**DSA (Data Structures & Algorithms)**

**MID-TERM EXAM**

**X1**

**Instructions:**

* Indent your code.
* Comment your code.
* Use meaningful variable names.
* Plan your code carefully on a piece of paper before you implement it.
* Name of the program should be same as the task name. i.e. the first program should be Task\_1.cpp

# void main() is not allowed. Use int main()

* **You have to work in multiple files. i.e separate .h and .cpp files**
* **You are not allowed to use any built-in functions**

# You are required to follow the naming conventions as follow:

* + **Variables:** firstName; (no underscores allowed)
  + **Function:** getName(); (no underscores allowed)
  + **ClassName:** BankAccount (no underscores allowed)

**Task 1:**

**Check Matching Parenthesis:**

**Description:**

Develop a program to check if a given expression has properly matched parentheses. The program should analyze the expression using a stack data structure to determine if the parentheses are balanced.

**Requirements:**

* Implement a stack data structure to assist in the analysis process.
* Define functions to check if parentheses in the expression are properly matched.
* Display appropriate messages indicating whether the parentheses are balanced or not.
* Handle scenarios involving nested parentheses.

**Example:**

**Input: exp = “**[()]{}{[()()]()}**”**

**Output: Balanced**

**Input: exp = “[(])”**

**Output: Not Balanced**

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

**Attributes:**

1. Type \* stackArray;
2. int maxSize;
3. int stackTop;

**Functions:**

virtual void Push(Type) = 0;

virtual Type Pop() = 0;

* Using the base class (Stack), make another derived (MyStack). Implement both pure virtual functions Push () and pop() declared in base class(Stack), into derived class (MyStack).
* Implement the ‘**CheckParentheses’** function which is given an expression, (string exp) to examine whether the pairs and the orders of “{“, “}”, “(“, “)”, “[“, “]” are correct in the given expression.

**Type CheckParentheses(string &exp)**

**Task 2:**

**Description:**

You are required to write a function ‘moveNthToHead’ that takes as a parameter a positive integer, n. The function should move the nth element of a LinkedList to the Head, while maintaining the order of the remaining elements in the list unchanged.

The function should modify the given LinkedList in place.

**Example:**

**Input:**

Suppose we have a LinkedList having following data = {5, 11, 34, 67, 43, 55} and n = 3.

**Output:**

LinkedList should be modified as follows:

LinkedList = {34, 5, 11, 67, 43, 55}

**Requirements:**

* **Function Implementation:** Write the definition of the function ‘moveNthToHead’ that modifies the given LinkedList according to the specified requirements.
* **LinkedList Operations:** You may use any appropriate data structure or implementation to represent the LinkedList. Ensure that the LinkedList operations (insert, delete) are properly implemented.
* **Error Handling:** You need to handle cases where ‘n’ is greater than the size of LinkedList.

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

**Attributes:**

1. Node \*head
2. Node \*tail

**Functions:**

virtual void insertAtTail(int) = 0;

virtual void deleteAtTail() = 0;

* Using the base class (LinkedList), make another derived (MyLinkedList). Implement both pure virtual functions **insertAtTail**(int) and **deleteAtTail**() declared in base class(LinkedList), into derived class (MyLinkedList).
* Implement the ‘**moveNthToHead’**, to modify linkedlist according to the requirement. (you can use object of linkedlist as parameter or without parameter it’s your own choice)

**Type moveNthToHead (Type &inputList, int n)**

**or**

**void moveNthToHead (int n)**