**PROGRAMS: Singly Linked List**

**Input:**

#include <stdio.h>

//node structure

struct node

{

    int data;

    struct node \*link;

};

//start declaration

struct node \*start = NULL;

int main()

{

    printf("This is the implementation of different operations that can be performed on Linked List\n\*\*\*\n\n");

    int option;

    do

    {

        printf("Enter the operation you want to perform\n");

        printf("Enter 0 to stop\nEnter 1 to print the list\nEnter 2 to Search an element in the list\n");

        printf("Enter 3 to insert an element in an empty list\nEnter 4 to insert an element at the beginning of the list\nEnter 5 to insert an element at the end of the list\n");

        printf("Enter 6 to insert an element before an element in the list\nEnter 7 to insert an element after an element in the list\nEnter 8 to insert an element at a given position in the list\n");

        printf("Enter 9 to delete an element from the beginning\nEnter 10 to delete the only node in the list\nEnter 11 to delete an element from the end of the list\n");

Expt no: 4

        printf("Enter 12 to delete an element before an element in the list\nEnter 13 to delete an element after an element in the list\nEnter 14 to delete an element from a given position in the list\n");

        printf("Enter 15 to reverse the list\n\n");

        printf("Enter 16 to exit\n\n");

        printf("Enter your option : ");

        scanf("%d", &option);

        switch(option)

        {

        case 0:

            printf("Program Ended\n");

            break;

        case 1:

            print\_list();

            break;

        case 2:

            search\_list();

            break;

        case 3:

            insert\_in\_empty\_list();

            break;

        case 4:

            insert\_at\_beginning();

            break;

        case 5:

            insert\_at\_end();

            break;

        case 6:

            insert\_before();

            break;

        case 7:

            insert\_after();

            break;

        case 8:

            insert\_at\_position();

            break;

        case 9:

            delete\_from\_beginning();

            break;

        case 10:

            delete\_only\_element();

            break;

        case 11:

            delete\_from\_end();

            break;

        case 12:

            delete\_before();

            break;

        case 13:

            delete\_after();

            break;

        case 14:

            delete\_from\_position();

            break;

        case 15:

            reverse\_list();

            break;

case 16:

exit(0);

break;

        }

    }

    while(option > 0 && option < 16);

}

int count()

{

    struct node \*p = start;

    int count = 0;

    while(p != NULL)

    {

        count++;

        p = p->link;

    }

    return count;

Expt no: 4

}

void search\_list()

{

    struct node \*p = start;

    int key, i, c = count();

    printf("Enter the element you want to search in the list : ");

    scanf("%d", &key);

    for(i = 1; i <= c; i++)

    {

        if(p->data == key)

        {

            printf("Number %d found at position %d.\n\n", key, i);

            return;

        }

        p = p->link;

    }

    if (i > c)

    {

        printf("Number not found\n\n");

        return;

    }

}

void print\_list()

{

    struct node \*p = start;

    if (start == NULL)

    {

        printf("List is empty.\n\n");

        return;

    }

    printf("List is : \n");

    while (p != NULL)

    {

        printf("%d\t", p->data);

        p = p->link;

    }

    printf("\n\n");

}

void insert\_in\_empty\_list()

{

    if (start == NULL)

    {

        struct node \*tmp;

        int x;

        tmp = (struct node \*)malloc(sizeof(struct node));

        printf("Enter the number you want to add : ");

        scanf("%d", &x);

        printf("\n\n");

        tmp->data = x;

        tmp->link = start;

        start = tmp;

    }

    else

    {

        printf("The list is not empty\nPlease select option 4 - 8 in order to add new elements in existing list\n\n");

        return;

    }

}

void insert\_at\_beginning()

{

    if(start == NULL)

        insert\_in\_empty\_list();

    else

    {

        struct node \*tmp = (struct node \*)malloc(sizeof(struct node));

        int x;

        printf("Enter the number you want to add : ");

        scanf("%d", &x);

        printf("\n\n");

        tmp->data = x;

        tmp->link = start;

        start = tmp;

    }

Expt no: 4

}

void insert\_at\_end()

{

    struct node \*p = start;

    if (start == NULL)

    {

        insert\_at\_beginning();

        return;

    }

    while(p->link != NULL)

        p = p->link;

    struct node \*tmp = (struct node \*)malloc(sizeof(struct node));

    int x;

    printf("Enter the number you want to add : ");

    scanf("%d", &x);

    printf("\n\n");

    tmp->data = x;

    tmp->link = p->link;

    p->link = tmp;

}

void insert\_before()

