# DATA STRUCTURES LAB

## LAB RECORD

***Submitted by***

## Palash Mishra – 201B172

## Submitted to:  Dr.KUNJ BIHARI MEENA

2021-2022

## Department of Computer Science & Engineering

# LAB Exercise 8: Linked List

**1. WAP to insert at the end of the linked list.**

**https://www.hackerrank.com/challenges/insert-a-node-at-the-tail-of-a-linked-list/problem**

**Solution:**

```
SinglyLinkedListNode* insertNodeAtTail(SinglyLinkedListNode* head, int data) {
        SinglyLinkedListNode *temp = head;
        SinglyLinkedListNode *insertNode = new SinglyLinkedListNode(data);

        if(head == NULL) {
            return insertNode;
        }

        while(temp->next != NULL) {
            temp = temp->next;
        }
        temp->next  = insertNode;
        insertNode->next = NULL;

        return head;
}
```

**2. WAP to print the elements of a linked list.**

**https://www.hackerrank.com/challenges/print-the-elements-of-a-linked-list/problem**

**Solution:**

```
void printLinkedList(SinglyLinkedListNode* head) {

 SinglyLinkedListNode *temp = head;

 while(temp != NULL){

    cout<<temp->data<<endl;

       temp = temp->next;

    }



}
```

**3. WAP to insert at the beginning of the linked list.**

**https://www.hackerrank.com/challenges/insert-a-node-at-the-head-of-a-linked-list/problem**

**Solution:**

```
SinglyLinkedListNode* insertNodeAtHead(SinglyLinkedListNode* list, int data) {
SinglyLinkedListNode* temp
=(SinglyLinkedListNode*)(malloc(sizeof(SinglyLinkedListNode))) ;
temp->data =data;
temp->next= NULL;
if(list==NULL)
{
   list=temp;
}
else{
temp->next = list;
list = temp;
}

return list;
```

**4. WAP to insert a node at specify position in a linked list.**

**https://www.hackerrank.com/challenges/insert-a-node-at-a-specific-position-in-a-linked-list/problem**

**Solution:**

```
SinglyLinkedListNode* insertNodeAtPosition(SinglyLinkedListNode* head, int data,
int position) {
      SinglyLinkedListNode *node = new SinglyLinkedListNode(data);
         SinglyLinkedListNode *temp = head;

         if(head == NULL){
             return node;
         }
        int i=0;
        while(i < position-1) {
            temp = temp->next;
            i++;
        }
        node->next = temp->next;
        temp->next = node;
        return head;
}
```

**5. WAP to delete a node from given position in a linked list.**

**https://www.hackerrank.com/challenges/delete-a-node-from-a-linked-list/problem**

**<u>Solution:</u>**

```
SinglyLinkedListNode* deleteNode(SinglyLinkedListNode* head, int position) {


SinglyLinkedListNode *p = head;
SinglyLinkedListNode *q;
int l=0;
if(position==0)
{
p=head->next;
head = p;
}
else{
while(p!=NULL && l<position)
{
   q=p;
   l++;
   p=p->next;
}
if(p==NULL)
{
   return head;
}
else
{
q->next=p->next;
}

}
return head;
}
```

**6. WAP to print the elements in reverse order in a linked list.**

**https://www.hackerrank.com/challenges/print-the-elements-of-a-linked-list-in-reverse/problem**

**Solution:**

```
void reversePrint(SinglyLinkedListNode* head) {

if(head==NULL)
{return;
}
else
{
   reversePrint(head->next);
   cout<<head->data<<endl;
}
}
```

**7. WAP to insert a node into a sorted doubly linked list.**

**https://www.hackerrank.com/challenges/insert-a-node-into-a-sorted-doubly-linked-list/problem**

**Solution:**

```
DoublyLinkedListNode* sortedInsert(DoublyLinkedListNode* head, int data) {
DoublyLinkedListNode *p = (DoublyLinkedListNode*)malloc(sizeof(DoublyLinkedListNode));
DoublyLinkedListNode *q = head;
p->data= data;
if(q->data>data)
{
   q->prev = p;
   p->next = q;
   p->prev = NULL;
   head = p;
   return head;
}
while(q!=NULL)
{
   if (q->data >= data)
   {
      p->next = q;
      p->prev = q->prev;
      q->prev->next = p;
      return head;
   }
   else if (q->next==NULL)
   {
      q->next = p;
      p->prev = q;
      p->next = NULL;
      return head;
   }
```

```
    q = q->next;
}
return head;
}
```

