NAME:- SAGAR ADHIKARI
ROLL NO:- PUR078BEI034
FACULTY:- ELECTRONICS(BEI)
SUBMISSION DATE:- 2080/09/

DSA LAB SHEET NO. 4

TITLE:- IMPLEMENTATION OF SINGLY LINKED LIST(SLL)

THEORY:-

List:-list is an ordered data structure that stores elements sequentially and can be accessed by the index of the elements.
Linked List:- A linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.
Singly Linked List:-A singly linked list is a linear data structure in which elements are not stored at a contiguous location. Hence, singly linked lists are not connected in sequence, like arrays, but are linked together by pointers. They typically consist of node(s). Each node has two components – a data field and a pointer.

PROGRAM CODE:-

```
#include <stdio.h>
#include <stdlib.h>
struct SLL
{
    int data;
    struct SLL *next; // self referential structure
};
struct SLL *first, *last, *temp;

void insertAtFront(int element)
{
    struct SLL *NewNode = (struct SLL *)malloc(sizeof(struct SLL)); // allocation of memory for
                                                                     // NewNode creation
    if (NewNode == NULL)
    {
        printf("Memory allocation failed\n"); // no element in NewNode
    }
    else
    {
        NewNode->data = element;
        NewNode->next = NULL;
        if (first == NULL) // List is empty
        {
            first = last = NewNode;
        }
        else
        {
            NewNode->next = first;
            first = NewNode;
        }
        printf("%d was inserted at the front.\n", first->data);
    }
}
void insertAtLast(int element)
{
    struct SLL *NewNode = (struct SLL *)malloc(sizeof(struct SLL));
    if (NewNode == NULL)
    {
        printf("Memory allocation failed/n");
    }
    else
    {
        NewNode->data = element;
        NewNode->next = NULL;
        if (first == NULL) // List is empty
        {
            first = last = NewNode;
        }
        else
        {
            last->next = NewNode;
            last = NewNode;
        }
        printf("%d was inserted at the last\n", last->data);
    }
}
int position;
void insertAtPos(int element)
{

    printf("Enter position to be inserted:- ");
    scanf("%d", &position);
    struct SLL *NewNode = (struct SLL *)malloc(sizeof(struct SLL));
    if (NewNode == NULL)
    {
        printf("Memory allocation failed\n");
    }
    else
    {
        NewNode->data = element;
        NewNode->next = NULL;
        temp = first;
        for (int i = 1; i < position - 1; i++)
        {
            temp = temp->next;
        }
        if (temp->next == NULL) // if position is after last element in list
        {
            temp->next = NewNode;
            last = NewNode;
        }
        else
        {
            NewNode->next = temp->next;
            temp->next = NewNode;
        }
        printf("%d was inserted in %d position\n",element,position);
    }
}
void deleteFromFront()
{
    struct SLL *temp;
    if (first == NULL)
    {
        printf("List Empty so failed to delete");
    }
    else if (first->next == NULL) // 1 element only
    {
        first = last = NULL;
    }
    else
    {
        temp = first;
        first = first->next;
        free(temp);
    }
}
void deleteFromEnd()
{
    if (first == NULL)
    {
        printf("List Empty so failed to delete");
    }
    else if (first->next == NULL) // contains only one element
    {
        first = last = NULL;
    }
    else
    {
        temp = first;
        while (temp->next != last) // reaches to 2nd last node
        {
            temp = temp->next;
        }
        last = temp;
        temp = last->next;
        last->next = NULL;
        free(temp);
    }
}
void deleteFromPos()
{
    if (first == NULL)
    {
        printf("List Empty so failed to delete");
    }
    else if (first->next == NULL) // if only one element
    {
        first = last = NULL;
    }
    else
    {
        printf("Enter position to be deleted:- ");
        scanf("%d", &position);
        temp = first;
        for (int i = 1; i < position - 1; i++)
        {
            temp = temp->next;
        }
        struct SLL *temp1 = temp->next; //temp1 introduced to cope with node that are behind the
                                        //deleting node
        temp->next = temp1->next;       // 'position-1 next' equals to 'position next'
        free(temp1);
    }
}
void display()
{
    temp = first;
    if (first == NULL)
    {
        printf("Empty list\n");
    }
    else
    {
        while (temp->next != NULL)
        {
            printf("%d ->", temp->data); // prints till 2nd last node
            temp = temp->next;           // runs till last node
        }
        printf("%d-> NULL\n", temp->data); // prints the last node pointing NULL
    }
}
int main()
{
    int inElement, option, choice;
    do
    {
        printf("\nEnter problem to be conducted:- ");
        printf("\n1.Insertion of element\n2.Deletion of element\n");
        scanf("%d", &choice);
        if (choice == 1)
        {
            printf("Enter the element to insert:- ");
            scanf("%d", &inElement);
            printf("1.Insert from Front\n2.Insert from end\n3.Insert from any position\n");
            scanf("%d", &option);
            switch (option)
            {
            case 1:
                insertAtFront(inElement);
                display();
                break;
            case 2:
                insertAtLast(inElement);
                display();
                break;
            case 3:
                insertAtPos(inElement);
                display();
                break;
            default:
                printf("Only enter 1 or 2 or 3\n");
                break;
            }
        }
        else if (choice == 2)
        {
            printf("1.Delete from Front\n2.Delete from end\n3.Delete from any position\n");
            scanf("%d", &option);
            switch (option)
            {
            case 1:
                deleteFromFront();
                display();
                break;
            case 2:
                deleteFromEnd();
                display();
                break;
            case 3:
                deleteFromPos();
                display();
                break;
            default:
                printf("Only enter 1 or 2 or 3 \n");
                break;
            }
        }
    } while (choice == 1||2);
    return 0;
}
```

OUTPUT:-

```
Enter problem to be conducted:-
1.Insertion of element
2.Deletion of element
1
Enter the element to insert:- 20
1.Insert from Front
2.Insert from end
3.Insert from any position
1
20 was inserted at the front.
20-> NULL

Enter problem to be conducted:-
1.Insertion of element
2.Deletion of element
1
Enter the element to insert:- 10
1.Insert from Front
2.Insert from end
3.Insert from any position
1
10 was inserted at the front.
10 ->20-> NULL

Enter problem to be conducted:-
1.Insertion of element
2.Deletion of element
1
Enter the element to insert:- 30
```

```
1.Insert from Front
2.Insert from end
3.Insert from any position
2
30 was inserted at the last
10 ->20 ->30-> NULL

Enter problem to be conducted:-
1.Insertion of element
2.Deletion of element
1
Enter the element to insert:- 40
1.Insert from Front
2.Insert from end
3.Insert from any position
3
Enter position to be inserted:- 4
40 was inserted in 4 position
10 ->20 ->30 ->40-> NULL

Enter problem to be conducted:-
1.Insertion of element
2.Deletion of element
2
1.Delete from Front
2.Delete from end
3.Delete from any position
3
Enter position to be deleted:- 2
10 ->30 ->40-> NULL

Enter problem to be conducted:-
1.Insertion of element
2.Deletion of element
2
1.Delete from Front
2.Delete from end
3.Delete from any position
1
30 ->40-> NULL
```

```
Enter problem to be conducted:-
1.Insertion of element
2.Deletion of element
2
1.Delete from Front
2.Delete from end
3.Delete from any position
2
30-> NULL

Enter problem to be conducted:-
1.Insertion of element
2.Deletion of element
2
1.Delete from Front
2.Delete from end
3.Delete from any position
1
Empty list

Enter problem to be conducted:-
1.Insertion of element
2.Deletion of element
```