VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C

Submitted by

SAI KRUTHIN CR

(1BM22CS232)

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
June-2023 to September-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled “**Data Structures using C**” carried out by **SAI KRUTHIN CR (1BM22CS232),** who is 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 academic semester December-2023 to March-2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **Data Structures using C (23CS3PCDST)** work prescribed for the said degree.

Radhika AD
Assistant Professer
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

	5b) Program - Leetcode platform	
6	6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists. 6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.	46-58
7	7a) 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 Display the contents of the list 7b) Program - Leetcode platform	59-65
8	8a) 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 To display the elements in the tree. 8b) Program - Leetcode platform	66-72
9	9a) Write a program to traverse a graph using BFS method. 9b) Write a program to check whether given graph is connected or not using DFS method.	73-79
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 -> 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.	-

Course Outcome

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyse data structure operations for a given problem.
CO3	CO3 Design and implement operations of linear and nonlinear data structure.
CO4	Conduct practical experiments for demonstrating the operations of different data structures and sorting techniques.

LAB 1

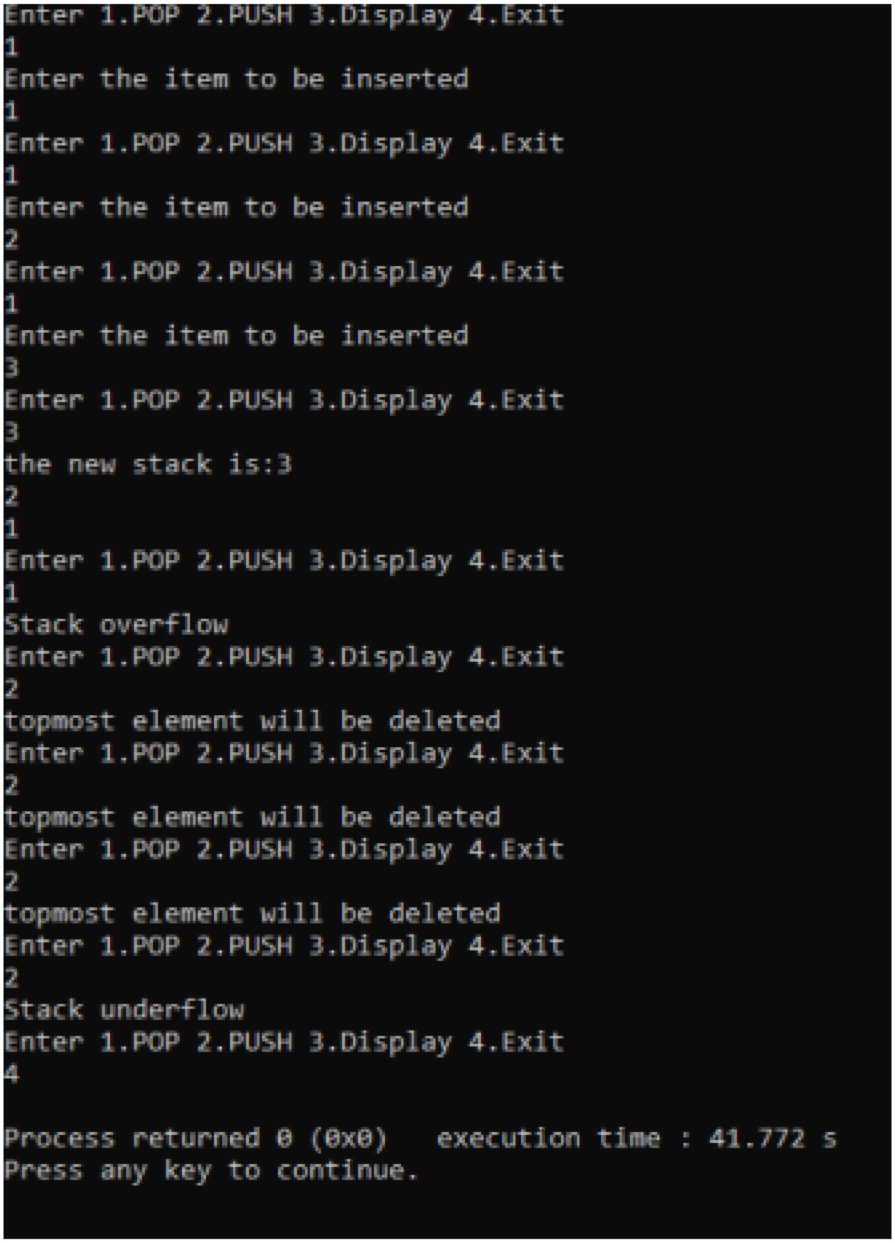
1. Write a program to simulate the working of stack using an array with the following : a) Push b) Pop c) Display

```
#include<stdio.h>

#include<stdlib.h>

#define N 5

int stack[N];

int top=-1;

void push()

{

 if(top==N)

 printf("Stack overflow");

 else

 {

 int item;

 printf("Enter the item to be inserted");  scanf("%d",&item);

 top++;

 stack[top]=item;

 }
```

```
}

void pop()

{

 if(top<0)
 printf("Stack underflow");

 else

 {

 printf("topmost element will be deleted\n");  top--;


 }

}

 void display()

{

 printf("the new stack is:");

 for(int i=top;i>=0;i--)

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

}

void main()

{

 int choice;

 while(1)
```

```
{


printf("Enter 1.POP 2.PUSH 3.Display 4.Exit");  scanf("%d",&choice);

switch(choice)

{

case 1: push();break;

case 2:pop();break;
case 3: display();break;

case 4:exit(0);


}

}

}
```

