**DS Lab 5**

Name : Vivian Vijay Ludrick

Branch : SE Comps A Batch C

Roll No: **9914**

**Code:**

#include <stdio.h>

#include <stdlib.h>

*// structure of a node*

typedef struct node

{

int data;

struct node \*next;

} Node;

*// structure of a node pointer*

typedef struct

{

Node \*start;

} LL;

*// append the node at the last index*

void appendNode(LL \**l*, int *ele*)

{

Node \*p = (Node \*)malloc(sizeof(Node)); *// creating a node to append to the linked list*

p->data = *ele*;

p->next = NULL;

*// if no nodes in LL-> append to start*

if (*l*->start == NULL)

{

*l*->start = p;

}

*// ->append at end*

else

{

Node \*q = *l*->start; *// used to store the address of the last node*

*// loop till end*

while (q->next != NULL)

{

q = q->next; *// q takes the address of stored in its next*

}

*// once reached end append p*

q->next = p;

}

}

*// add a node in the beginning*

void addBegin(LL \**l*, int *ele*)

{

Node \*p = (Node \*)malloc(sizeof(Node)); *// creating a node to append to the linked list*

p->data = *ele*;

*// editing the address values*

p->next = *l*->start;

*l*->start = p;

}

*// adding the node at the nth index*

void addMid(LL \**l*, int *n*, int *ele*)

{

Node \*p = (Node \*)malloc(sizeof(Node)); *// creating a node to append to the linked list*

p->data = *ele*;

p->next = NULL; *// since we dont know what the next address will be*

Node \*q = *l*->start; *// pointer to store the address we want*

*/\**

*what address do you have? => 1 node => i = 1*

*what address you want? => nth node => where is it? => (n-1).next=> therefore loop till n-1 i.e. till <n*

*\*/*

*// loops till we find the nth node's address*

for (int i = 1; i < *n*; i++)

{

q = q->next;

}

*// editing the addresses to insert in between*

p->next = q->next;

q->next = p;

}

*// count the number of nodes*

int countNodes(LL *l*)

{

Node \*q = *l*.start; *// pointer to point next at each iteration*

int count = 0;

while (q != NULL)

{

count++; *// increment if node found*

q = q->next; *// go to next address*

}

return count;

}

*// delete the node via its element*

void deleteNode(LL \**l*, int *ele*)

{

Node \*prev; *// pointer to store the address of the previous element to change the address when the node is found*

Node \*q = *l*->start; *// pointer to store the address of the required element*

*// while you dont reach the end*

while (q != NULL)

{

*// if data matches the element end loop*

if (q->data == *ele*)

{

break;

}

prev = q;

q = q->next;

}

*// address == NULL => there is no node*

if (q == NULL)

{

printf("No node with %d exists in the linked list\n", *ele*);

}

else

{

if (prev == NULL)

{

*l*->start = q->next;

}

else

{

*// change addresses to ignore the node*

prev->next = q->next;

}

*// freeing the node which has the required data part*

free(q);

}

}

*// concatenate two linked lists into first linked list*

void concat(LL \**l1*, LL *l2*)

{

*// l2 has no nodes*

if (*l2*.start == NULL)

{

return;

}

*// l1 empty => then l1 = l2*

if (*l1*->start == NULL)

{

*l1*->start = *l2*.start;

}

else

{

Node \*q = *l1*->start;

*// locate last node of LL 1*

while (q->next != NULL)

{

q = q->next;

}

q->next = *l2*.start; *// link last node of l1 with first node of l2*

}

}

*// reverse a linked list*

void reverse(LL \**l*)

{

Node \*q, \*r, \*p;

p = NULL; *// previous node*

q = *l*->start; *// current node*

while (q != NULL)

{

r = q->next; *// next node into r pointer*

q->next = p; *// current node points to previous node*

p = q; *// current node becomes previous node*

q = r; *// next node becomes current node*

}

*l*->start = p; *// end of the previous ll becomes start of this ll*

}

*// display all the elements of the linked list*

void display(LL *l*)

