**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**RACHANA N**

**2023BMS02608**

**in partial fulfillment for the award of the degree of**

**BACHELOR OF ENGINEERING**

**in**

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**Dec 2023- March 2024**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

**(Affiliated To Visvesvaraya Technological University, Belgaum)**

**Department of Computer Science and Engineering**

****

This is to certify that the Lab work entitled **“DATA STRUCTURES”** carried out by RACHANA N **(2023BMS02608)**, who is a bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST) work** prescribed for the said degree.

**Prof. Sneha S Bagalkot** **Dr. Jyothi S Nayak**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 1 | Week1 | **04-10** |
| 2 | Week2 | **11-15** |
| 3 | Week3 | **15-27** |
| 4 | Week4(Leetcode-1) | **28-37** |
| 5 | Week5(Leetcode-2) | **38-52** |
| 6 | Week6 | **53-72** |
| 7 | Week7(Leetcode-3) | **73-79** |
| 8 | Week8(Leetcode-4) | **80-91** |
| 9 | Week9(HackerRank) | **92-99** |
| 10 | Week10 | **100-103** |

**Course outcomes:**

|  |  |
| --- | --- |
| CO1 | Apply the concept of linear and nonlinear data structures. |
| CO2 | Analyze data structure operations for a given problem |
| CO3 | Design and develop solutions using the operations of linear and nonlinear data structure for a given specification. |
| CO4 | Conduct practical experiments for demonstrating the operations of different data structures. |

**Lab program 1:**

**Write a program to simulate the working of stack using an array with the following:**

**a) Push**

**b) Pop**

**c) Display**

**The program should print appropriate messages for stack overflow, stack underflow.**

#include<stdio.h>

#define SIZE 5

int stack[SIZE];

int top=-1;

void push(int element);

void pop();

void display();

int main()

{

int choice , element;

do{

printf("\n stack operation\n");

printf("1.push\n");

printf("2.pop\n");

printf("3.display\n");

printf("4.exit\n");

printf("enter your choice:");

scanf("%d",&choice);

switch(choice){

case 1:

printf("enter the element to push");

scanf("%d",&element);

push(element);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

printf("exiting the program");

break;

default:

printf("invalid choice");

}

}while(choice!=4);

return 0;

}

void push(int element){

if (top == SIZE -1)

{

printf("stack overflow");

}else

{

top ++;

stack[top]=element;

printf("%d pushed onto the stack\n",element);

}

}

void pop()

{

if (top ==-1){

printf("stack underflow");

}else{

printf("%d popped from the stack\n",stack[top]);

top--;

}

}

void display()

{

if (top==-1){

printf("stack is empty");

}else{

printf("elements in the stack\n");

for(int i=0; i<=top; i++)

{

printf("%d",stack[i]);

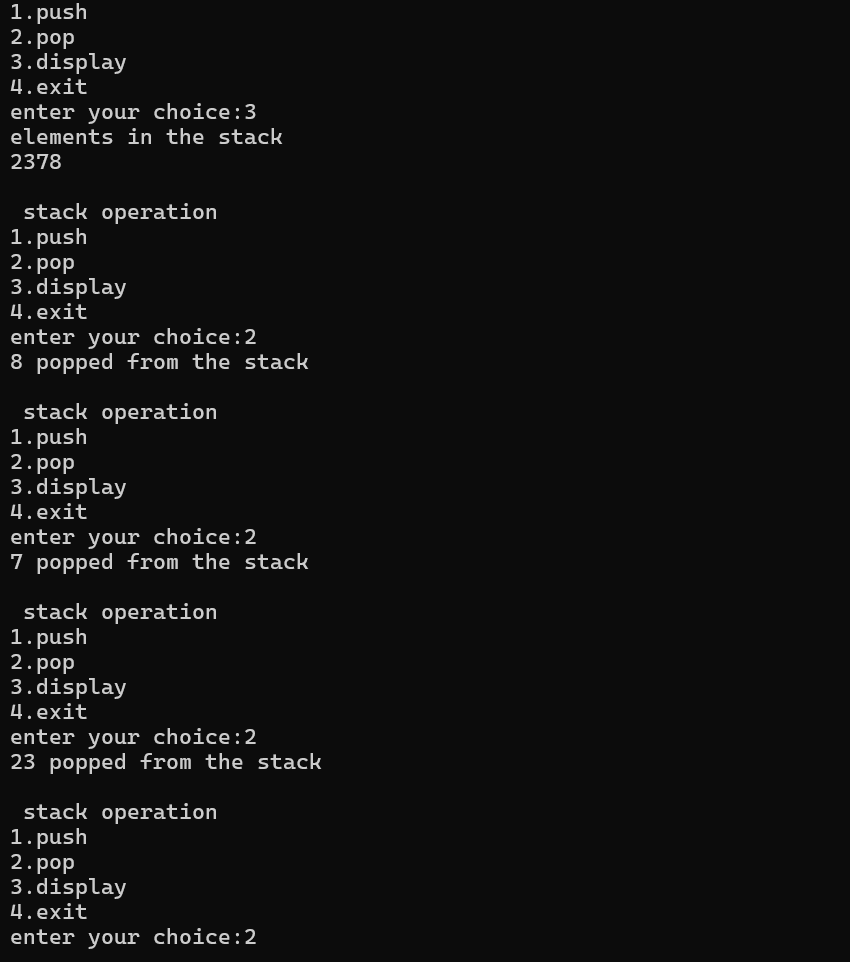
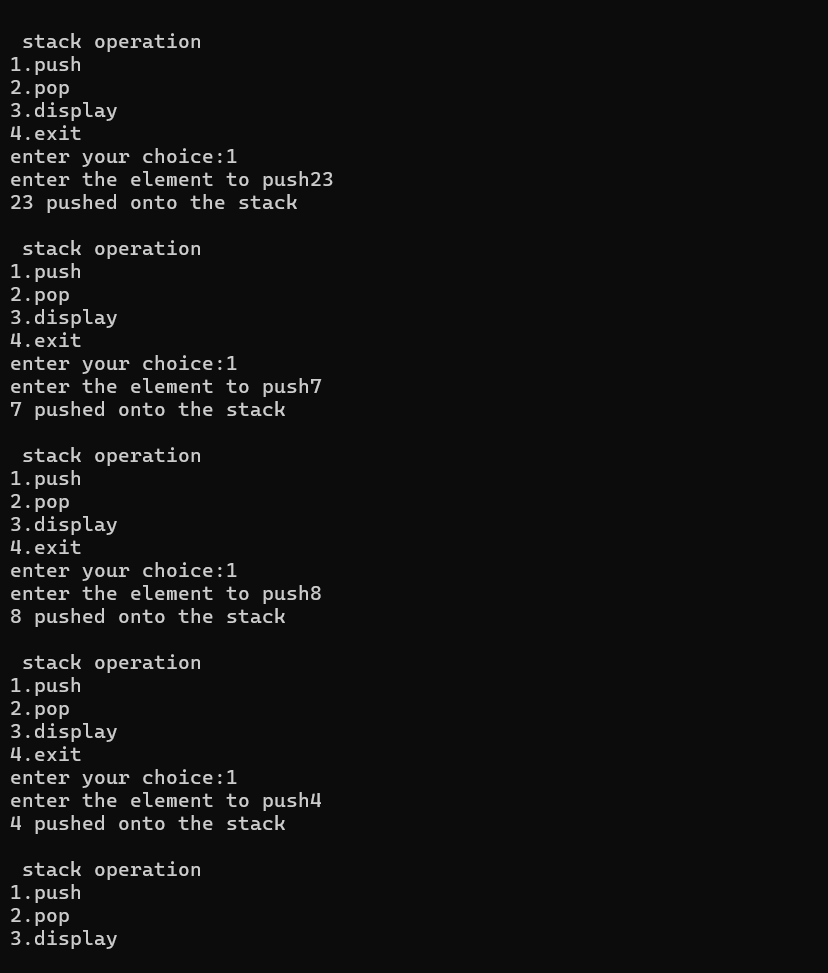
}

printf("\n");

}

}

**Output:**



Lab program 2:

**WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), \* (multiply) and /(divide).**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<string.h>**