{

    struct node \*p = start;

    int target;

    printf("Enter the element before which you want to add new element : ");

    scanf("%d", &target);

    while(p != NULL)

    {

        if(p->link->data == target)

        {

            struct node \*tmp = (struct node \*)malloc(sizeof(struct node));

            int x;

            printf("Enter the number you want to add : ");

            scanf("%d", &x);

            printf("\n\n");

            tmp->data = x;

            tmp->link = p->link;

            p->link = tmp;

            return;

        }

        p = p->link;

    }

    printf("Element not found in the list\n\n");

}

void insert\_after()

{

    struct node \*p = start;

    int target;

    printf("Enter the element after which you want to add new element : ");

    scanf("%d", &target);

    while(p != NULL)

    {

        if(p->data == target)

        {

            struct node \*tmp = (struct node \*)malloc(sizeof(struct node));

            int x;

            printf("Enter the number you want to add : ");

            scanf("%d", &x);

            printf("\n\n");

            tmp->data = x;

            tmp->link = p->link;

            p->link = tmp;

            return;

        }

        p = p->link;

    }

    printf("Element not found in the list\n\n");

}

void insert\_at\_position()

{

    struct node \*p = start;

    struct node \*prev = NULL;

Expt no: 4

    int pos, i, j = 1, c = count();

    printf("Enter the position at which you want to add new element : ");

    scanf("%d", &pos);

    if(pos == 0)

    {

        printf("We refer natural number indexing starting from 1.\n");

    }

    if (pos == 1)

    {

        insert\_at\_beginning();

        return;

    }

    for (i = 1; i <= c + 1; i++)

    {

        if (i == pos)

        {

            struct node \*tmp = (struct node \*)malloc(sizeof(struct node));

            int x;

            printf("Enter the number you want to add : ");

            scanf("%d", &x);

            printf("\n\n");

            tmp->data = x;

            tmp->link = p->link;

            p->link = tmp;

            return;

        }

        if(i == 1)

            continue;

        else

            p = p->link;

    }

    if(i > c + 1)

    {

        printf("Out of bound.\n\n");

        return;

    }

}

void delete\_from\_beginning()

{

    if (start == NULL)

    {

        printf("The List is empty.\n\n");

        return;

    }

    if (start->link == NULL)

    {

        delete\_only\_element();

        return;

    }

    start = start->link;

}

void delete\_only\_element()

{

    if (start == NULL)

    {

        printf("The List is empty.\n\n");

        return;

    }

    if (count() > 1)

    {

        printf("There are more than one elements in the list.\n\n");

        return;

    }

    if (start->link == NULL)

    {

        start = NULL;

        return;

    }

}

void delete\_from\_end()

{

    if (start == NULL)

    {

        printf("The List is empty.\n\n");

        return;

Expt no: 4

    }

    if (start->link == NULL)

    {

        delete\_only\_element();

        return;

    }

    struct node \*p = start;

    while (p->link->link != NULL)

        p = p->link;

    p->link = NULL;

}

void delete\_before()

{

    struct node \*p = start;

    int target;

    printf("Enter the element before which you want to delete : ");

    scanf("%d", &target);

    while(p != NULL)

    {

        if(p->link->link->data == target)

        {

            struct node \*tmp = p->link->link;

            p->link = tmp;

            return;

        }

        p = p->link;

    }

    printf("Element not found in the list\n\n");

}

void delete\_after()

{

    struct node \*p = start;

    int target;

    printf("Enter the element after which you want to delete : ");

    scanf("%d", &target);

    while(p != NULL)

    {

        if(p->data == target)

        {

            struct node \*tmp = p->link->link;

            p->link = tmp;

            return;

        }

        p = p->link;

    }

    printf("Element not found in the list\n\n");

}

void delete\_from\_position()

{

    struct node \*p = start;

    struct node \*prev = NULL;

    int pos, i, j = 1, c = count();

    printf("Enter the position from which you want to delete element : ");

    scanf("%d", &pos);

    if(pos == 0)

    {

        printf("We refer natural number indexing starting from 1.\n");

    }

    if (pos == 1)

    {

        delete\_from\_beginning();

        return;

    }

    for (i = 1; i <= c; i++)

    {

        if (i == pos)

        {

            struct node \*tmp = p->link->link;

            p->link = tmp;

            return;

        }

        if(i == 1)

            continue;

Expt no: 4

        else

            p = p->link;

    }

    if(i > c)

    {

        printf("Out of bound.\n\n");

        return;

    }

}

void reverse\_list()

{

    struct node \*p = start, \*prev = NULL, \*next = NULL;

    while (p != NULL)

