Lab 7: Implementation of SLL.

Theory: A *linked list* is an ordered collection of data in which each element (node) contains a minimum of two values, *data* and *link*(*s*) to its successor (and/or predecessor). A list with one link field using which every element is associated to its successor is known as a *singly linked list*.

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int info;
    struct node *next;
};
typedef struct node NodeType;
NodeType *first,*last;
void insertatbegin(int);
void insertatend(int);
void insertatspecific(int);
void deleteatbegin();
void deleteatend();
void deleteatspecific();
void display();
int main()
{
    first=last=NULL;
    int choice,item;
    do
    {
        printf("\n");
```

```
printf("Select a option. \n");
printf("1. INSERT ELEMENT AT BEGINNING\n");
printf("2. INSERT ELEMENT AT END\n");
printf("3. INSERT ELEMENT AT SPECIFIED POSITION\n");
printf("4. DELETE ELEMENT AT BEGINNING\n");
printf("5. DELETE ELEMENT AT END\n");
printf("6. DELETE ELEMENT AT SPECIFIED POSITION\n");
printf("7. DISPLAY ELEMENTS\n");
printf("8. EXIT\n");
printf(">");
scanf("%d",&choice);
switch (choice)
{
    case 1:
        printf("Enter item to be inserted : ");
        scanf("%d",&item);
        insertatbegin(item);
        break;

    case 2:
        printf("Enter item to be inserted : ");
        scanf("%d",&item);
        insertatend(item);
        break;

    case 3:
        printf("Enter item to be inserted : ");
```

```
                scanf("%d",&item);
                insertatspecific(item);
                break;

            case 4:
                deleteatbegin();
                break;

            case 5:
                deleteatend();
                break;

            case 6:
                deleteatspecific();
                break;

            case 7:
                display();
                break;


        }
        }while (choice!= 8);
        return 0;
    return 0;
}
```

```
void insertatbegin(int item)
{
    NodeType *NewNode;
    NewNode=(NodeType*)malloc(sizeof(NodeType));
    NewNode->info=item;
    NewNode->next=NULL;
    if(first==NULL)
    {
        NewNode->next=NULL;
        first=NewNode;
        last=NewNode;
    }
    else
    {
        NewNode->next=first;
        first=NewNode;
    }
    printf("%d inserted at beginning \n",NewNode->info);
}

void insertatend(int item)
{
    NodeType *NewNode;
    NewNode=(NodeType*)malloc(sizeof(NodeType));
    NewNode->info=item;
    NewNode->next=NULL;
    if(first==NULL)
```

```
    {
        NewNode->next=NULL;
        first=NewNode;
        last=NewNode;
    }
    else
    {
        last->next=NewNode;
        last=NewNode;
    }
     printf("%d inserted at end \n",NewNode->info);
}

void insertatspecific(int item)
{
    int pos,i;
    NodeType *NewNode,*temp;
    NewNode=(NodeType*)malloc(sizeof(NodeType));
    NewNode->info=item;
    NewNode->next=NULL;
    printf("Enter Position : ");
    scanf("%d",&pos);
    if(first==NULL)
    {
        NewNode->next=NULL;
        first=NewNode;
        last=NewNode;
```

```
        printf("%d inserted at beginning \n",NewNode->info);
    }
    else
    {
        temp=first;
        for(i=1;i<(pos-1);i++)
        {
            temp=temp->next;
        }
        NewNode->next=temp->next;
        temp->next=NewNode;
        printf("%d inserted at position %d \n",NewNode-
>info,pos);
    }
}

void deleteatbegin()
{
    NodeType *temp;
    if(first==NULL)
    {
        printf("List is Empty \n");
    }
    else
    {
        temp=first;
        printf("%d deleted from beginning\n",temp->info);
```

```
        first=first->next;
        free(temp);
    }
}

void deleteatend()
{
    NodeType *temp,*hold;
    if(first==NULL)
    {
        printf("List is Empty \n");
    }
    else if(first==last)
    {
       printf("Only one node\n");
       hold=first;
       first=NULL;
       last=NULL;
       printf("%d deleted \n",hold->info);
       free(hold);
    }
    else
    {
        temp=first;
        hold=first;
        while(temp->next!=last)
        {
```

```
            temp=temp->next;
        }
        hold=temp->next;
        temp->next=NULL;
        last=temp;
        printf("%d deleted from end\n",hold->info);
        free(hold);
    }
}

void deleteatspecific()
{
    int pos,i;
    NodeType *temp,*hold;
    if(first==NULL)
    {
        printf("List is Empty \n");
    }
    else if(first==last)
    {
       printf("Only one node\n");
       hold=first;
       first=NULL;
       last=NULL;
       printf("%d deleted\n",hold->info);
       free(hold);
    }
```

```
    else
    {
        temp=first;
        printf("Enter Position : ");
        scanf("%d",&pos);
        for(i=1;i<(pos-1);i++)
        {
            temp=temp->next;
        }
        hold=temp->next;
        temp->next=hold->next;
        printf("%d deleted from position %d ",hold-
>info,pos);
        free(hold);
    }
}

void display()
{
    NodeType *temp;
    temp=first;
    if(first==NULL)
    {
        printf("Empty List... \n");
    }
    else
    {
```

```
        while(temp!=NULL)
        {
            printf("%d ->",temp->info);
            temp=temp->next;
        }
      printf("NULL\n");
    }
}
```

Conclusion: In this lab we implemented different insertion and deletion operations on SLL.