**#define MAX 100**

**char stack[MAX];**

**char infix[MAX];**

**char postfix[MAX];**

**int top=-1;**

**void push(char);**

**char pop();**

**int isEmpty();**

**void inToPost();**

**void print();**

**int precedence(char);**

**int main()**

**{**

**printf("enter infix expression: ");**

**gets(infix);**

**inToPost();**

**print();**

**return 0;**

**}**

**void inToPost()**

**{**

**int i,j=0;**

**char symbol,next;**

**for(i=0;i<strlen(infix);i++)**

**{**

**symbol=infix[i];**

**switch(symbol)**

**{**

**case '(':**

**push(symbol);**

**break;**

**case ')':**

**while((next=pop())!='(')**

**postfix[j++]=next;**

**break;**

**case '+':**

**case '-':**

**case '\*':**

**case '/':**

**case '^':**

**while (!isEmpty() && precedence(stack[top]) >= precedence(symbol))**

**postfix[j++] = pop();**

**push(symbol);**

**break;**

**default:**

**postfix[j++] = symbol;**

**}**

**}**

**while (!isEmpty())**

**postfix[j++] = pop();**

**postfix[j] = '\0';**

**}**

**int precedence(char symbol)**

**{**

**switch (symbol)**

**{**

**case '^':**

**return 3;**

**case '/':**

**case '\*':**

**return 2;**

**case '+':**

**case '-':**

**return 1;**

**default:**

**return 0;**

**}**

**}**

**void print()**

**{**

**int i = 0;**

**printf("The equivalent postfix expression is: ");**

**while (postfix[i])**

**{**

**printf("%c ", postfix[i++]);**

**}**

**printf("\n");**

**}**

**void push(char c)**

**{**

**if(top==MAX-1)**

**{**

**printf("stack overflow");**

**return;**

**}**

**top++;**

**stack[top]=c;**

**}**

**char pop()**

**{**

**char c;**

**if(top==-1)**

**{**

**printf("stack underflow");**

**exit(1);**

**}**

**c=stack[top];**

**top=top-1;**

**return c;**

**}**

**int isEmpty()**

**{**

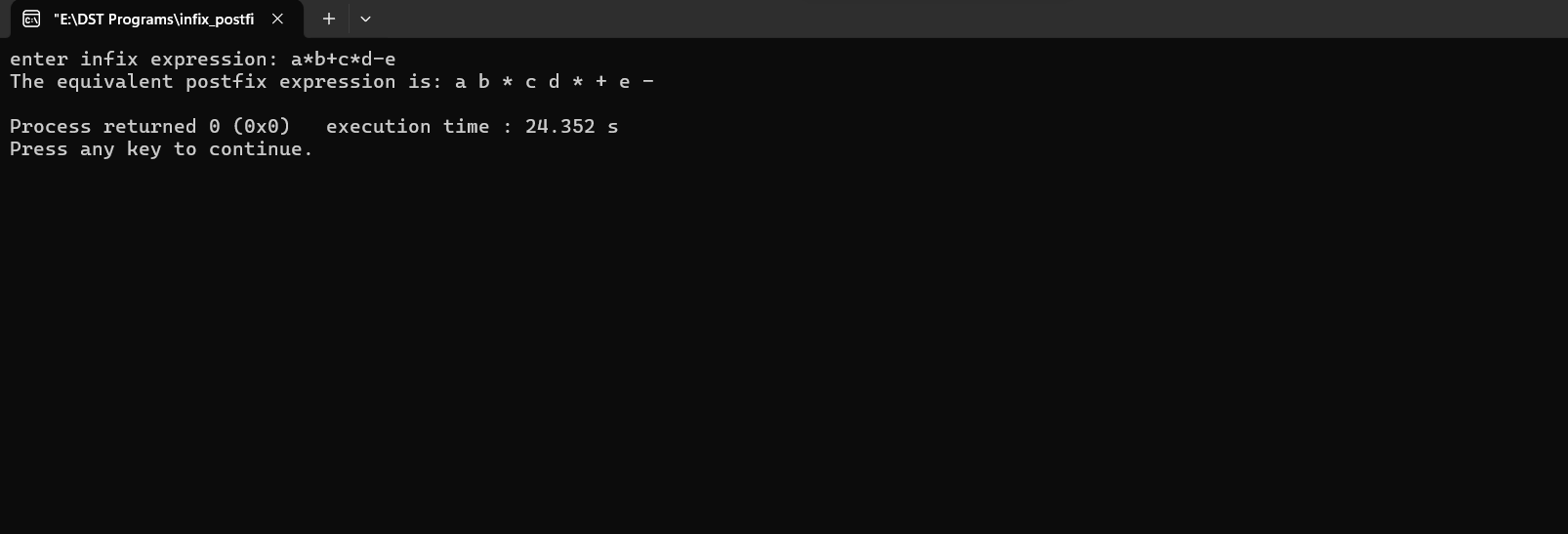
**if(top==-1)**

**return 1;**

**else**

**return 0;**

**}**

****

**Lab program 3:**

**a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display**

**The program should print appropriate messages for queue empty and queue overflow conditions**

**b ) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete &amp; Display**

**The program should print appropriate messages for queue empty and queue overflow conditions**

**a)**

**#include <stdio.h>**

**#define MAX 3**

**void insert();**

**void delete();**

**void display();**

**int queue\_array[MAX];**

**int rear = - 1;**

**int front = - 1;**

**main()**

**{**

**int choice;**

**while (1)**

**{**

**printf("1.Insert element to queue \n");**

**printf("2.Delete element from queue \n");**

**printf("3.Display all elements of queue \n");**

**printf("4.Quit \n");**

**printf("Enter your choice : ");**

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

**switch (choice)**

**{**

**case 1:**

**insert();**

**break;**

**case 2:**

**delete();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**exit(1);**

**default:**

**printf("Wrong choice \n");**

**} /\* End of switch \*/**

**} /\* End of while \*/**

**} /\* End of main() \*/**

**void insert()**

**{**

**int add\_item;**

**if (rear == MAX - 1)**

**printf("Queue Overflow \n");**

**else**

**{**

**if (front == - 1)**

**/\*If queue is initially empty \*/**

**front = 0;**

**printf("Inset the element in queue : ");**

**scanf("%d", &add\_item);**

**rear = rear + 1;**

**queue\_array[rear] = add\_item;**

**}**

**} /\* End of insert() \*/**

**void delete()**

**{**

**if (front == - 1 || front > rear)**

**{**

**printf("Queue Underflow \n");**

**return ;**

**}**

**else**

**{**

**printf("Element deleted from queue is : %d\n", queue\_array[front]);**

**front = front + 1;**

**}**

**} /\* End of delete() \*/**

**void display()**

**{**

**int i;**

**if (front == - 1)**

**printf("Queue is empty \n");**

**else**

**{**

**printf("Queue is : \n");**

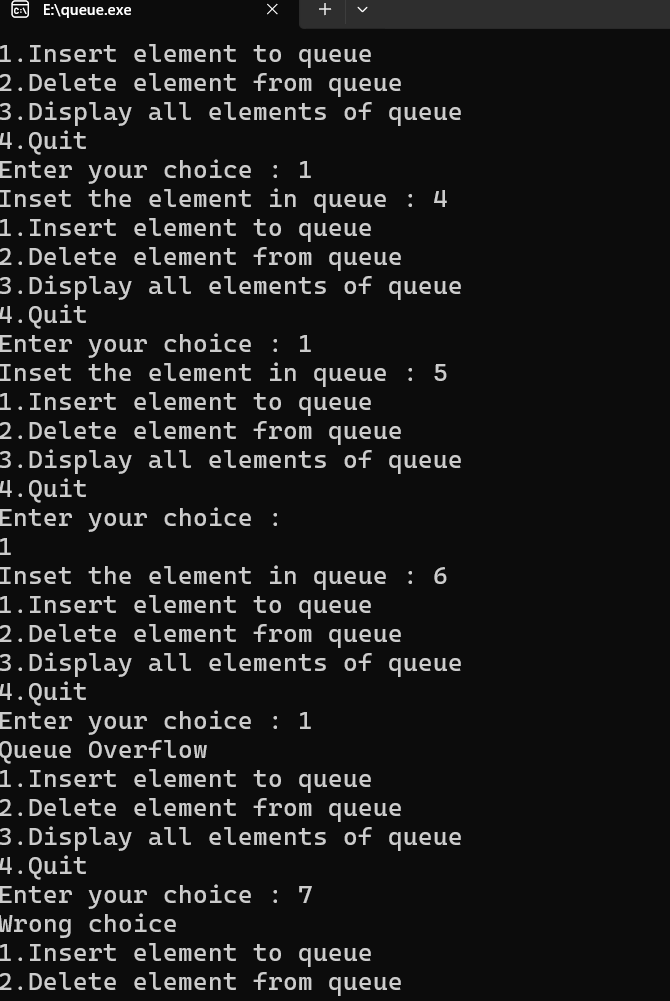
**for (i = front; i <= rear; i++)**

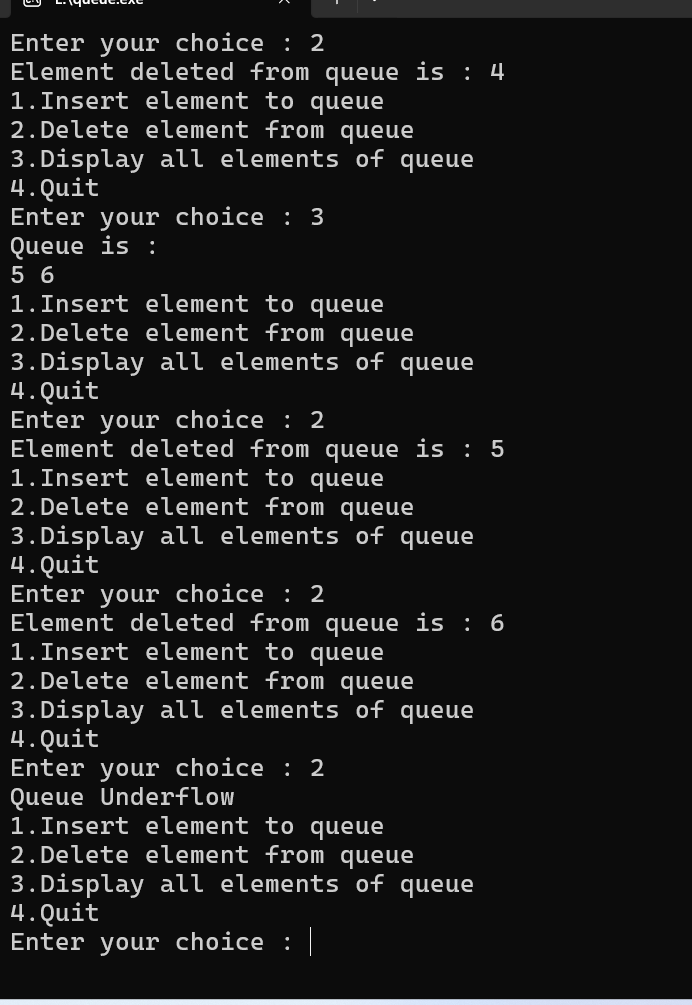
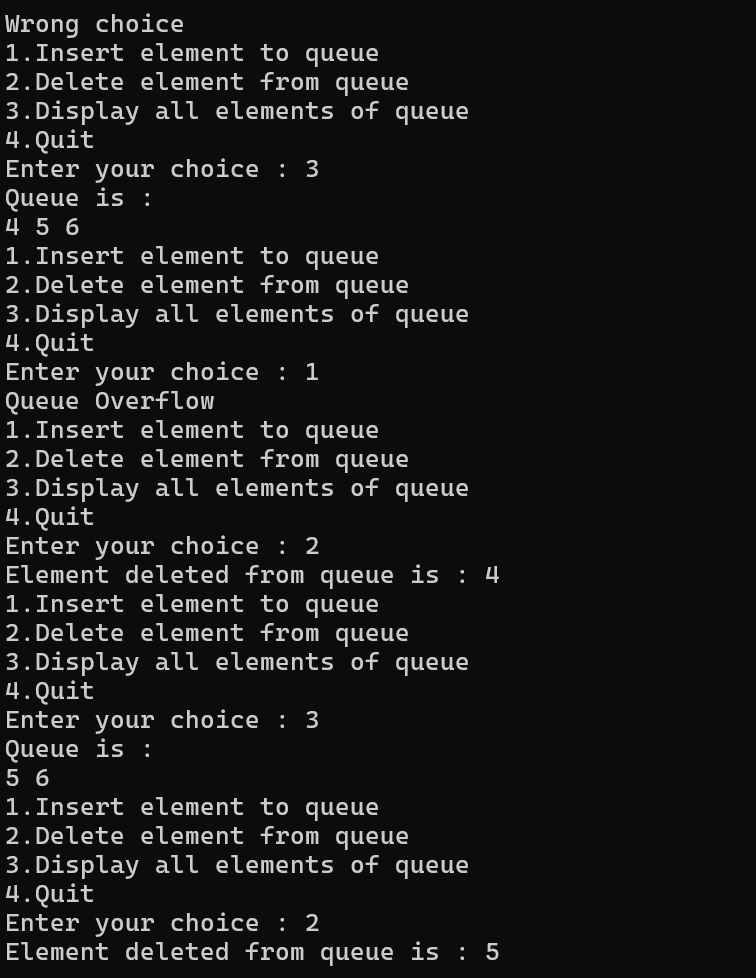
**printf("%d ", queue\_array[i]);**

**printf("\n");**

**}**

**} /\* End of display() \*/**

****

****

**b)**

**#include<stdio.h>**

**#include<stdlib.h>**

**#define Size 5**

**int Q[Size];**

**int rear=-1;**

**int front=-1;**

**int IsFull()**

**{**

**if(front==(rear+1)%Size)**

**{**

**return 0;**

**}**

**else{**

**return -1;**

**}**

**}**

**int IsEmpty()**

**{**

**if (front==-1 && rear==-1)**

**{**

**return 0;**

**}**

**else{**

**return -1;**

**}**

**}**

**void Enqueue(int x)**

**{**

**int item;**

**if(IsFull()==0)**

**{**

**printf("queue overflow\n");**

**return;**

**}**

**else{**

**if (IsEmpty()==0)**

**{**

**front=0;**

**rear=0;**

**}**

**else{**

**rear=(rear+1)%Size;**

**}**

**Q[rear]=x;**

**}**

**}**

**int Dequeue()**

**{**

**int x;**

**if(IsEmpty()==0)**

**{**

**printf("queue underflow \n");**

**}**

**else{**

**if(front==rear)**

**{**

**x=Q[front];**

**front=-1;**

**rear=-1;**

**}**

**else{**

**x=Q[front];**

**front=(front+1)%Size;**

**}**

**return x;**

**}**

**}**

**void Display()**

**{**

**int i;**

**if(IsEmpty()==0)**

**{**

**printf("queue id empty\n");**

**}**

**else{**

**printf("queue elements:\n");**

**for(i=front; i!=rear; i=(i+1)%Size)**

**{**

**printf("%d\n",Q[i]);**

**}**

**printf("%d\n",Q[i]);**

**}**

**}**

**void main()**

**{**

**int choice,x,b;**

**while(1)**

**{**

**printf("1.Enqueue,2.Dequeue,3.Display,4.Exit\n");**

**printf("enter your choice:\n");**

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

**switch(choice)**

**{**

**case 1:**

**printf("enter the number to be inserted into the queue\n");**

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

**Enqueue(x);**

**break;**

**case 2:**

**b=Dequeue();**

**printf("%d was removed from the queue\n",b);**

**break;**

**case 3:**

**Display();**

**break;**

**case 4:**

**exit(1);**

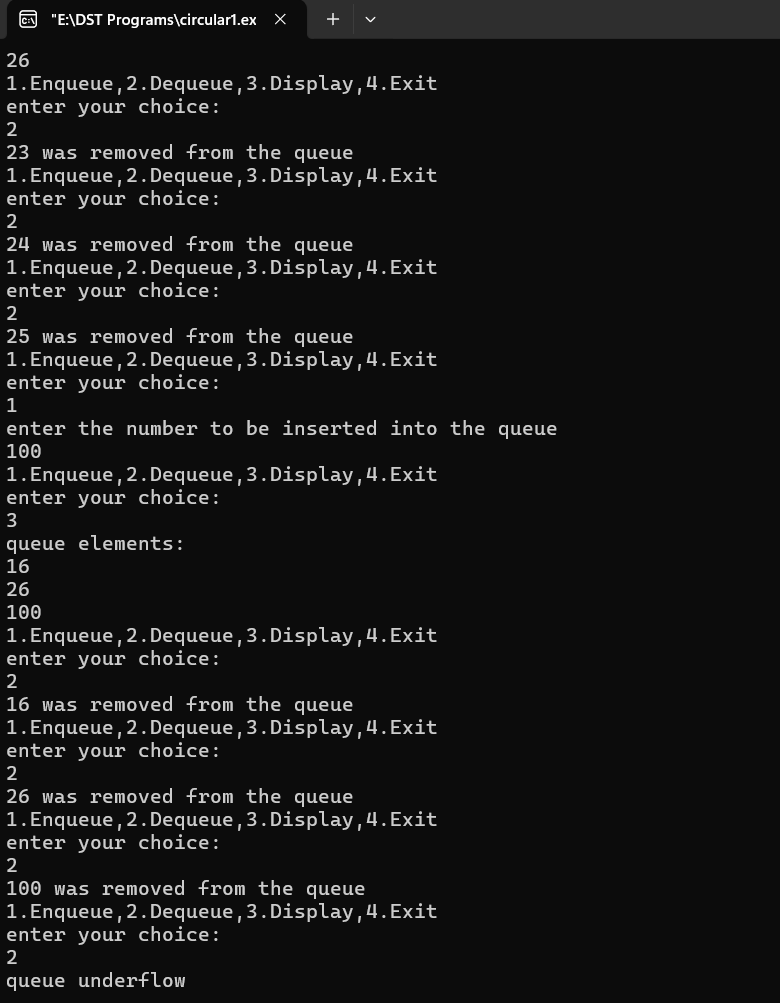
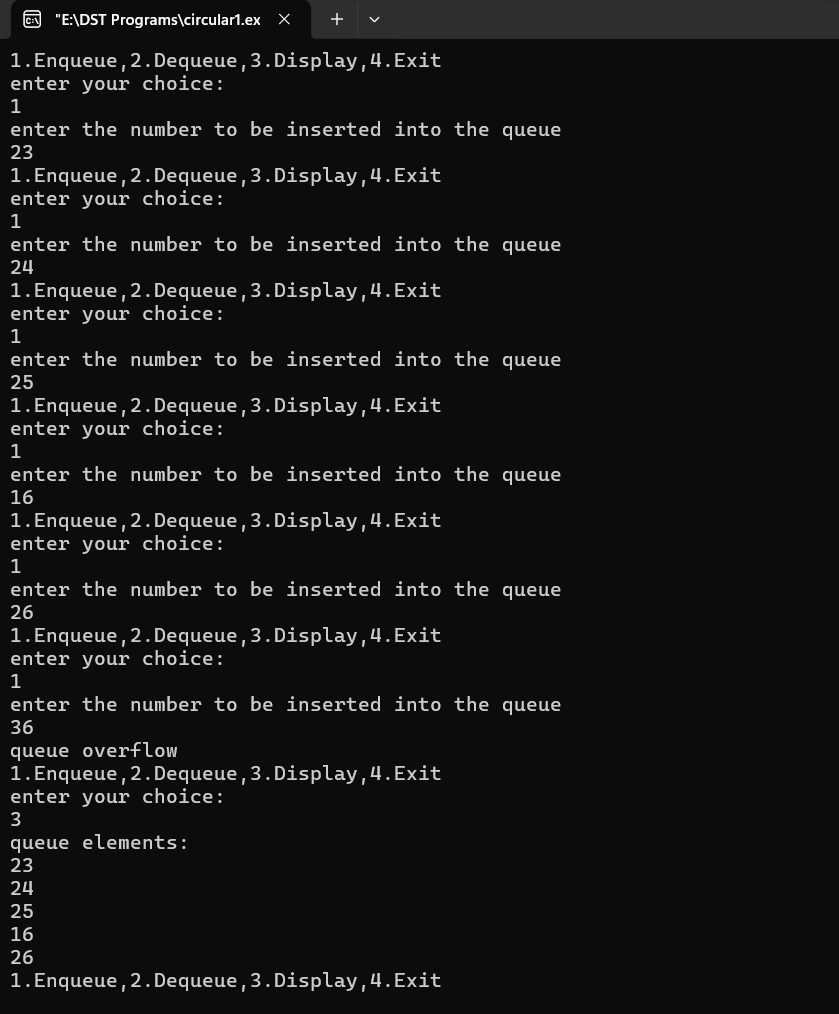
**default:**