{

Node \*q = *l*.start;

printf("Elements are:\n");

*// while node exists*

while (q != NULL)

{

printf("%d\n", q->data);

q = q->next;

}

}

void freeAll(LL \**l*)

{

Node \*q = *l*->start;

Node \*prev;

while (q != NULL)

{

prev = q;

q = q->next;

free(prev);

}

*l*->start = NULL;

}

*// start of the main method*

int main()

{

LL l1, l2, l3;

*// always initialise start to null*

l1.start = NULL;

l2.start = NULL;

l3.start = NULL;

int option, ele;

do

{

printf("Enter an option:\nA.For linked list 1-1.Insert at beginning 2 Insert at end 3.Display 4.Delete Data 5.Count Nodes\nB.For linked list 2-6.Concat 2 LL \nC.For linked list 3-7.Reverse 8.Exit\n");

scanf("%d", &option);

switch (option)

{

case 1:

printf("Enter the element:\n");

scanf("%d", &*ele*);

addBegin(&*l1*, ele);

break;

case 2:

printf("Enter the element:\n");

scanf("%d", &*ele*);

appendNode(&*l1*, ele);

break;

case 3:

display(l1);

break;

case 4:

printf("Enter the data to be deleted:\n");

scanf("%d", &*ele*);

deleteNode(&*l1*, ele);

break;

case 5:

printf("The total nodes in the linked list are :%d\n", countNodes(l1));

break;

case 6:

appendNode(&*l2*, 60);

appendNode(&*l2*, 50);

addBegin(&*l2*, 40);

concat(&*l1*, l2);

printf("The concatenated linked list is :\n");

display(l1);

break;

case 7:

appendNode(&*l3*, 80);

appendNode(&*l3*, 90);

addBegin(&*l3*, 100);

reverse(&*l3*);

display(l3);

break;

case 8:

printf("Freeing the elements in the linked list");

freeAll(&*l1*);

freeAll(&*l2*);

freeAll(&*l3*);

exit(0);

break;

default:

printf("Enter valid option\n");

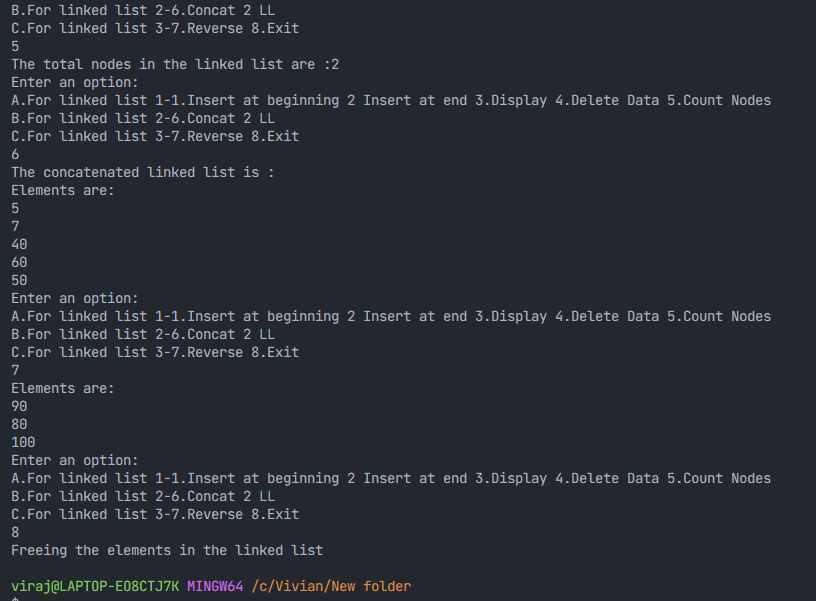
}

} while (1); *// infinite loop*

return 0;

}

**OUTPUT :**

****