**8. WAP to detect loop or cycle in a linked list.**

**https://www.hackerrank.com/challenges/detect-whether-a-linked-list-contains-a-cycle/problem**

**Solution:**

```
bool has_cycle(SinglyLinkedListNode* head) {

SinglyLinkedListNode *t = head;
SinglyLinkedListNode *r = head;
if(head == NULL || head->next==NULL) // Condition 1
{
    return false;
}
while( r!=NULL&&r->next!=NULL) // Condition 2
{
    t = t->next; // Tortoise node
    r = r->next->next; // Hare node
    if(t==r)  // Condition 3
    {
        return true;
        break;
    }

}
return false;
}
```

**Complete following programs:**

**9. WAP to create the doubly linked list of n nodes.**

**Solution:**

```
#include <iostream>
using namespace std;
struct Node
{
   int data;
   struct Node *prev;
   struct Node *next;
};
struct Node *head = NULL;
void insert(int newdata)
{
   struct Node *newnode = new struct Node;
   newnode->data = newdata;
   newnode->prev = NULL;
   newnode->next = head;
   if (head != NULL)
      head->prev = newnode;
   head = newnode;
}
void display()
{
   struct Node *ptr;
   ptr = head;
```

```
    while (ptr != NULL)
    {
        cout << ptr->data << " ";
        ptr = ptr->next;
    }
}
int main()
{
    int n,num;
    cout<<"Enter the value for n : ";
    cin>>n;
    for(int i=0;i<n;i++)
    {
    cout<<"\nEnter a number : ";
    cin>>num;
    insert(num);
    }
    cout << "\nThe doubly linked list is: ";
    display();
    return 0;
}
```

**10. Write a menu driven program for implementing doubly linked list.**

**1. To insert new node at beginning,**

**2. To insert new node after specified position**

**3. To insert new node at the end**

**4. To delete the node from beginning**

**5. To delete after specified position**

**6. To delete from the end**

```
#include <iostream>
using namespace std;
```

**Solution:**

```
#include <stdlib.h>
struct node
{
   int info;
   struct node *prev, *next;
};
struct node *start = NULL;
void traverse()
{

   if (start == NULL)
   {
      cout<<"\nList is empty\n";
      return;
   }
   struct node *temp;
   temp = start;
   while (temp != NULL)
   {
      cout<<"Data = "<<temp->info;
      temp = temp->next;
   }
}

void insertAtFront()
{
   int data;
   struct node *temp;
```

```
    temp = (struct node *)malloc(sizeof(struct node));
    cout<<"\nEnter number to be inserted: ";
    cin>>data;
    temp->info = data;
    temp->prev = NULL;
    temp->next = start;
    start = temp;
}
void insertAtEnd()
{
    int data;
    struct node *temp, *trav;
    temp = (struct node *)malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    cout<<"\nEnter number to be inserted: ";
    cin>>data;
    temp->info = data;
    temp->next = NULL;
    trav = start;
    if (start == NULL)
    {

        start = temp;
    }
    else
    {
        while (trav->next != NULL)
            trav = trav->next;
        temp->prev = trav;
        trav->next = temp;
    }
}
void insertAtPosition()
{
    int data, pos, i = 1;
    struct node *temp, *newnode;
    newnode = new struct node;
    newnode->next = NULL;
    newnode->prev = NULL;
    cout<<"\nEnter position : ";
    cin>>pos;
    cout<<"\nEnter number to be inserted: ";
    cin>>data;
    newnode->info = data;
    temp = start;
    if (start == NULL)
    {
        start = newnode;
        newnode->prev = NULL;
        newnode->next = NULL;
    }
    else if (pos == 1)
    {
        newnode->next = start;
        newnode->next->prev = newnode;
        newnode->prev = NULL;
        start = newnode;
    }
    else
    {
        while (i < pos - 1)
        {
            temp = temp->next;
            i++;
        }
        newnode->next = temp->next;
        newnode->prev = temp;
        temp->next = newnode;
        temp->next->prev = newnode;
    }
}
void deleteFirst()
{
    struct node *temp;
    if (start == NULL)
        cout<<"\nList is empty\n";
    else
    {
        temp = start;
        start = start->next;
        if (start != NULL)
            start->prev = NULL;
        delete(temp);
    }
}
void deleteEnd()
{
    struct node *temp;
    if (start == NULL)
```

```
        cout<<"\nList is empty\n";
    temp = start;
    while (temp->next != NULL)
        temp = temp->next;
    if (start->next == NULL)
        start = NULL;
    else
    {
        temp->prev->next = NULL;
        delete(temp);
    }
}
void deletePosition()
{
    int pos, i = 1;
    struct node *temp, *position;
    temp = start;
    if (start == NULL)
        cout<<"\nList is empty\n";
    else
    {
        cout<<"\nEnter position : ";
        cin>>pos;
        if (pos == 1)
        {
            position = start;
            start = start->next;
            if (start != NULL)
            {
                start->prev = NULL;
            }
            free(position);
            return;
        }
        while (i < pos - 1)
        {
            temp = temp->next;
            i++;
        }
        position = temp->next;
        if (position->next != NULL)
            position->next->prev = temp;
        temp->next = position->next;
        delete(position);
    }
}
int main()
{
    int choice;
    while (1)
    {

        cout<<"\n\t1 To print the list\n";
        cout<<"\t2 For insertion at"
               " starting\n";
        cout<<"\t3 For insertion at"
               " end\n";
        cout<<"\t4 For insertion at "
               "any position\n";
        cout<<"\t5 For deletion of "
```

```
       "first element\n";
cout<<"\t6 For deletion of "
       "last element\n";
cout<<"\t7 For deletion of "
       "element at any position\n";
cout<<"\t8 To exit\n";
cout<<"\nEnter Choice :\n";
cin>>choice;

switch (choice)
{
case 1:
   traverse();
   break;
case 2:
   insertAtFront();
   break;
case 3:
   insertAtEnd();
   break;
case 4:
   insertAtPosition();
   break;
case 5:
   deleteFirst();
   break;
case 6:
   deleteEnd();
   break;
case 7:
```

```
            deletePosition();
            break;

        case 8:
            exit(1);
            break;
        default:
            cout<<"Incorrect Choice. Try Again \n";
            continue;
        }
    }
    return 0;
}
```