**printf("invalid input\n");**

**}**

**}**

**}**

****

**Lab program 4**

**WAP to Implement Singly Linked List with following operations**

**a) Create a linked list.**

**b) Insertion of a node at first position, at any position and at end of list. Display the contents of the linked list.**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct node{**

**int data;**

**struct node\*next;**

**};**

**void display();**

**void insert\_begin();**

**void insert\_end();**

**void insert\_pos();**

**struct node \*head=NULL;**

**void display()**

**{**

**printf("elements are :\n");**

**struct node \*ptr;**

**if(head==NULL)**

**{**

**printf("list is empty");**

**return;**

**}**

**else{**

**ptr=head;**

**while(ptr !=NULL)**

**{**

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

**ptr=ptr->next;**

**}**

**}**

**}**

**void insert\_begin()**

**{**

**struct node\*temp;**

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

**printf("enter the value to be inserted\n");**

**scanf("%d",&temp->data);**

**temp->next=NULL;**

**if(head==NULL)**

**head=temp;**

**else{**

**temp->next=head;**

**head=temp;**

**}**

**}**

**void insert\_end()**

**{**

**struct node \*temp,\*ptr;**

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

**printf("enter the value to be inserted \n");**

**scanf("%d",&temp->data);**

**temp->next=NULL;**

**if(head==NULL)**

**{**

**head=temp;**

**}**

**else**

**{**

**ptr=head;**

**while(ptr->next != NULL)**

**{**

**ptr=ptr->next;**

**}**

**ptr->next=temp;**

**}**

**}**

**void insert\_pos()**

**{**

**int pos,i;**

**struct node\*temp,\*ptr;**

**printf("enter the position");**

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

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

**printf("enter the value to be inserted\n");**

**scanf("%d",&temp->data);**

**temp->next=NULL;**

**if(pos==0)**

**{**

**temp->next=head;**

**head=temp;**

**}**

**else**

**{**

**for(i=0, ptr=head; i<pos-1;i++)**

**{**

**ptr=ptr->next;**

**}**

**temp->next=ptr->next;**

**ptr->next=temp;**

**}**

**}**

**void main()**

**{**

**int choice;**

**while(1)**

**{**

**printf("\n 1.to insert at the beginning\n"**

**" 2.to insert at the end\n "**

**"3.to insert at the position\n "**

**"4.to display\n "**

**"5.exit\n");**

**printf("enter you choice:\n");**

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

**switch(choice)**

**{**

**case 1:**

**insert\_begin();**

**break;**

**case 2:**

**insert\_end();**

**break;**

**case 3:**

**insert\_pos();**

**break;**

**case 4:**

**display();**

**break;**

**case 5:**

**exit(0);**

**break;**

**default:**

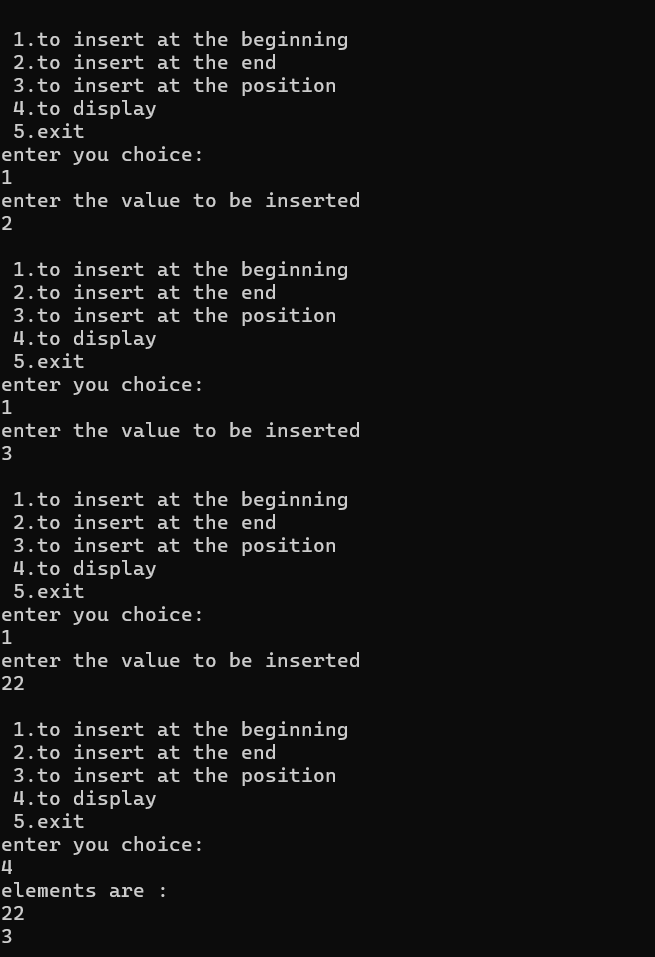
**printf("invalid choice\n");**

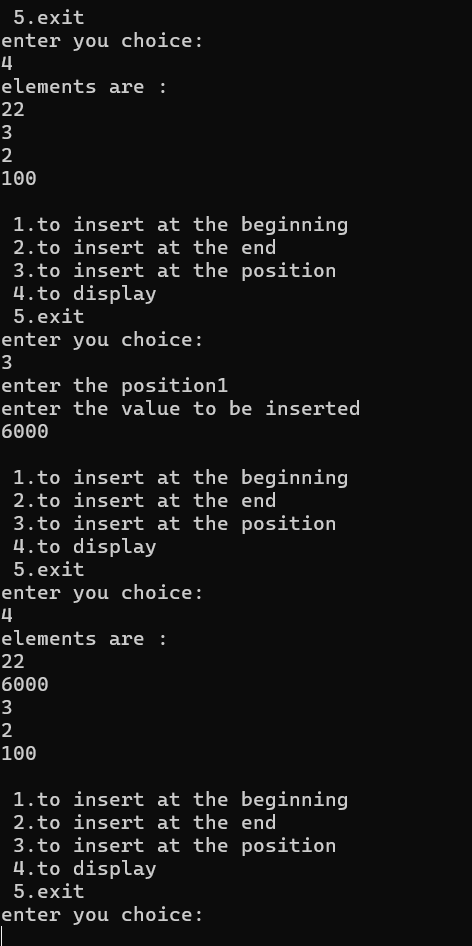
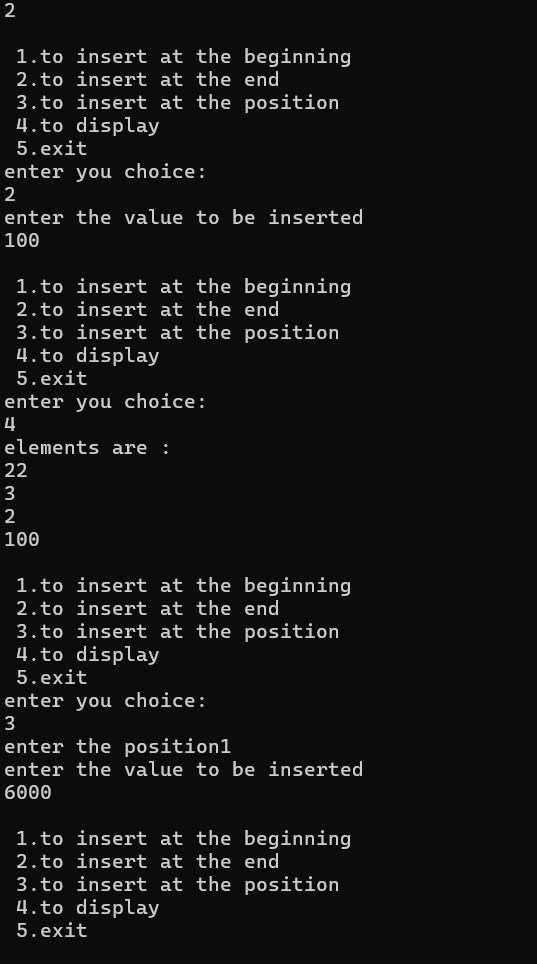
**break;**

**}**

**}**

**}**

****

****

**LEETCODE 1:**

Demonstration of stack program

typedef struct {

int \*stack;

int \*minStack;

int top

} MinStack;

MinStack\* minStackCreate() {

MinStack\* stack = (MinStack\*)malloc(sizeof(MinStack));

stack->stack = (int\*)malloc(sizeof(int) \* 10000);

stack->minStack = (int\*)malloc(sizeof(int) \* 10000);

stack->top = -1;

return stack;

}

void minStackPush(MinStack\* obj, int val) {

obj->top++;

obj->stack[obj->top] = val;

if (obj->top == 0 || val <= obj->minStack[obj->top - 1]) {

obj->minStack[obj->top] = val;

} else {

obj->minStack[obj->top] = obj->minStack[obj->top - 1];

}

}

void minStackPop(MinStack\* obj) {

obj->top--;

}

int minStackTop(MinStack\* obj) {

return obj->stack[obj->top];

}

int minStackGetMin(MinStack\* obj) {

return obj->minStack[obj->top];

}

void minStackFree(MinStack\* obj) {

free(obj->stack);

free(obj->minStack);

free(obj);