Output:



```
Enter 1.POP 2.PUSH 3.Display 4.Exit
1
Enter the item to be inserted
1
Enter 1.POP 2.PUSH 3.Display 4.Exit
1
Enter the item to be inserted
2
Enter 1.POP 2.PUSH 3.Display 4.Exit
1
Enter the item to be inserted
3
Enter 1.POP 2.PUSH 3.Display 4.Exit
3
the new stack is:3
2
1
Enter 1.POP 2.PUSH 3.Display 4.Exit
1
Stack overflow
Enter 1.POP 2.PUSH 3.Display 4.Exit
2
topmost element will be deleted
Enter 1.POP 2.PUSH 3.Display 4.Exit
2
topmost element will be deleted
Enter 1.POP 2.PUSH 3.Display 4.Exit
2
topmost element will be deleted
Enter 1.POP 2.PUSH 3.Display 4.Exit
2
Stack underflow
Enter 1.POP 2.PUSH 3.Display 4.Exit
4

Process returned 0 (0x0)   execution time : 41.772 s
Press any key to continue.
```

LAB 2

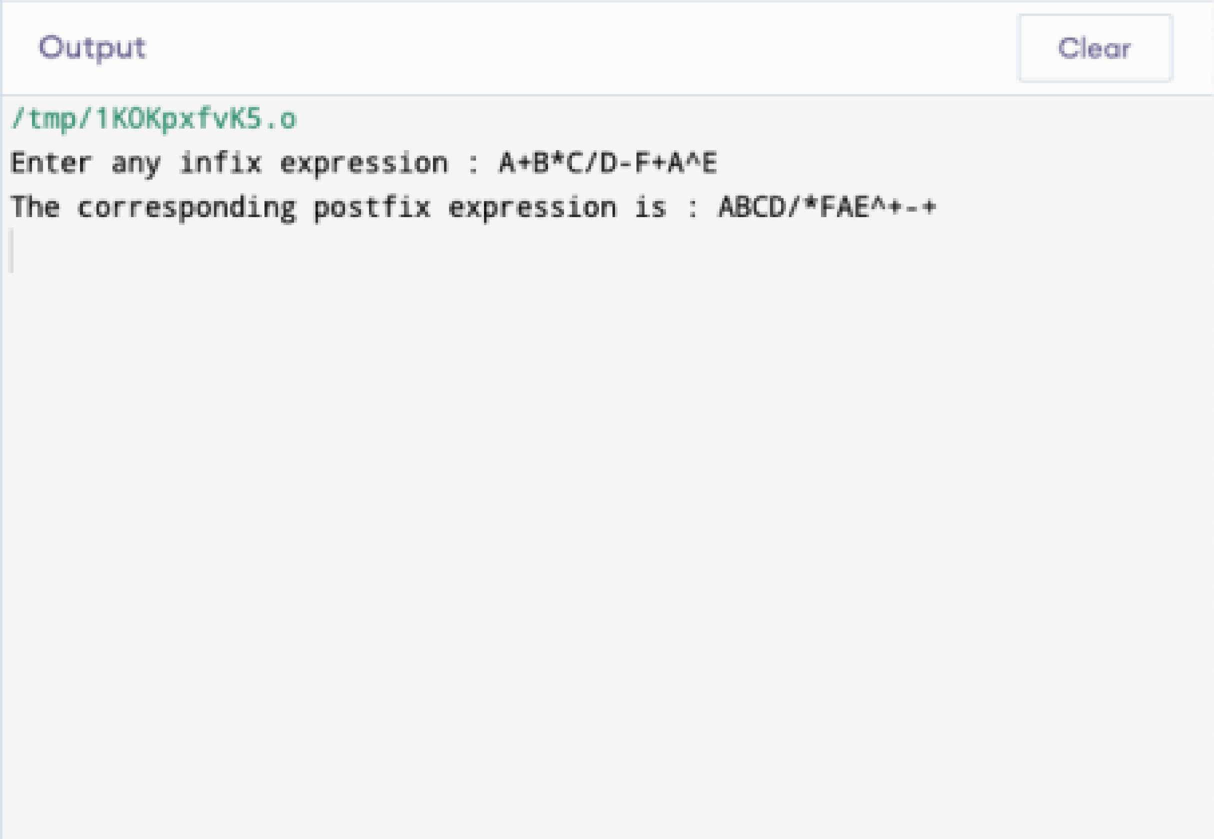
Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply), / (divide) and ^ (power).

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#define MAX 100
char st[MAX];
int top = -1;
void push(char st[], char);
char pop(char st[]);
void InfixtoPostfix(char source[], char target[]);
int getpri(char);

void main()
{
char infix[100], postfix[100];
printf("\n Enter any infix expression : ");
gets(infix);
strcpy(postfix, "");
InfixtoPostfix(infix, postfix);
printf("\n The corresponding postfix expression is : ");
puts(postfix);
}