**11. WAP to create circular linked list of n nodes.**

**Solution:**

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;

/*
 * Basic structure of Node
 */
struct node
{
    int data;
    struct node *next;
} * head;
```

```
/*
 * Functions used in this program
 */
void createList(int n);
void displayList();

int main()
{
   int n, data, choice = 1;

   head = NULL;

   /*
    * Run forever until user chooses 0
    */
   while (choice != 0)
   {
      cout << "========================================\n";
      cout << "CIRCULAR LINKED LIST PROGRAM\n";
      cout << "========================================\n";
      cout << "1. Create List\n";
      cout << "2. Display list\n";
      cout << "0. Exit\n";
      cout << "----------------------------------------\n";
      cout << "Enter your choice : ";

      cin >> choice;

      switch (choice)
      {
      case 1:
          cout << "Enter the total number of nodes in list: ";
          cin >> n;
          createList(n);
          break;
      case 2:
          displayList();
          break;
      case 0:
          break;
      default:
          cout << "Error! Invalid choice. Please choose between 0-2";
      }

      cout << "\n";
   }

   return 0;
}

void createList(int n)
{
   int i, data;
   struct node *prevNode, *newNode;

   if (n >= 1)
   {
      /*
       * Creates and links the head node
       */
```

```
head = (struct node *)malloc(sizeof(struct node));

cout << "Enter data of 1 node: ";
cin >> data;

head->data = data;
head->next = NULL;

prevNode = head;

/*
 * Creates and links rest of the n-1 nodes
 */
for (i = 2; i <= n; i++)
{
    newNode = (struct node *)malloc(sizeof(struct node));

    cout << "Enter data of " << i << "node : ";
    cin >> data;

    newNode->data = data;
    newNode->next = NULL;

    // Link the previous node with newly created node
    prevNode->next = newNode;

    // Move the previous node ahead
    prevNode = newNode;
}
```

```
        // Link the last node with first node
        prevNode->next = head;

        cout << "\nCIRCULAR LINKED LIST CREATED SUCCESSFULLY\n";
    }
}

/**
 * Display the content of the list
 */
void displayList()
{
    struct node *current;
    int n = 1;

    if (head == NULL)
    {
        cout << "List is empty.\n";
    }
    else
    {
        current = head;
        cout << "DATA IN THE LIST:\n";

        do
        {
            cout << "Data " << n << "=" << current->data << endl;

            current = current->next;
            n++;
        } while (current != head);
    }
}
```

**12. WAP to count the number of nodes in circular linked list if only start pointer of circular linked list is given.**

**Solution:**

```
#include <iostream>
using namespace std;
struct node
{
    int data;
    node *next;
    node(int x)
    {
        data = x;
        next = NULL;
    }
};
struct node *push(struct node *last, int data)
{
    if (last == NULL)
    {
        struct node *temp = (struct node *)malloc(sizeof(struct node));
        temp->data = data;
        last = temp;
        temp->next = last;

        return last;
    }
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = data;
    temp->next = last->next;
    last->next = temp;

    return last;
}

int count_Nodes(node *head)
{
    node *temp = head;
    int result = 0;
    if (head != NULL)
    {
        do
        {
            temp = temp->next;
            result++;
        } while (temp != head);
    }

    return result;
}
int main()
{
    node *head = NULL;
    head = push(head, 0);
    head = push(head, 84);
    head = push(head, 4);
    head = push(head, 8);
    cout << count_Nodes(head);
    return 0;
}
```