}

****

**Lab program 5**

**WAP to Implement Singly Linked List with following operations**

**a) Create a linked list.**

**b) Deletion of first element, specified element and last element in the list.**

**c) Display the contents of the linked list.**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct node{**

**int data;**

**struct node\*next;**

**};**

**void display();**

**void insert\_begin();**

**void insert\_end();**

**void insert\_pos();**

**void begin\_delete();**

**struct node \*head=NULL;**

**void display()**

**{**

**printf("elements are :\n");**

**struct node \*ptr;**

**if(head==NULL)**

**{**

**printf("list is empty");**

**return;**

**}**

**else{**

**ptr=head;**

**while(ptr !=NULL)**

**{**

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

**ptr=ptr->next;**

**}**

**}**

**}**

**void insert\_begin()**

**{**

**struct node\*temp;**

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

**printf("enter the value to be inserted\n");**

**scanf("%d",&temp->data);**

**temp->next=NULL;**

**if(head==NULL)**

**head=temp;**

**else{**

**temp->next=head;**

**head=temp;**

**}**

**}**

**void insert\_end()**

**{**

**struct node \*temp,\*ptr;**

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

**printf("enter the value to be inserted \n");**

**scanf("%d",&temp->data);**

**temp->next=NULL;**

**if(head==NULL)**

**{**

**head=temp;**

**}**

**else**

**{**

**ptr=head;**

**while(ptr->next != NULL)**

**{**

**ptr=ptr->next;**

**}**

**ptr->next=temp;**

**}**

**}**

**void insert\_pos()**

**{**

**int pos,i;**

**struct node\*temp,\*ptr;**

**printf("enter the position");**

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

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

**printf("enter the value to be inserted\n");**

**scanf("%d",&temp->data);**

**temp->next=NULL;**

**if(pos==0)**

**{**

**temp->next=head;**

**head=temp;**

**}**

**else**

**{**

**for(i=0, ptr=head; i<pos-1;i++)**

**{**

**ptr=ptr->next;**

**}**

**temp->next=ptr->next;**

**ptr->next=temp;**

**}**

**}**

**void begin\_delete()**

**{**

**struct node \*ptr;**

**if(head == NULL)**

**{**

**printf("\nList is empty\n");**

**}**

**else**

**{**

**ptr = head;**

**head = ptr->next;**

**free(ptr);**

**printf("\nNode deleted from the begining ...\n");**

**}**

**}**

**void last\_delete()**

**{**

**struct node \*ptr,\*ptr1;**

**if(head == NULL)**

**{**

**printf("\nlist is empty");**

**}**

**else if(head -> next == NULL)**

**{**

**head = NULL;**

**free(head);**

**printf("\nOnly node of the list deleted ...\n");**

**}**

**else**

**{**

**ptr = head;**

**while(ptr->next != NULL)**

**{**

**ptr1 = ptr;**

**ptr = ptr ->next;**

**}**

**ptr1->next = NULL;**

**free(ptr);**

**printf("\nDeleted Node from the last ...\n");**

**}**

**}**

**void random\_delete()**

**{**

**struct node \*ptr,\*ptr1;**

**int loc,i;**

**printf("\n Enter the location of the node after which you want to perform deletion \n");**

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

**ptr=head;**

**for(i=0;i<loc;i++)**

**{**

**ptr1 = ptr;**

**ptr = ptr->next;**

**if(ptr == NULL)**

**{**

**printf("\nCan't delete");**

**return;**

**}**

**}**

**ptr1 ->next = ptr ->next;**

**free(ptr);**

**printf("\nDeleted node %d ",loc+1);**

**}**

**void main()**

**{**

**int choice;**

**while(1)**

**{**

**printf("\n 1.to insert at the beginning\n"**

**" 2.to insert at the end\n "**

**"3.to insert at the position\n "**

**"4.to display\n "**

**"5.delete from beginning\n"**

**"6.delete from end\n"**

**"7.random delete\n"**

**"8.exit\n");**

**printf("enter you choice:\n");**

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

**switch(choice)**

**{**

**case 1:**

**insert\_begin();**

**break;**

**case 2:**

**insert\_end();**

**break;**

**case 3:**

**insert\_pos();**

**break;**

**case 4:**

**display();**

**break;**

**case 5:**

**begin\_delete();**

**break;**

**case 6:**

**last\_delete();**

**break;**

**case 7:**

**random\_delete();**

**break;**

**case 8:**

**exit(0);**

**break;**

**default:**

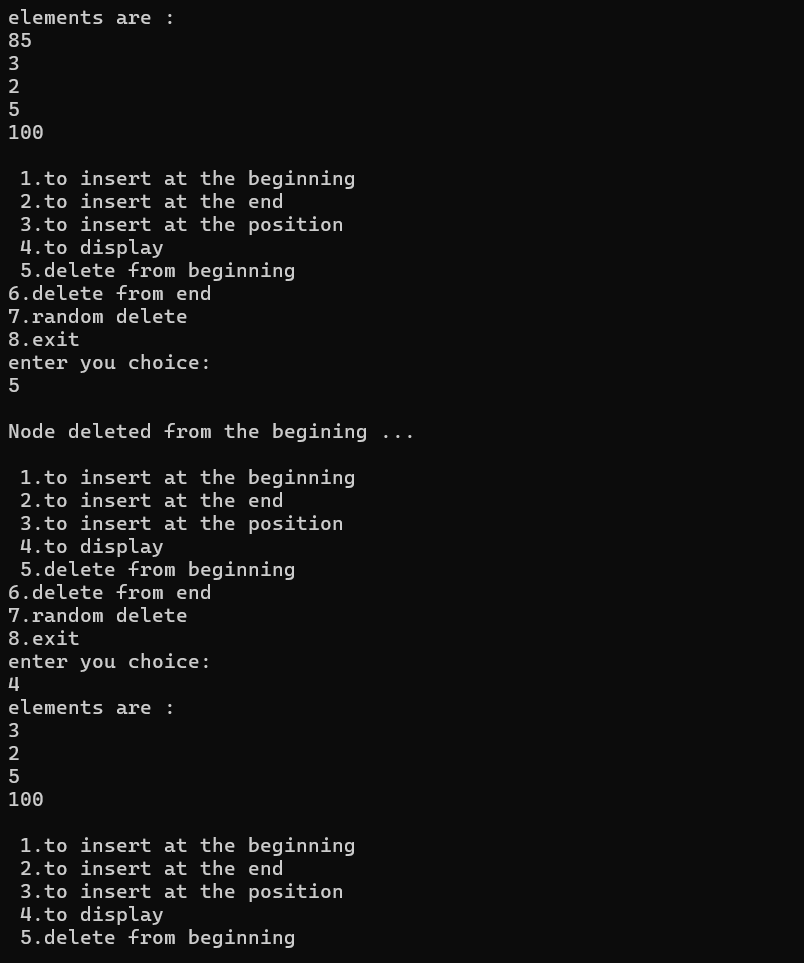
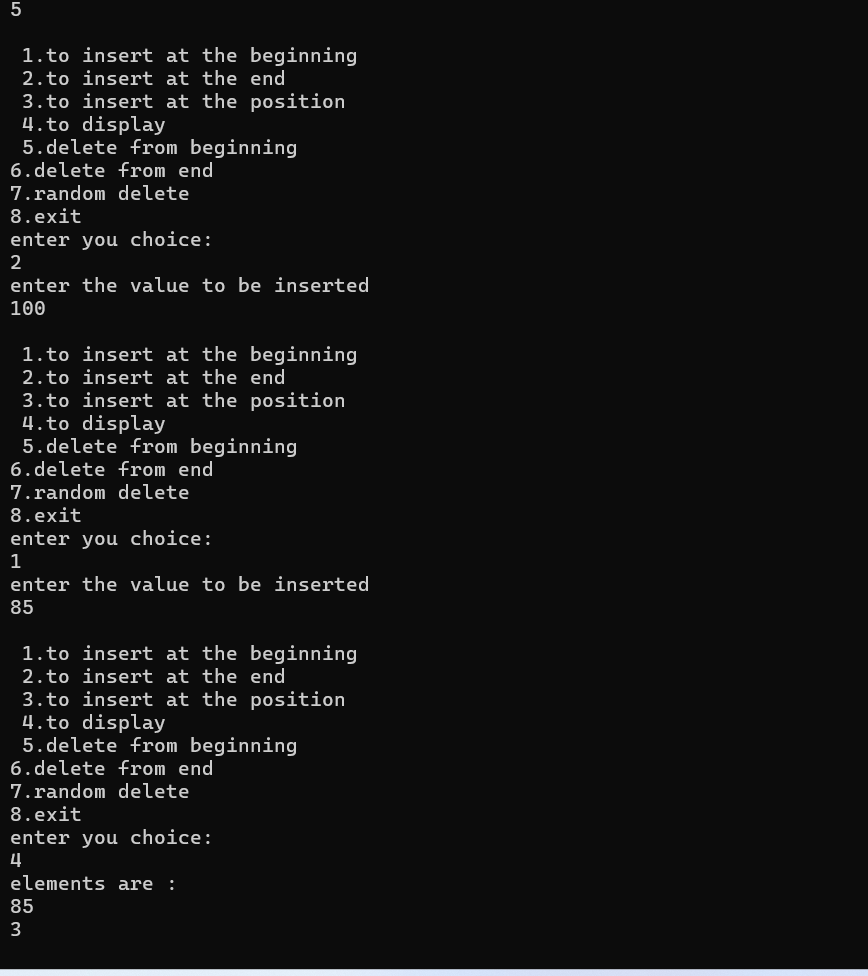
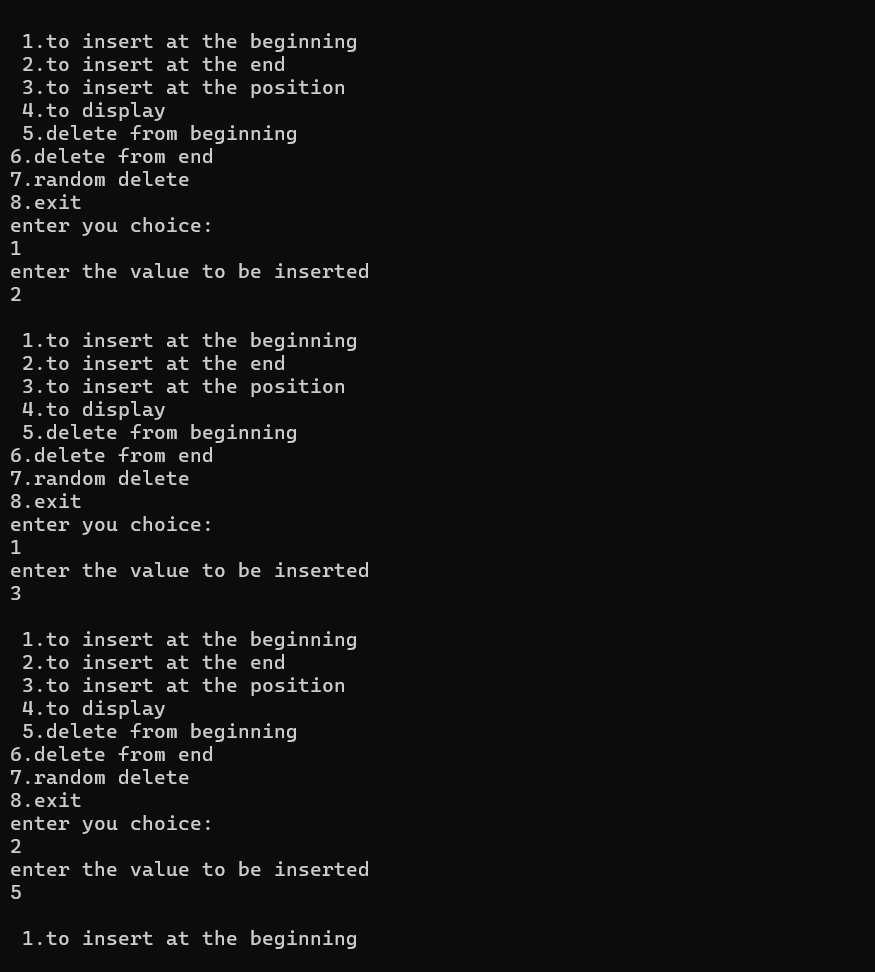
**printf("invalid choice\n");**

**break;**

**}**

**}**

**}**

****

**LEETCODE 2:**

 Demonstration of LeetCode program on Singly linked list

struct ListNode\* reverseBetween(struct ListNode\* head, int left, int right) {

    if (head == NULL|| left == right)

    {

        return head;

    }

    struct ListNode \*dummy = (struct ListNode\*)malloc(sizeof(struct ListNode));

    dummy->next=head;

    struct ListNode \*prev=dummy;

    for(int i=1; i<left; i++)

    {

        prev=prev->next;

    }

    struct ListNode\* current=prev->next;

    struct ListNode\* next=NULL;

    struct ListNode\* tail=current;

    for(int i=left; i<=right; i++){

        struct ListNode\* temp=current->next;

        current->next=next;

        next=current;

        current=temp;

    }

    prev->next=next;

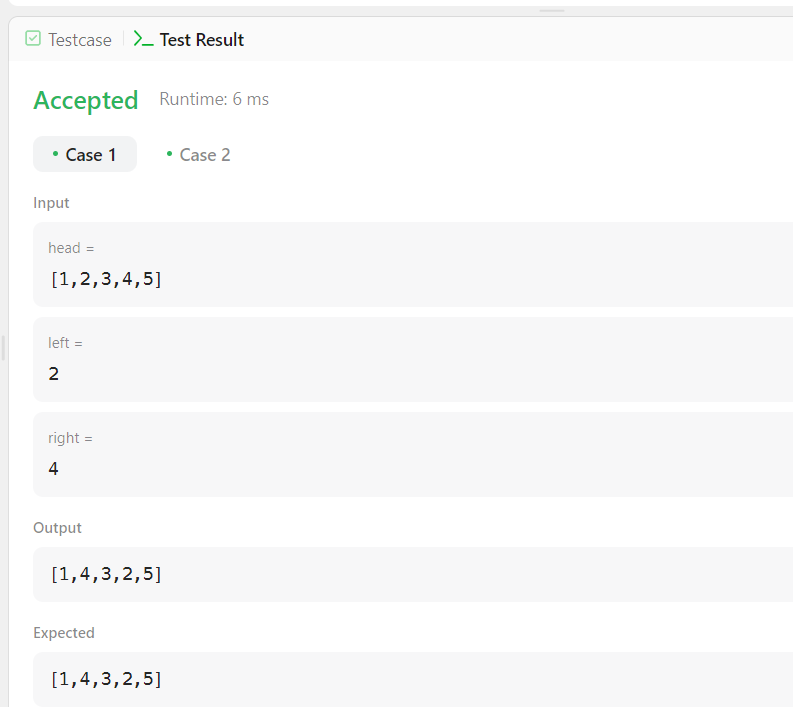
    tail->next=current;

    struct ListNode\* result=dummy->next;

    free(dummy);

    return result;

