SGSITS
Dept. of Computer Engineering
CO24007: Data Structure
Lab Assignment 2

Friday 14th July, 2017

1. **Traverse a linked list**
   Write a function **void printLinkedList(Node * head)** that prints elements of the linked list from first to last on separate lines.

2. Given a linked list, write a function **Node * NthNodeFromLast(Node * head, int n)**that returns n^{th} node from the end if it exists, $NULL$ otherwise

3. **Revering a linked list**

   (a) Given a linked list, write a function **Node* reverseLinkedList(Node * head)** that takes head of the linked list and reverses it. e.g. if linked list is $1- > 2- > 3- > 4- > 6- > 7- > NULL$, then after reversing the linked list should be $7- > 6- > 5- > 4- > 3- > 2- > 1- > NULL$ and the head should point to 7. The function returns *head* pointer to the new head of the linked list.

   (b) Given a linked list, write a function **Node* reverseLinkedList(Node * head, int n)** that takes head of the linked list and reverses reverses every n nodes. e.g. if linked list is $1- > 2- > 3- > 4- > 6- > 7- > NULL$ and $n = 3$, then after reversing the linked list should be $3- > 2- > 1- > 6- > 5- > 4- > 7- > NULL$ and the head should point to 7. The function returns *head* pointer to the new head of the linked list.

4. **Palindrome in a linked list**
   Given a singly linked list of characters, write a function **int isLinkedListPalindrome(Node * head)** that returns ***1*** if the given list is palindrome, else ***0***.

5. **Rotate a Linked List**
   Given a singly linked list, write a function **Node * rotateLinkedList(Node *head,**

int k) to rotate the linked list counter-clockwise by k nodes and return the new head pointer. k is a given positive integer. For example, if the given linked list is $10- > 20- > 30- > 40- > 50- > 60- > NULL$ and k is 2, the list should be modified to $30- > 40- > 50- > 60- > 10- > 20- > NULL$.

6. **Deleting from linked list**
   Write a function **int deleteFromLinkedList(Node * head, int data)** that removes every occurance of **data** from the linked list pointed by *head*.

7. **Sorting**
   Write a function **Node* sortLinkedList(Node * head)** that sorts the elements of the linked list in increasing order. Assume that the elements are comparable (e.g. *int*, $float$ etc.).

8. **Merging two linked lists**
   Write a function **Node * mergeLinkesLists(Node * head1, Node * head2)** that takes two sorted linked lists and merges them into one linked list, then returns the head of new linked list.

9. **Union and Intersection**

   (a) Write a function **Node * findUnionOfLinkesLists(Node * head1, Node * head2)** that returns a new linked list with union of elements of linked lists pointed by *head*1 and *head*2.

   (b) Write a function **Node * findIntersectionOfLinkesLists(Node * head1, Node * head2)** that returns a new linked list with intersection of elements of linked lists pointed by *head*1 and *head*2. In case there are no common elements, return an empty linked list.

   NOTE: the linked list pointed by *head*1 and *head*2 may not be sorted.

10. **Deleting a linked list: memory leaking**
    Write a function **void deleteLinkedList(Node * head)** that deletes the linked list. Note: Deleting the linked list involves deallocate memory consumed by each and every node of the linked list. Just by pointing *head* to $NULL$ DOES NOT free up the space, but the nodes keep occupying the space. Pointing the *head* to $NULL$ is incorrect way of deleting linked list, this way system's available memory decreases, this phenomenon is also called *memory leaking*.