative)	3%
	Evaluation:	Assignment-03	Page(s):	4

Submission Path: Google classroom

Part A: Implementing a Generic Stack [3 marks]

In computer science, a **stack** is an abstract data type that serves as a collection of elements, with two main operations:

- **Push**, which adds an element to the collection, and
- **Pop**, which removes an element from the collection

The order in which elements come off a stack is **LIFO** (**last in, first out**).

In this part of the assignment you are required to implement a stack using a singly linked list.

List class is provided; you are required to use this class and do not make any changes except in destructor.

```
template <class T>
class Node
{
        public:
        T data;
        Node<T>* next;
};

template <class T>
class List
{
        Node<T>* head;
        Node<T>* tail;
        public:

        List()
        {
                head = nullptr;
                tail = nullptr;
        }

void insertAtEnd(T value)
{
        Node<T>* newNode = new Node<T>;
        newNode->data = value;
        newNode->next = nullptr;

        if (head == nullptr)
```

```
	{head = tail = newNode;}
	else
	{tail->next = newNode;
	tail = tail->next;}
}

void RemoveAtEnd()
{
	Node<T> * current = head;

	// Do nothing if list is already Empty
	if (current != nullptr)
	{
		//One element
		if (head == tail)
		{
		head = tail = nullptr;
		delete current;
		}
		else
		{
		//Find second last element
			while (current->next != tail)
				{
					current = current->next;
				}
			tail = current;
			current = current->next;
			tail->next = nullptr;
			delete current;
		}
	}
}
bool isEmpty()
{
	return head == nullptr;
}

~List()
{
	// Write: Your Code here [1 mark]
}
};
```

Requirements for Stack Class: **[2 marks]**

```
template <class T>
class Stack
{
//Data Member:
	List<T>collection
//Required Functions:
	void Push (T val){}
	T Pop (){}
```

```
    bool isEmpty(){}
    bool isFull(){}      //Think about this
    Stack() {}    //default constructor: This will be empty
    }
```

- Keep function signatures exactly same in your code
- You can't add/remove any function or data member in this class.
- You will **NOT** need a destructor here.

Note: Test your code until here thoroughly before moving to next part.

Part B: Application of Stack: Web Browser [7 marks]

One of the most common applications of stack is in web browsers. To implement the undo / redo functionality a stack data structure is used. In this section you will create a web browser with that functionality.

You are free to design the **web browser class** as you wish (give it a unique name). However, follow the guidelines:

Restrictions: [1 mark]

- You can only use objects of your stack class as data structure (No Arrays, List, Queues, and Trees etc).
- You may create variables if you wish.
- For filing you can use string from std.

Requirements:

1. Prompt the user to:
    - **give a URL:** (e.g www.google.com for simplicity you can use "google" "facebook"): **[2 marks]**
        - Open a file named "google.txt", "facebook.txt" and display the content of file on screen
        - if file doesn't exist, error: "Invalid URL"
    - **Undo**: if a URL was opened after another URL the previous goes into undo **[1 mark]**
        - Open and display content of previous URL, do nothing if no previous URL exists.
    - **Redo**: if an undo was called from a URL, then that URL goes into redo **[1 mark]**
        - if a redo URL exist, open and display content on screen, else do nothing
    - **History**: **[2 marks]**
        - Search for the URL in undo stack. print "*You have visited/not visited this website*"

- You can only use stack object and its operations to search.
- Make sure the undo stack remains **intact**
- (Optional) you can search in redo stack as well. **[bonus 1 mark]**
- (Optional) you can permanently store all history in a file (Data persistence) so when the program launch you can search in undo, redo stacks as well as past history if you like. **[bonus 1 mark]**

Try to make re-usable methods. For example if you create a method: **display(filename){}** in your class, You can use this in giving a URL, Undo and Redo methods.

Happy Programming with Data Structures 😊