}

****

**Lab program 6**

**a) WAP to Implement Single Link List with following operations: Sort**

**the linked list, Reverse the linked list, Concatenation of two linked lists.**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node\* next;**

**};**

**void append(struct Node\*\* head\_ref, int new\_data) {**

**struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));**

**struct Node\* last = \*head\_ref;**

**new\_node->data = new\_data;**

**new\_node->next = NULL;**

**if (\*head\_ref == NULL) {**

**\*head\_ref = new\_node;**

**return;**

**}**

**while (last->next != NULL) {**

**last = last->next;**

**}**

**last->next = new\_node;**

**}**

**void printList(struct Node\* node) {**

**while (node != NULL) {**

**printf("%d -> ", node->data);**

**node = node->next;**

**}**

**printf("NULL\n");**

**}**

**void sortList(struct Node\*\* head\_ref) {**

**if (\*head\_ref == NULL) {**

**return;**

**}**

**int swapped, temp;**

**struct Node\* ptr1;**

**struct Node\* lptr = NULL;**

**do {**

**swapped = 0;**

**ptr1 = \*head\_ref;**

**while (ptr1->next != lptr) {**

**if (ptr1->data > ptr1->next->data) {**

**temp = ptr1->data;**

**ptr1->data = ptr1->next->data;**

**ptr1->next->data = temp;**

**swapped = 1;**

**}**

**ptr1 = ptr1->next;**

**}**

**lptr = ptr1;**

**} while (swapped);**

**}**

**void reverseList(struct Node\*\* head\_ref) {**

**struct Node\* prev = NULL;**

**struct Node\* current = \*head\_ref;**

**struct Node\* next = NULL;**

**while (current != NULL) {**

**next = current->next;**

**current->next = prev;**

**prev = current;**

**current = next;**

**}**

**\*head\_ref = prev;**

**}**

**void concatenateLists(struct Node\*\* head1, struct Node\* head2) {**

**if (\*head1 == NULL) {**

**\*head1 = head2;**

**return;**

**}**

**struct Node\* temp = \*head1;**

**while (temp->next != NULL) {**

**temp = temp->next;**

**}**

**temp->next = head2;**

**}**

**int main() {**

**struct Node\* list1 = NULL;**

**struct Node\* list2 = NULL;**

**int n, data;**

**printf("Enter the number of elements for List 1: ");**

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

**printf("Enter the elements for List 1:\n");**

**for (int i = 0; i < n; ++i) {**

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

**append(&list1, data);**

**}**

**printf("Enter the number of elements for List 2: ");**

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

**printf("Enter the elements for List 2:\n");**

**for (int i = 0; i < n; ++i) {**

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

**append(&list2, data);**

**}**

**printf("\nOriginal List 1: ");**

**printList(list1);**

**printf("Original List 2: ");**

**printList(list2);**

**sortList(&list1);**

**sortList(&list2);**

**printf("\nSorted List 1: ");**

**printList(list1);**

**printf("Sorted List 2: ");**

**printList(list2);**

**concatenateLists(&list1, list2);**

**printf("\nConcatenated List: ");**

**printList(list1);**

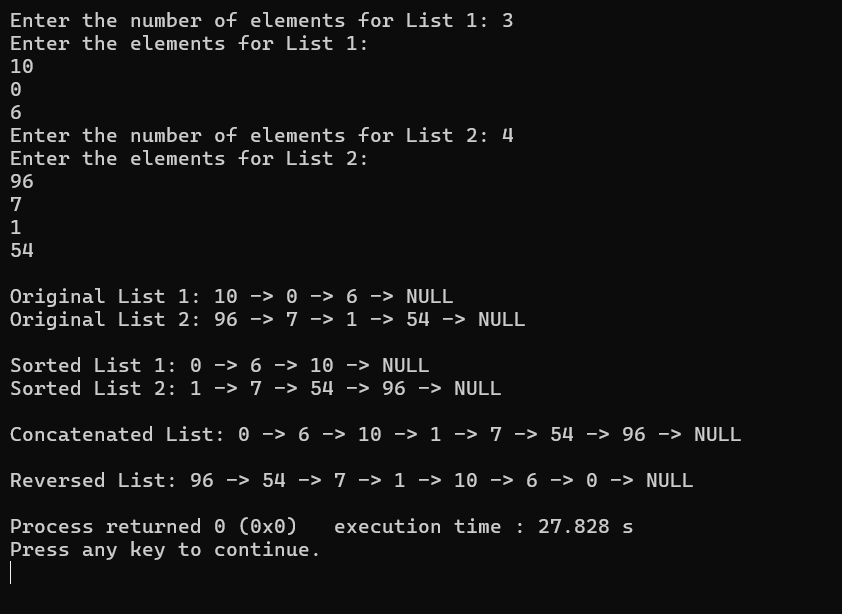
**reverseList(&list1);**

**printf("\nReversed List: ");**

**printList(list1);**

**return 0;**

**}**

****

**b) WAP to Implement Single Link List to simulate Stack &amp; Queue**

**Operations.**

**Stack using linkedlist:**

**#include <stdio.h>**

**#include <stdlib.h>**

**void push();**

**void pop();**

**void display();**

**struct node**

**{**

**int val;**

**struct node \*next;**

**};**

**struct node \*head;**

**void main ()**

**{**

**int choice=0;**

**printf("\nStack operations using linked list\n");**

**while(choice != 4)**

**{**

**printf("\n\nChose one from the below options...\n");**

**printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");**

**printf("\n Enter your choice \n");**

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

**switch(choice)**

**{**

**case 1:**

**{**

**push();**

**break;**

**}**

**case 2:**

**{**

**pop();**

**break;**

**}**

**case 3:**

**{**

**display();**

**break;**

**}**

**case 4:**

**{**

**printf("Exiting....");**

**break;**

**}**

**default:**

**{**

**printf("Please Enter valid choice ");**

**}**

**};**

**}**

**}**

**void push ()**

**{**

**int val;**

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

**if(ptr == NULL)**

**{**

**printf("not able to push the element");**

**}**

**else**

**{**

**printf("Enter the value");**

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

**if(head==NULL)**

**{**

**ptr->val = val;**

**ptr -> next = NULL;**

**head=ptr;**

**}**

**else**

**{**

**ptr->val = val;**

**ptr->next = head;**

**head=ptr;**

**}**

**printf("Item pushed");**

**}**

**}**

**void pop()**

**{**

**int item;**

**struct node \*ptr;**

**if (head == NULL)**

**{**

**printf("Underflow");**

**}**

**else**

**{**

**item = head->val;**

**ptr = head;**

**head = head->next;**

**free(ptr);**

**printf("Item popped");**

**}**

**}**

**void display()**

**{**

**int i;**

**struct node \*ptr;**

**ptr=head;**

**if(ptr == NULL)**

**{**

**printf("Stack is empty\n");**

**}**

**else**

**{**

**printf("Printing Stack elements \n");**

**while(ptr!=NULL)**

**{**

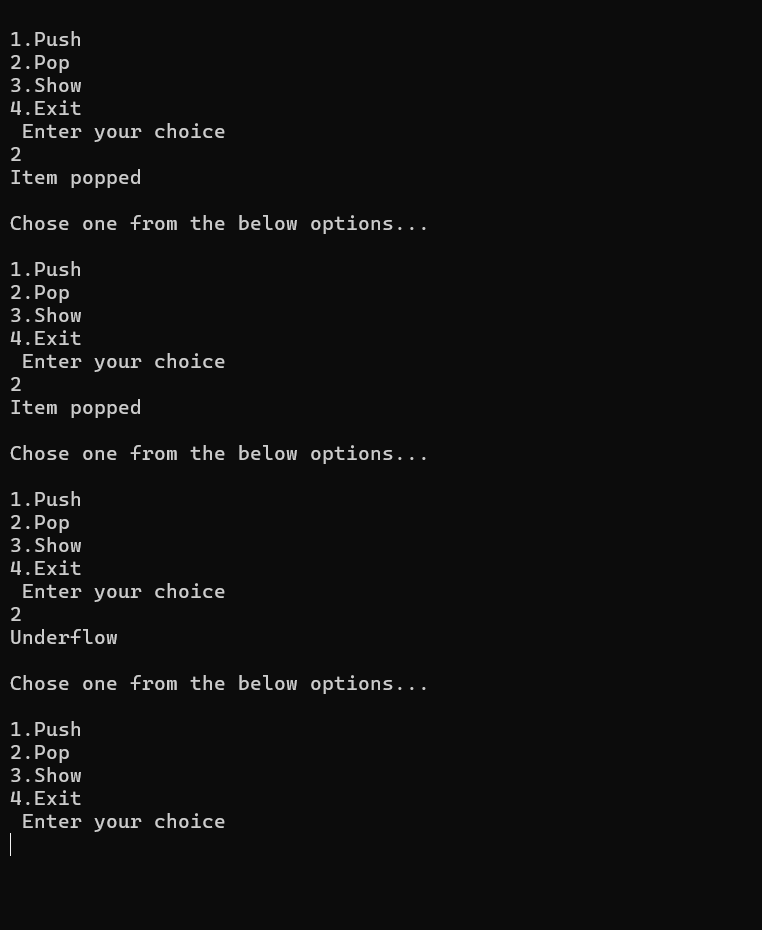
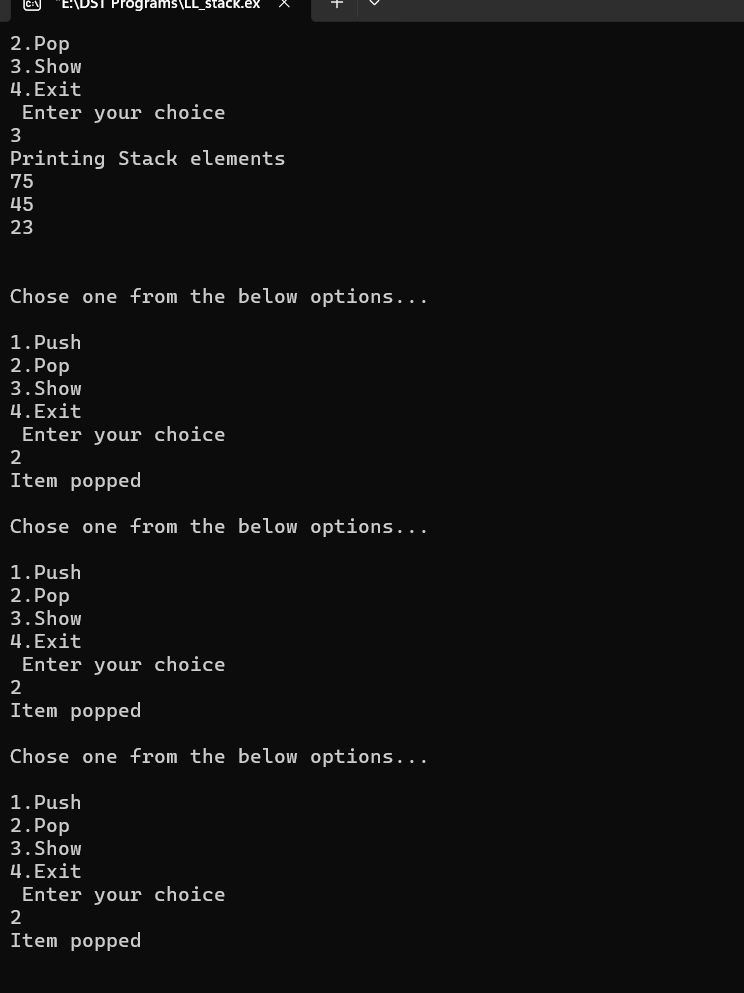
**printf("%d\n",ptr->val);**