    {

        next = p->link;

        p->link = prev;

        prev = p;

        p = next;

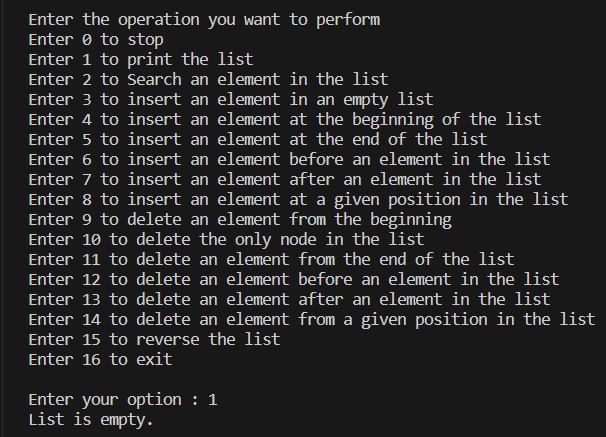
    }

    start = prev;

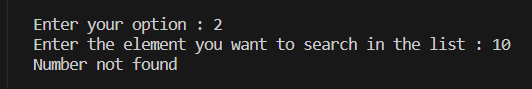
}

**Output:**

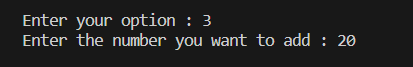
1. To print the list(empty):



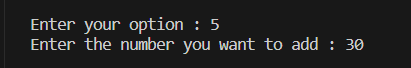
1. To search for an element(empty list):



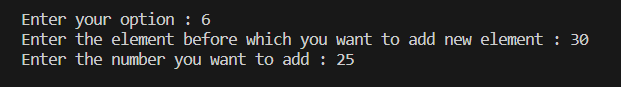
1. To insert a number in an empty list:



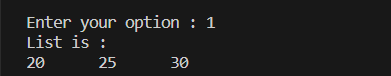
1. To insert an element at the end of the list:



1. To insert an element before an element in the list:

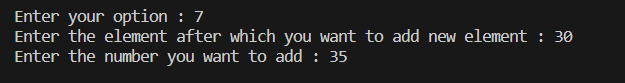


1. To display the list:

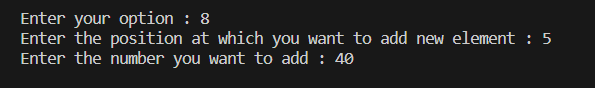


1. To insert an element after an element in the list:

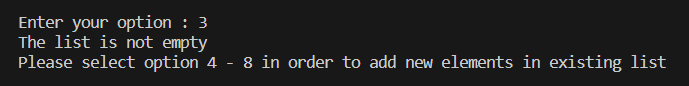
Expt no: 4



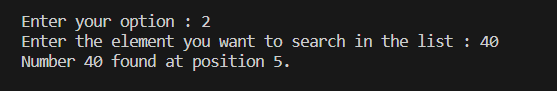
1. To insert an element at a given position:



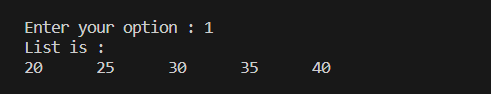
1. To insert an element in an empty list(list not empty):



1. To search for an element(empty list):



1. To print the list:



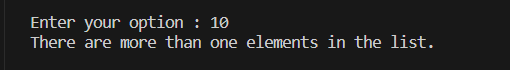
1. To delete an element from the beginning:



1. To print the list:



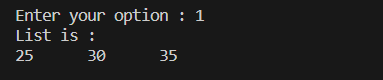
1. To delete the only node in the list(multiple nodes present):



1. To delete an element from the end of the list:



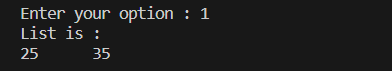
1. To print the list:



1. To delete an element before an element in the list:



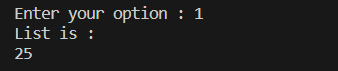
1. To print the list:



1. To delete an element from a given position in the list:



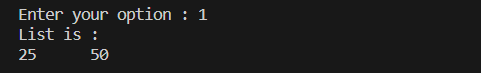
1. To print the list:



1. To insert an element at the end of the list:



1. To print the list:



1. To reverse the list:



1. To print the list:

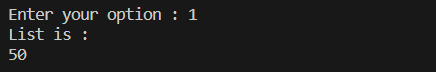


1. To delete an element from the end of the list:



Expt no: 4

1. To print the list:



1. To delete the only node in the list:



1. To print the list:

