Assignment No:06

Problem Statement:

Linked list operations: Create a linked list of names and birthdays of students. Write a menu driven C++ program to perform following operations
1. Insert name and birthday of new student
2. Delete a student entry
3. Display a happy birthday message for whom today (based on system date) is birthday
4. Display list of students with their birthdays

Objectives

- To understand use of link list as data structure
- To know the operations on linked lists
- To know the variations of linked lists

Theory :

Linked List:

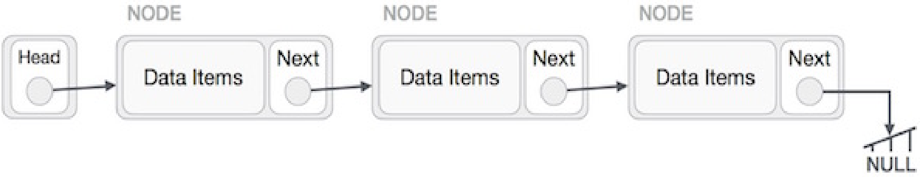
A linked list is a **linear** data structure where each element is a **separate** object. Each element (we will call it a **node**) of a list is comprising of two items - the data and a reference to the next **node**. The last **node** has a reference to null. The entry point into a linked list is called the head of the list.

Linked List Representation:

Linked list is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

- **Link** − Each link of a linked list can store a data called an element.
- **Next** − Each link of a linked list contains a link to the next link called Next.
- **LinkedList** − A Linked List contains the connection link to the first link called First.

Linked list can be visualized as a chain of nodes, where every node points to the next node.



Types of Linked List

Following are the various types of linked list.

- **Simple Linked List** − Item navigation is forward only.
- **Doubly Linked List** − Items can be navigated forward and backward.
- **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.

Basic Operations

Following are the basic operations supported by a list.

- **Insertion** − Adds an element at the beginning of the list.
- **Deletion** − Deletes an element at the beginning of the list.
- **Display** − Displays the complete list.
- **Search** − Searches an element using the given key.

- **Delete** − Deletes an element using the given key.

Advantages

- Efficient insertion and removal of elements at any index
- Efficient memory allocation (No need to resize/reallocate memory)
- As size of linked list can increase or decrease at run time so there is no memory wastage

Disadvantages

- More memory is required to store elements in linked list as compared to array
- Elements or nodes traversal is difficult in linked list.
- In linked list reverse traversing is really difficult.

Algorithm:

1.Add node in the start of the link list.

Input: Starting node of the link list i.e. start, Create temporary node in the list i.e. temp, accept data to add in the list from user.

O**utput**: node temp gets added to the start of the list.

```
Step1 : Start

Step2 : Accept input. Suppose start is the starting node of link list.

Step3 : Create new temporary node. Assign data field to that node.
i.e. struct node *temp;
temp=(struct node*)malloc(sizeof(struct node));

Step4 : add new node to the begining of the list perform following steps.
set temp->next = start;
set start=temp

Step5 : display message node added successfully.

Step6 : Stop.
```

2.Add node in between the link list.

Input : starting node of the link list i.e. start , position of the node in the link list, create temporary node , accept data to field of temporary node from user.

```
Step 1 : Start

Step 2 : Accept input from user. create temporary node to add in the link list i.e.
                temp=(struct node *)malloc(sizeof(struct node));
          accept data field of link list from user i.e.
                scanf("%d",&temp->data);
          accept location of node after which want to add node in the list.
                   i.e. scanf("%d",&num);
Step 3 : Check whether list is empty. then, display message list is empty otherwise,
                   set count = 1
                   set q = start
                   a) iterate or traverse each node  untill last node encounters  i.e.
                     while(q!=NULL)
                     {
                      check count==location number
                     i.e.
                         if(count == num)
                         {
                         set temp->next = q->next
                         set q->next = temp
                         }

                     }
```

Time complexity:
Discuss the time and space complexity of all the functions you have implemented

Conclusion: Thus we have successfully implement the singly link list

Practice Problem:
Write C++ code for implementing Circular link list.