**ptr = ptr->next;**

**}**

**}**

**}**

****

**Queue using linkedlist:**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct node**

**{**

**int data;**

**struct node \*next;**

**};**

**struct node \*front;**

**struct node \*rear;**

**void insert();**

**void delete();**

**void display();**

**void main ()**

**{**

**int choice;**

**while(choice != 4)**

**{**

**printf("\nQueue operation using linked list\n");**

**printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");**

**printf("\nEnter your choice ");**

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

**switch(choice)**

**{**

**case 1:**

**insert();**

**break;**

**case 2:**

**delete();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**exit(0);**

**break;**

**default:**

**printf("\nEnter valid choice??\n");**

**}**

**}**

**}**

**void insert()**

**{**

**struct node \*ptr;**

**int item;**

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

**if(ptr == NULL)**

**{**

**printf("\nOVERFLOW\n");**

**return;**

**}**

**else**

**{**

**printf("\nEnter value?\n");**

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

**ptr -> data = item;**

**if(front == NULL)**

**{**

**front = ptr;**

**rear = ptr;**

**front -> next = NULL;**

**rear -> next = NULL;**

**}**

**else**

**{**

**rear -> next = ptr;**

**rear = ptr;**

**rear->next = NULL;**

**}**

**}**

**}**

**void delete ()**

**{**

**struct node \*ptr;**

**if(front == NULL)**

**{**

**printf("\nUNDERFLOW\n");**

**return;**

**}**

**else**

**{**

**ptr = front;**

**front = front -> next;**

**free(ptr);**

**}**

**}**

**void display()**

**{**

**struct node \*ptr;**

**ptr = front;**

**if(front == NULL)**

**{**

**printf("\nEmpty queue\n");**

**}**

**else**

**{ printf("\nprinting values .....\n");**

**while(ptr != NULL)**

**{**

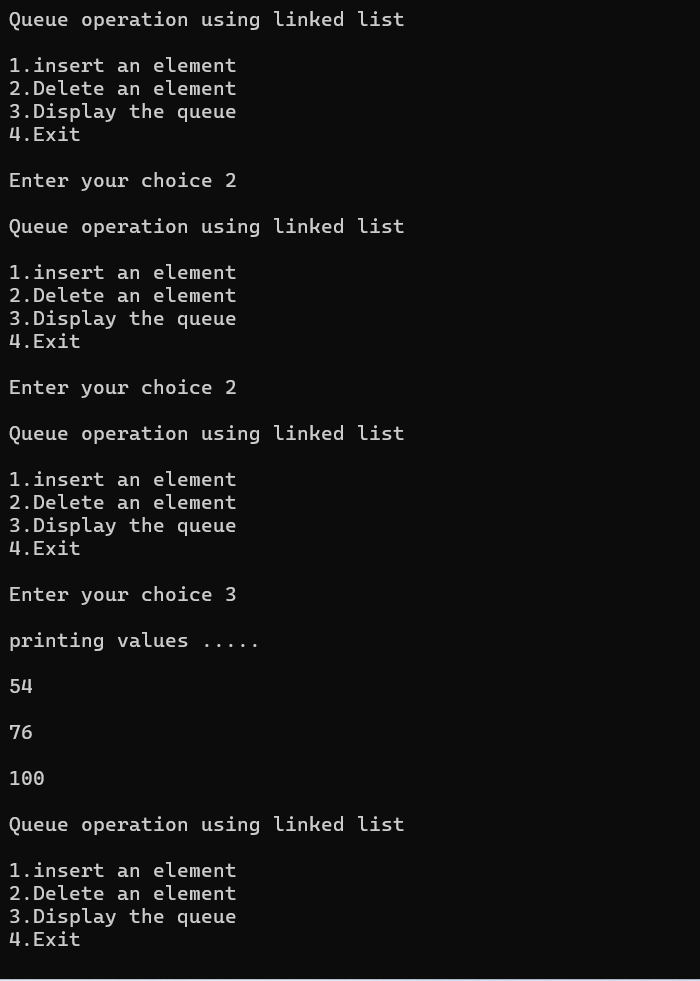
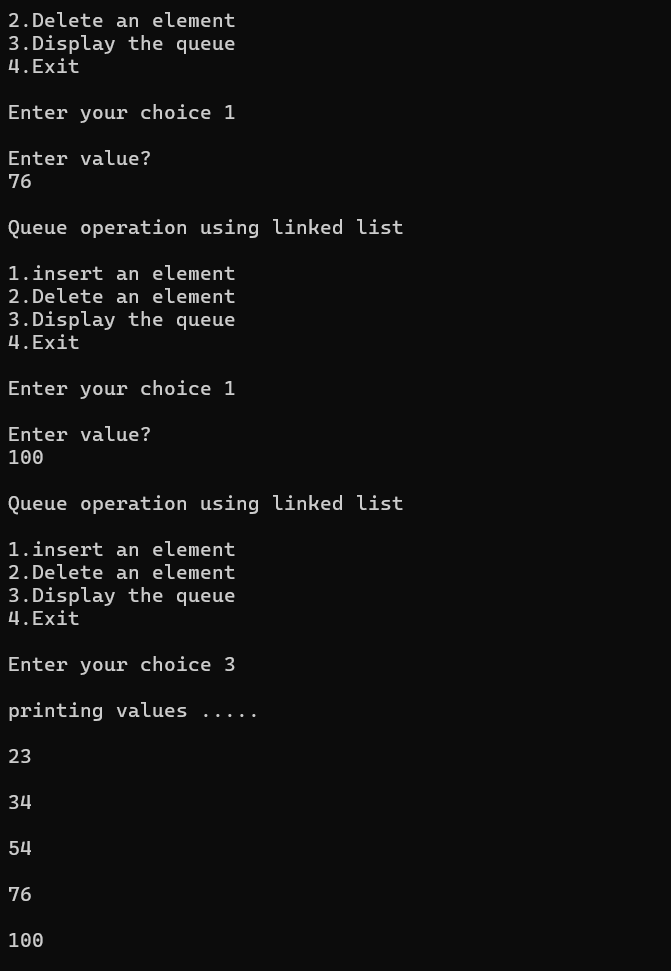
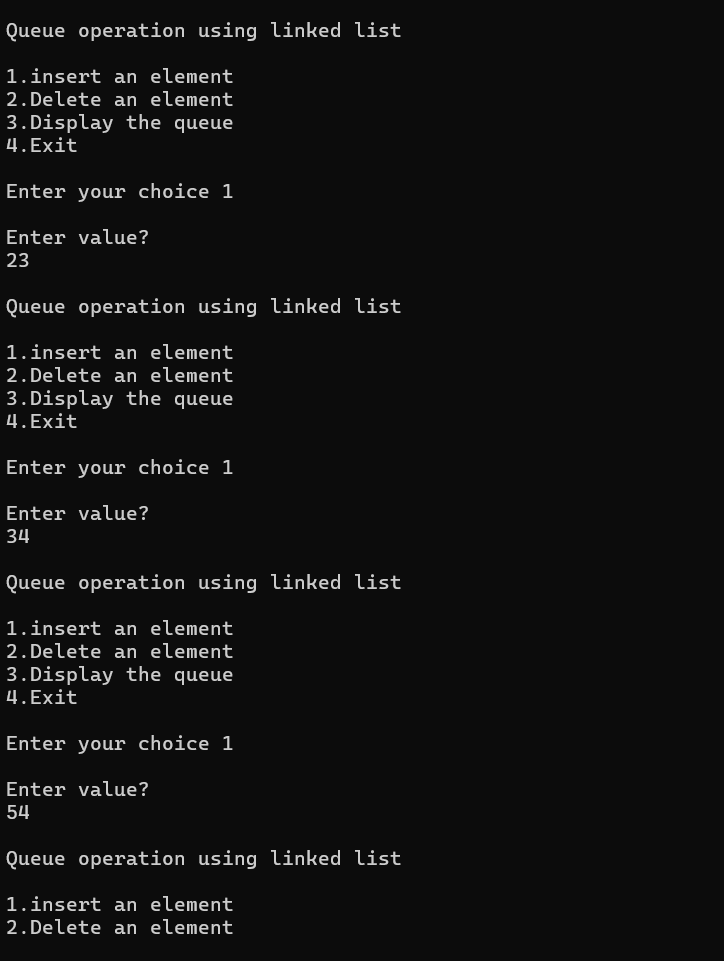
**printf("\n%d\n",ptr -> data);**

**ptr = ptr -> next;**

**}**

**}**

**}**

****

**Lab program 7**

**WAP to Implement doubly link list with primitive operations**

**a) Create a doubly linked list.**

**b) Insert a new node to the left of the node.**

**c) Delete the node based on a specific value**

**d) Display the contents of the list**

#include <stdio.h>

#include <stdlib.h>

struct node{

int data;

struct node \*prev;

struct node \*next;

}\*first=NULL;

void create(int a[],int n){

struct node \*t,\*last;

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

first->data=a[0];

first->prev=first->next=NULL;

last=first;

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

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

t->data=a[i];

t->next=last->next;

t->prev=last;

last->next=t;

last=t;

}

}

void insertleft(int val,intpos){

struct node \*t,\*ptr;

int i,loc;

loc=pos;

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

if(t==NULL){

printf("overflow");

}

t->data=val;

if(pos==1){

t->prev=NULL;

t->next=first;

if(first!=NULL){

first->prev=t;

}

first=t;

}

else

{

ptr=first;

for(int i=0;i<loc-2;i++){

ptr=ptr->next;

}

t->next=ptr->next;

t->prev=ptr;

if(ptr!=NULL){

ptr->next->prev=t;

}

ptr->next=t;

}

printf("node inserted ");

}

void delevalue(int val){

struct node \*ptr;

int value;

value=val;

ptr=first;

while(ptr!=NULL){

if(ptr->data==value){

if(ptr->prev!=NULL){

ptr->prev->next=ptr->next;

}

if(ptr->next!=NULL){

ptr->next->prev=ptr->prev;

}

if(ptr==first){

first=ptr->next;

}

free(ptr);

printf("value %d deleted",value);

return;

}

ptr=ptr->next;

}

printf("%d value not found",value);

}

void display(struct node \*p)

{

while(p!=NULL){

printf("%d\t",p->data);

p=p->next;

}

printf("\n");

}

void main(){

int a[10],n;

int val,key,loc;

printf("read n");

scanf("%d",&n);

printf("enter the values:");

for(int i=0;i<n;i++){

scanf("%d",&a[i]);

}

create(a,n);

display(first);

printf("enter the value to be inserted:");

scanf("%d",&val);

printf("enter the loc to be inserted at:");

scanf("%d",&loc);

insertleft(val,loc);

display(first);

printf("enter the key element to be deleted");

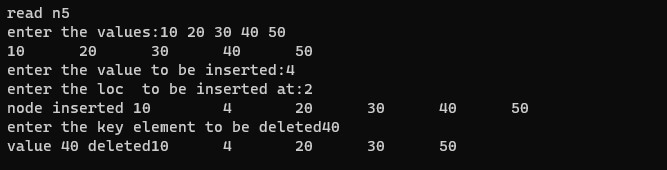
scanf("%d",&key);

delevalue(key);

display(first);

}

**Output:**



**LEETCODE 3:**

Demonstration of LeetCode program on Singly linked list

struct ListNode\*\* splitListToParts(struct ListNode\* head, int k, int\* returnSize) {

    // Calculate the length of the linked list

    int length = 0;

    struct ListNode\* current = head;

    while (current != NULL) {

        length++;

        current = current->next;

    }

    // Calculate the size of each part and the number of nodes that will have an extra node

    int partSize = length / k;

    int extraNodes = length % k;

    // Initialize the array to store the heads of the parts

    struct ListNode\*\* result = (struct ListNode\*\*)malloc(k \* sizeof(struct ListNode\*));

    // Split the linked list into parts

    current = head;

    for (int i = 0; i < k; i++) {

        // Determine the size of the current part

        int currentPartSize = partSize + (i < extraNodes ? 1 : 0);

        // Store the head of the current part in the result array

        result[i] = current;

        // Move to the end of the current part

        for (int j = 0; j < currentPartSize - 1 && current != NULL; j++) {

            current = current->next;

        }

        // If there are more nodes, break the link between parts

        if (current != NULL) {

            struct ListNode\* nextNode = current->next;

            current->next = NULL;

            current = nextNode;

        }

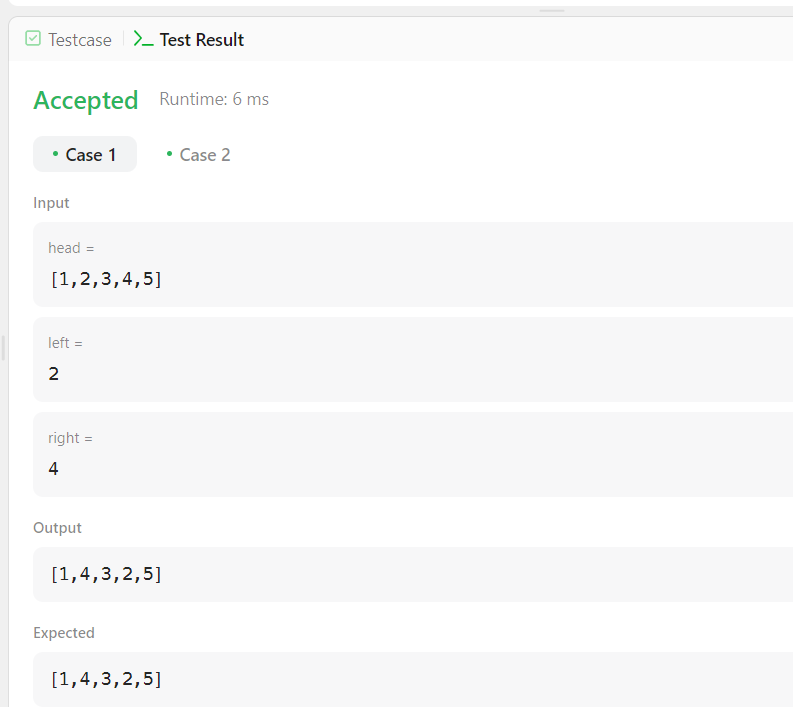
    }

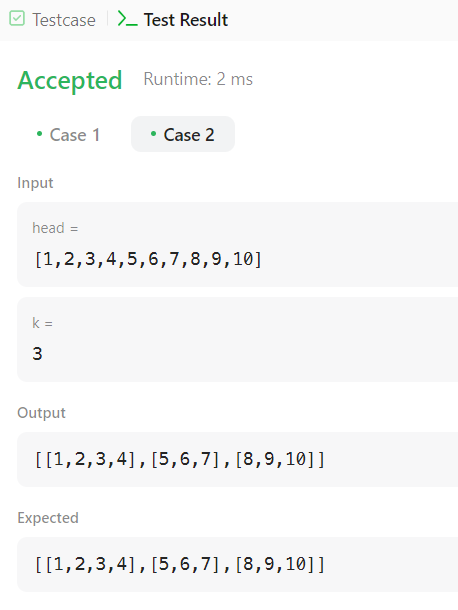
    // Set the returnSize

    \*returnSize = k;

    return result;

