University of Central Punjab

Faculty of Information Technology

Data Structures and Algorithms
Spring 2021

Instructions:

- Indent your code.
- Comment your code.
- Use meaningful variable names.
- Plan your code carefully on a piece of paper before you implement it.
- Submit only .CPP and .H files in a single ZIP folder.
- The name of each CPP file should be the question number. E.g., Q1.cpp
- The name of the ZIP folder should be your complete registration number. E.g., L1S21BSCS0000
- **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 system("pause")**
- **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)

Students are required to complete the following tasks in lab timings.

Task

Linked List:

Linked List is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a Linked List are linked using pointers.

Create a class named LinkedList with the following attribute and functions. You must implement the LinkedList using head pointer only.

Attribute:

• Node* head; //Node is a structure comprising of a variable (datatype is your discretion) to store the value and a pointer of Node type to store the address of the next node.

Functions:

• int getHead(); //Returns the value of the first node in the linked list.

• int getTail(); //Returns the value of the last node in the linked list.

• void insertAtFront(int); //Adds a new node with the given value at the beginning of the linked list.

• void insertAtEnd(int); //Adds a new node with the given value at the end of the linked list.

• void insertAtPosition(int); //Adds a new node with the given value at the given position in the linked list.

• void insertSorted(int); //Adds elements to the linked list in ascending order.

• void removeFromFront(); //Removes the first node of the linked list, and reduces the size of the linked list by 1.

• void removeFromEnd(); //Removes the last node of the linked list, and reduces the size of the linked list by 1.

• void removeFromPosition(int); //Removes the node from the given position in the linked list and reduces the size of the linked list by 1.

• void remove(int, int); //Removes all nodes from the given starting position to the ending position in the linked list, and reduces the size of the linked accordingly.

• bool IsEmpty(); //Returns whether the list is empty or not.

• int size(); //Returns the number of nodes in the list.

• void printList(); //Prints all the values stored in the linked list.

• Constructor

• Destructor

Task

Implement doubly circular Linked list CLASS consisting of all basic utilities and follwing functions too.

1 — Count()
Write a Count() function that counts the number of times a given int occurs in a list. The code for this has the classic list traversal structure as demonstrated in Length().

2 — GetNth()
Write a GetNth() function that takes a linked list and an integer index and returns the data value stored in the node at that index position.

3 — DeleteList()
Write a function DeleteList() that takes a list, deallocates all of its memory and sets its head pointer to NULL (the empty list).

4 — Pop()
Write a Pop() function that is the inverse of Push(). Pop() takes a non-empty list, deletes the head node, and returns the head node's data.

5 — InsertNth()
write a function InsertNth() which can insert a new node at any given index within a list. if 'i' is the index and 'l' is the List's total length then {i<=l}

6 — SortedInsert()
Write a SortedInsert() function which given a list that is sorted in increasing order, and a single node, inserts the node into the correct sorted position in the list.

7 — InsertSort()
Write an InsertSort() function which given a list, rearranges its nodes so they are sorted in increasing order. It should use SortedInsert().

8 — Append()
Write an Append() function that takes two lists, 'a' and 'b', appends 'b' onto the end of 'a', and then sets 'b' to NULL

9 — RemoveDuplicates()
Write a RemoveDuplicates() function which takes a list sorted in increasing order and deletes any duplicate nodes from the list. Ideally, the list should only be traversed once.

10 — ShuffleMerge()
Given two lists, merge their nodes together to make one list, taking nodes alternately between the two lists. So ShuffleMerge() with {1, 2, 3} and {7, 13, 1} should yield {1, 7, 2, 13, 3, 1}. If either list runs out, all the nodes should be taken from the other list.

11 — Reverse()
Write an iterative Reverse() function that reverses a list by rearranging all the .next pointers and the head pointer.