}

****

****

**Lab program 8**

**Write a program**

**a) To construct a binary Search tree.**

**b) To traverse the tree using all the methods i.e., in-order, preorder and post order**

**c) To display the elements in the tree.**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct node{**

**int value;**

**struct node \*left;**

**struct node \*right;**

**}\*root=NULL,\*temp=NULL;**

**void insert()**

**{**

**int data;**

**printf("Enter data to be inserted-");**

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

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

**temp->value=data;**

**temp->left=temp->right=NULL;**

**if(root==NULL)**

**root=temp;**

**else**

**search(root);**

**}**

**void search(struct node\*t)**

**{**

**if((temp->value>t->value)&&(t->right!=NULL))**

**search(t->right);**

**else if((temp->value>t->value)&&(t->right==NULL))**

**t->right=temp;**

**else if ((temp->value<t->value)&&(t->left!=NULL))**

**search(t->left);**

**else if((temp->value<t->value)&&(t->left==NULL))**

**t->left=temp;**

**}**

**void inorder(struct node \*t)**

**{**

**if(root==NULL)**

**{**

**printf("No elements in the tree\n");**

**return;**

**}**

**if(t->left!=NULL)**

**inorder(t->left);**

**printf("%d->",t->value);**

**if(t->right!=NULL)**

**inorder(t->right);**

**}**

**void preorder(struct node \*t)**

**{**

**if(root==NULL)**

**{**

**printf("No elements in the tree\n");**

**return;**

**}**

**printf("%d->",t->value);**

**if(t->left!=NULL)**

**preorder(t->left);**

**if(t->right!=NULL)**

**preorder(t->right);**

**}**

**void postorder(struct node \*t)**

**{**

**if(root==NULL)**

**{**

**printf("No elements in the tree\n");**

**return;**

**}**

**if(t->left!=NULL)**

**postorder(t->left);**

**if(t->right!=NULL)**

**postorder(t->right);**

**printf("%d->",t->value);**

**}**

**struct node\* maxvaluenode(struct node\* t)**

**{**

**struct node\* current = t;**

**while (current && current->right != NULL)**

**current = current->right;**

**return current;**

**}**

**int main()**

**{**

**int choice;**

**while(1)**

**{**

**printf("\nBinary Search Tree\n");**

**printf("1. insert an element into tree\n");**

**printf("2. to print the tree elements in inorder traversal\n");**

**printf("3. to print the tree elements in preorder traversal\n");**

**printf("4. to print the tree elements in postorder traversal\n");**

**printf("5. to exit\n");**

**printf("Enter your choice\n");**

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

**switch(choice)**

**{**

**case 1:**

**insert();**

**break;**

**case 2:**

**printf("\*\*\*inorder traversal\*\*\*\n");**

**inorder(root);**

**break;**

**case 3:**

**printf("\*\*\*preorder traversal\*\*\*\n");**

**preorder(root);**

**break;**

**case 4:**

**printf("\*\*\*postorder traversal\*\*\*\n");**

**postorder(root);**

**break;**

**case 5:**

**exit(0);**

**default:**

**printf("Invalid choice");**

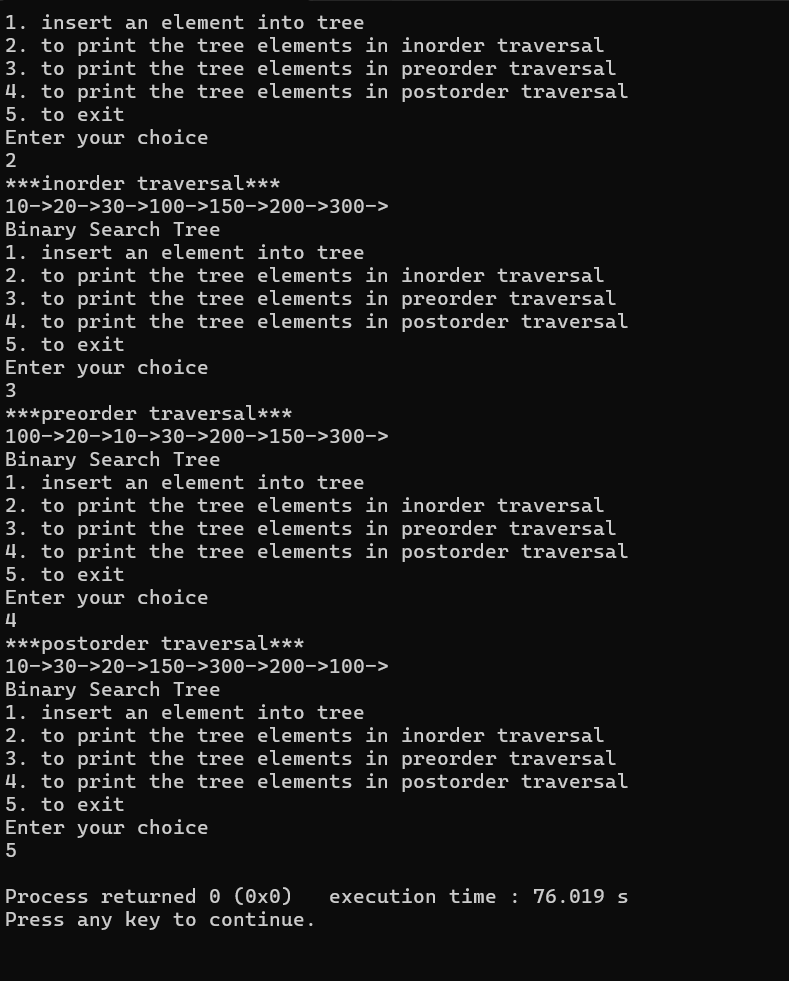
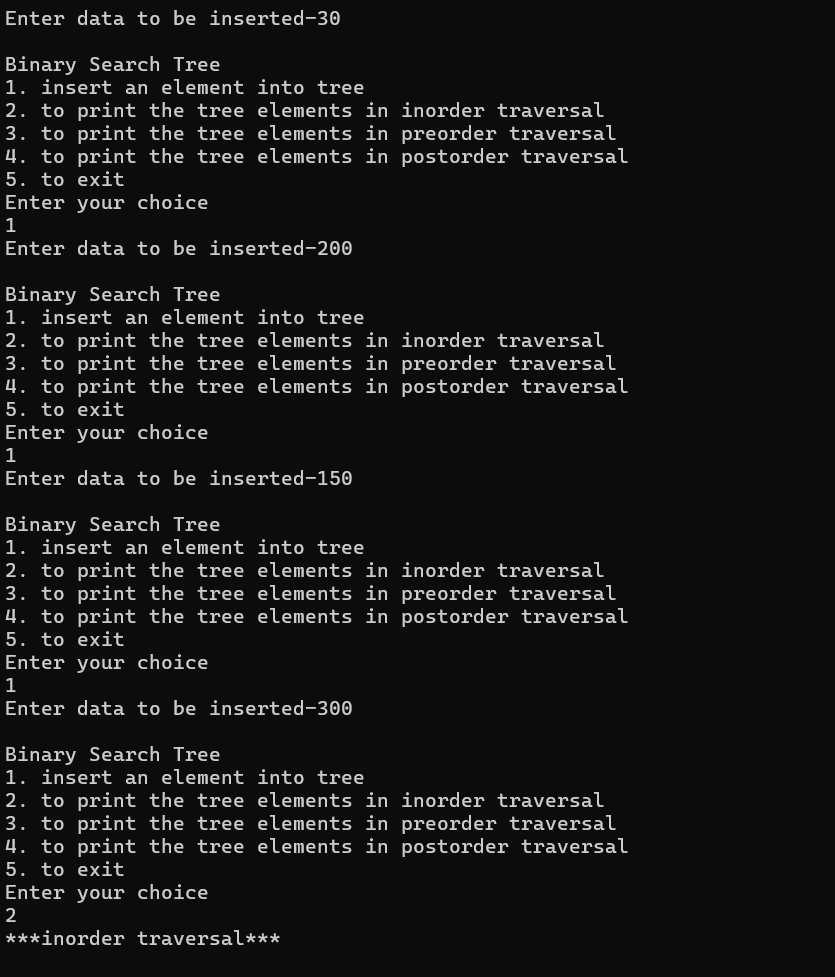
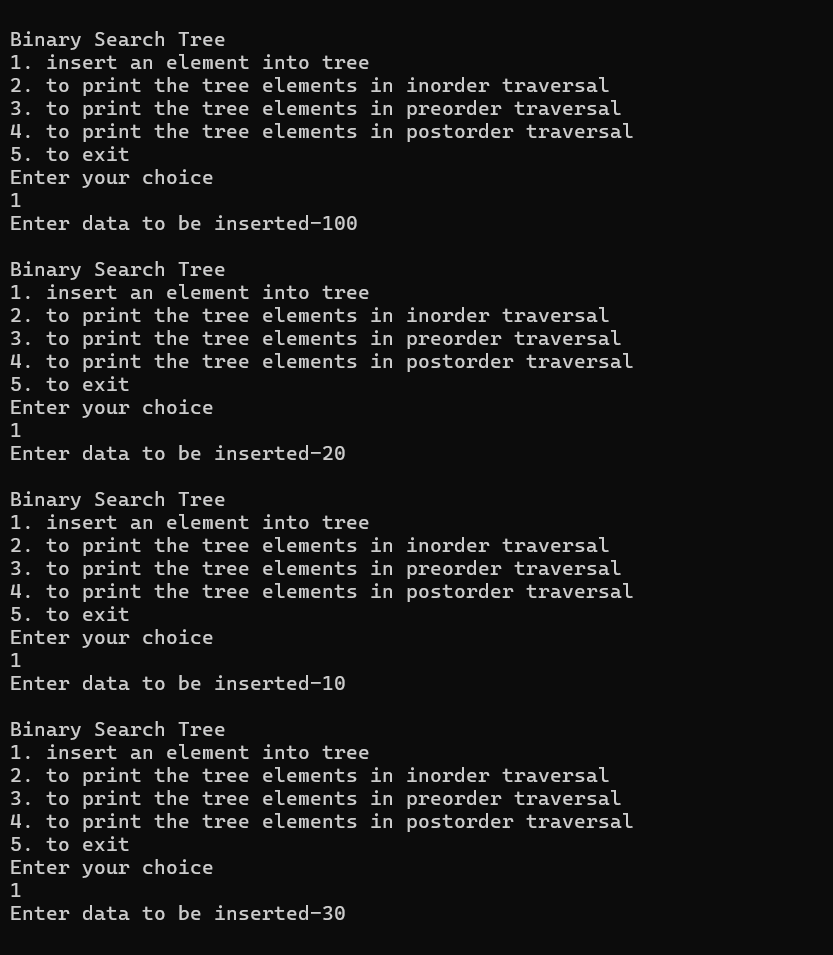
**break;**

**}**

**}**

**return 0;**

**}**

****

**LEETCODE 4 :**

struct ListNode\* rotateRight(struct ListNode\* head, int k) {

    if (head == NULL || head->next == NULL || k == 0) // If the list is empty or has only one node or k is zero, no need to rotate

        return head;

    int length = 1;

    struct ListNode \*tail = head;

    while (tail->next != NULL) { // Finding the length of the list and the tail node

        length++;

        tail = tail->next;

    }

    k = k % length; // Adjusting k if it's greater than the length of the list

    if (k == 0) // If k becomes 0 after adjustment, no rotation is needed

        return head;

    struct ListNode \*new\_tail = head;

    for (int i = 0; i < length - k - 1; i++) { // Finding the new tail node after rotation

        new\_tail = new\_tail->next;

    }

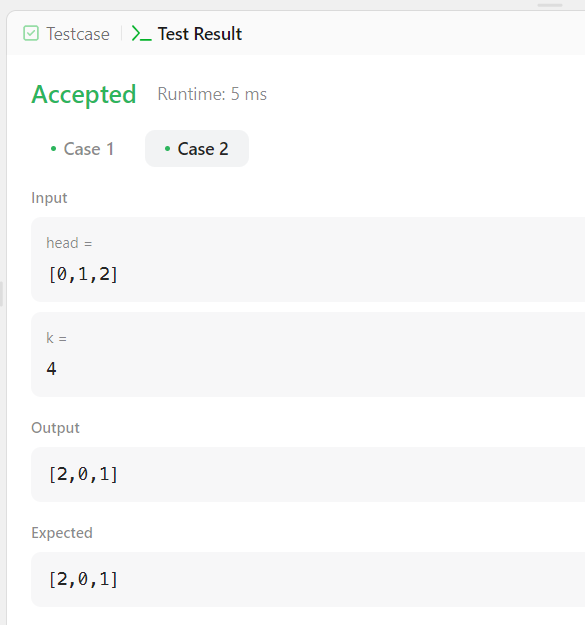
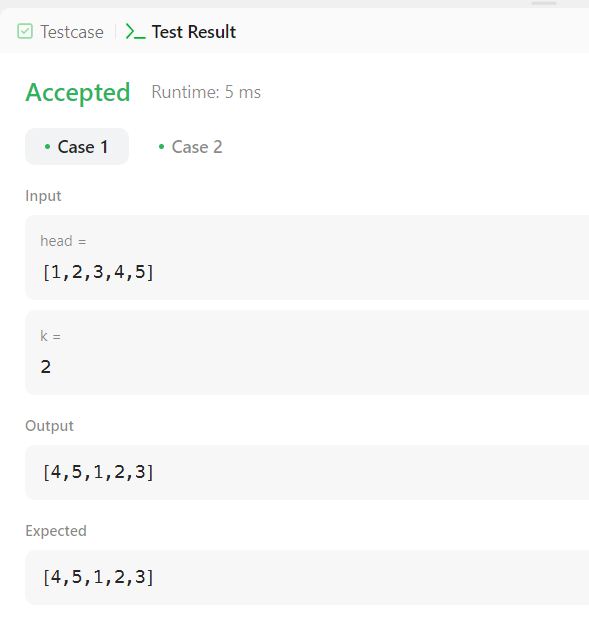
    struct ListNode \*new\_head = new\_tail->next; // New head after rotation

    new\_tail->next = NULL; // Breaking the link between the new tail and the next node to form a new list

    tail->next = head; // Connecting the original tail to the original head to form a circular list

    return new\_head;

}

****

**Lab program 9**

1. **Write a program to traverse a graph using BFS method.**

**#include <stdio.h>**

**#define MAX\_VERTICES 10**

**int n, i, j, visited[MAX\_VERTICES], queue[MAX\_VERTICES], front = 0, rear = 0;**

**int adj[MAX\_VERTICES][MAX\_VERTICES];**

**void bfs(int v) {**

**visited[v] = 1;**

**queue[rear++] = v;**

**while (front < rear) {**

**int current = queue[front++];**

**printf("%d\t", current);**

**for (int i = 0; i < n; i++) { // Corrected loop condition**

**if (adj[current][i] && !visited[i]) {**

**visited[i] = 1; // Corrected setting visited flag**

**queue[rear++] = i;**

**}**

**}**

**}**

**}**

**int main() {**

**int v;**

**printf("Enter the number of vertices: ");**

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

**for (i = 0; i < n; i++) { // Corrected loop condition**

**visited[i] = 0; // Initialize all nodes as unvisited**

**}**

**printf("Enter graph data in matrix form:\n");**

**for (i = 0; i < n; i++) // Corrected loop condition**

**for (j = 0; j < n; j++) // Corrected loop condition**

**scanf("%d", &adj[i][j]);**

**printf("Enter the starting vertex: ");**

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

**bfs(v);**

**for (i = 0; i < n; i++) { // Corrected loop condition**

**if (!visited[i]) {**

**printf("\nBFS is not possible. Not all nodes are reachable.\n");**

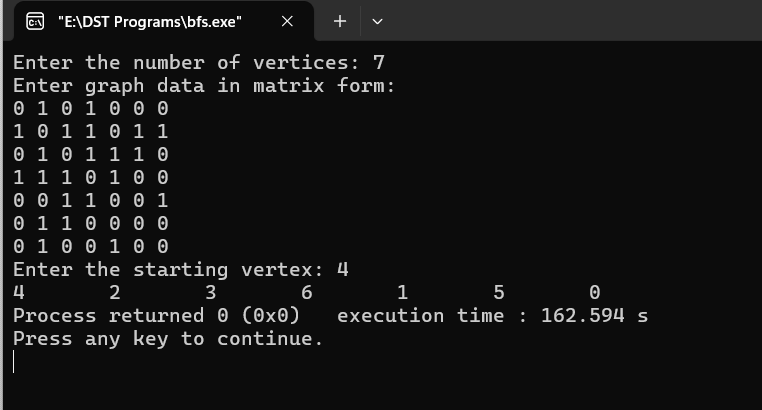
**return 0;**

**}**

**}**

**return 0;**

**}**

****

1. **Write a program to check whether given graph is connected or not using DFS method.**

**#include <stdio.h>**

**#include <stdbool.h>**

**#define MAX\_VERTICES 10**

**int n, i, j, visited[MAX\_VERTICES];**

**int adj[MAX\_VERTICES][MAX\_VERTICES];**

**void dfs(int v) {**

**visited[v] = 1;**

**for (int i = 0; i < n; i++) {**

**if (adj[v][i] && !visited[i]) {**

**dfs(i);**

**}**

**}**

**}**

**int main() {**

**int v;**

**printf("Enter the number of vertices: ");**

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

**for (i = 0; i < n; i++) {**

**visited[i] = 0; // Initialize all nodes as unvisited**

**}**

**printf("Enter graph data in matrix form:\n");**

**for (i = 0; i < n; i++)**

**for (j = 0; j < n; j++)**

**scanf("%d", &adj[i][j]);**

**printf("Enter the starting vertex: ");**

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

**dfs(v);**

**for (i = 0; i < n; i++) {**

**if (!visited[i]) {**

**printf("\nThe graph is not connected.\n");**

**return 0;**

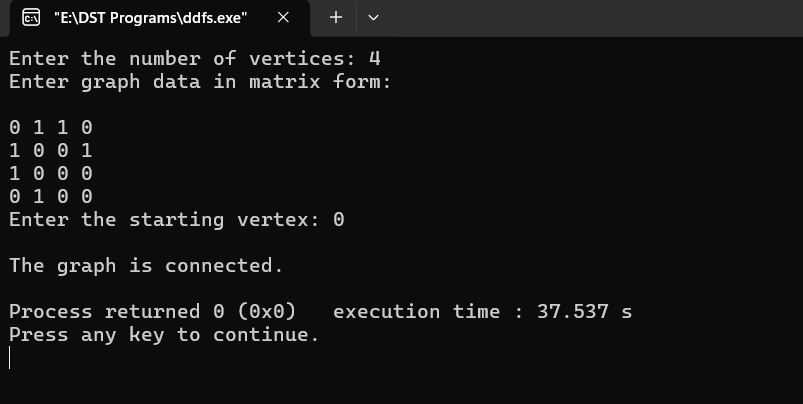
**}**

**}**

**printf("\nThe graph is connected.\n");**

**return 0;**

**}**

****

**HACKER RANK ON TREE:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int id;

int depth;

struct node \*left, \*right;

};

void

inorder(struct node\* tree)

{

if(tree == NULL)

return;

inorder(tree->left);

printf("%d ",tree->id);

inorder((tree->right));

}

int

main(void)

{

int no\_of\_nodes, i = 0;

int l,r, max\_depth,k;

struct node\* temp = NULL;

scanf("%d",&no\_of\_nodes);

struct node\* tree = (struct node \*) calloc(no\_of\_nodes , sizeof(struct node));

tree[0].depth = 1;

while(i < no\_of\_nodes )

{

tree[i].id = i+1;

scanf("%d %d",&l,&r);

if(l == -1)

tree[i].left = NULL;

else

{

tree[i].left = &tree[l-1];

tree[i].left->depth = tree[i].depth + 1;

max\_depth = tree[i].left->depth;

}

if(r == -1)

tree[i].right = NULL;

else

{

tree[i].right = &tree[r-1];

tree[i].right->depth = tree[i].depth + 1;

max\_depth = tree[i].right->depth+2;

}

i++;

}

scanf("%d", &i);

while(i--)

{

scanf("%d",&l);

r = l;

while(l <= max\_depth)

{

for(k = 0;k < no\_of\_nodes; ++k)

{

if(tree[k].depth == l)

{

temp = tree[k].left;

tree[k].left = tree[k].right;

tree[k].right = temp;

}

}

l = l + r;

}

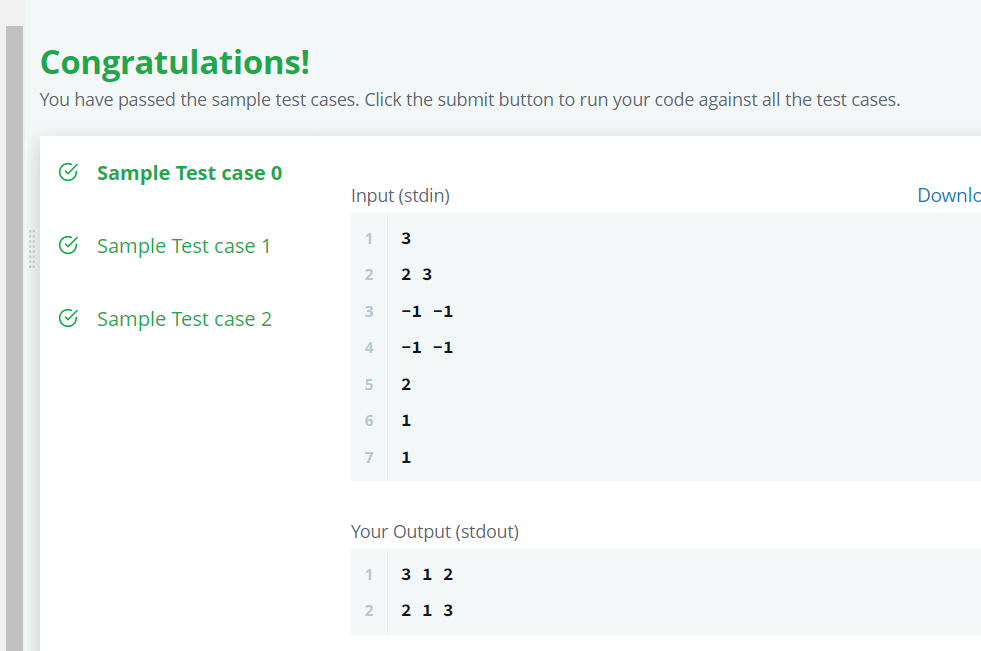
inorder(tree);

printf("\n");

}

return 0;

}

****

**Lab program 10**

**Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -&gt; L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX\_EMPLOYEES 100 // Maximum number of employees**

**#define HASH\_TABLE\_SIZE 7 // Size of the hash table**

**// Structure for employee record**

**struct Employee {**

**int key; // 4-digit key**

**// Other employee details can be added here**

**};**

**// Function prototypes**

**int hashFunction(int key);**

**void insertEmployee(struct Employee employees[], int hashTable[], struct Employee emp);**

**void displayHashTable(int hashTable[]);**

**int main() {**

**struct Employee employees[MAX\_EMPLOYEES]; // Array to hold employee records**

**int hashTable[HASH\_TABLE\_SIZE] = {0}; // Hash table initialized with 0**

**int n, m, i;**

**// Input the number of employees**

**printf("Enter the number of employees: ");**

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

**// Input employee records**

**printf("Enter employee records:\n");**

**for (i = 0; i < n; ++i) {**

**printf("Employee %d:\n", i + 1);**

**printf("Enter key: ");**

**scanf("%d", &employees[i].key);**

**// Additional details can be input here**

**insertEmployee(employees, hashTable, employees[i]);**

**}**

**// Display the hash table**

**printf("\nHash Table:\n");**

**displayHashTable(hashTable);**

**return 0;**

**}**

**// Hash function: H(K) = K mod m**

**int hashFunction(int key) {**

**return key % HASH\_TABLE\_SIZE;**

**}**

**// Function to insert an employee into the hash table**

**void insertEmployee(struct Employee employees[], int hashTable[], struct Employee emp) {**

**int index = hashFunction(emp.key);**

**// Linear probing to resolve collisions**

**while (hashTable[index] != 0) {**

**index = (index + 1) % HASH\_TABLE\_SIZE;**

**}**

**// Insert the employee key into the hash table**

**hashTable[index] = emp.key;**

**}**

**// Function to display the hash table**

**void displayHashTable(int hashTable[]) {**

**int i;**

**for (i = 0; i < HASH\_TABLE\_SIZE; ++i) {**

**printf("%d -> ", i);**

**if (hashTable[i] == 0) {**

**printf("Empty\n");**

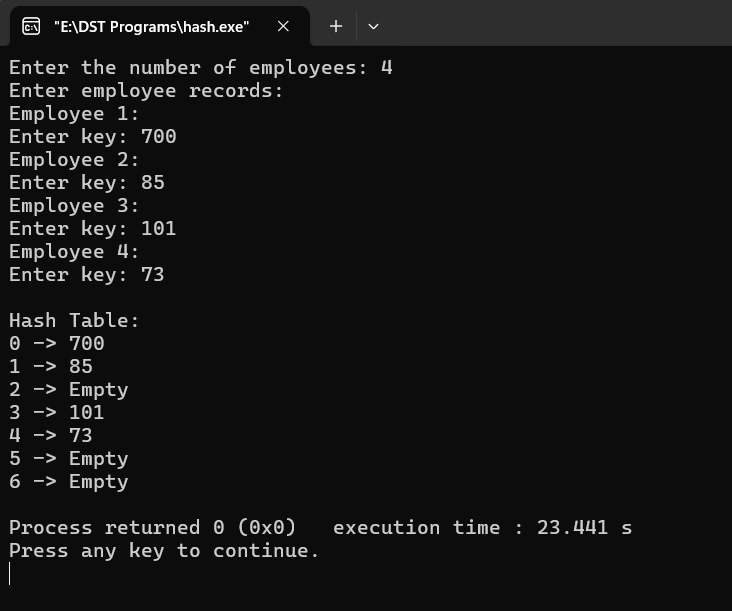
**} else {**

**printf("%d\n", hashTable[i]);**

**}**

**}**

**}**

****