void InfixtoPostfix(char source[], char target[])
```

```
{
int i = 0, j = 0;
char temp;
strcpy(target, "");
while (source[i] != '\0')
{
if (source[i] == '(')
{
push(st, source[i]);
i++;
}
else if (source[i] == ')')
{
while ((top != -1) && (st[top] != '('))
{
target[j] = pop(st);
j++;
}
if (top == -1)
{
printf("\n INCORRECT EXPRESSION");
exit(1);
}
temp = pop(st);
i++;
}
else if (isdigit(source[i]) || isalpha(source[i]))
{
target[j] = source[i];
j++;
i++;
```

```
}
else if (source[i] == '+' || source[i] == '-' || source[i] == '*' ||
source[i] == '/' || source[i] == '%' || source[i] == '^')
{
while ((top != -1) && (st[top] != '(') && (getpri(st[top]) >
        getpri(source[i]))) {
target[j] = pop(st);
j++;
}
push(st, source[i]);
i++;
}
else
{
printf("\n INCORRECT ELEMENT IN EXPRESSION");
exit(1);
}
}
while ((top != -1) && (st[top] != '('))
{
target[j] = pop(st);
j++;
}
target[j] = '\0';
}
int getpri(char op)
{
if (op == '^')
return 2;
else if (op == '/' || op == '*' || op ==
        '%') return 1;
else if (op == '+' || op == '-')
return 0;
}
void push(char st[], char val)
{
if (top == MAX - 1)
printf("\n STACK OVERFLOW"); else
{
top++;
st[top] = val;
}
}
char pop(char st[])
{
char val = ' ';
if (top == -1)
printf("\n STACK UNDERFLOW"); else
{
val = st[top];
top--;
}
return val;
}
```

Output:

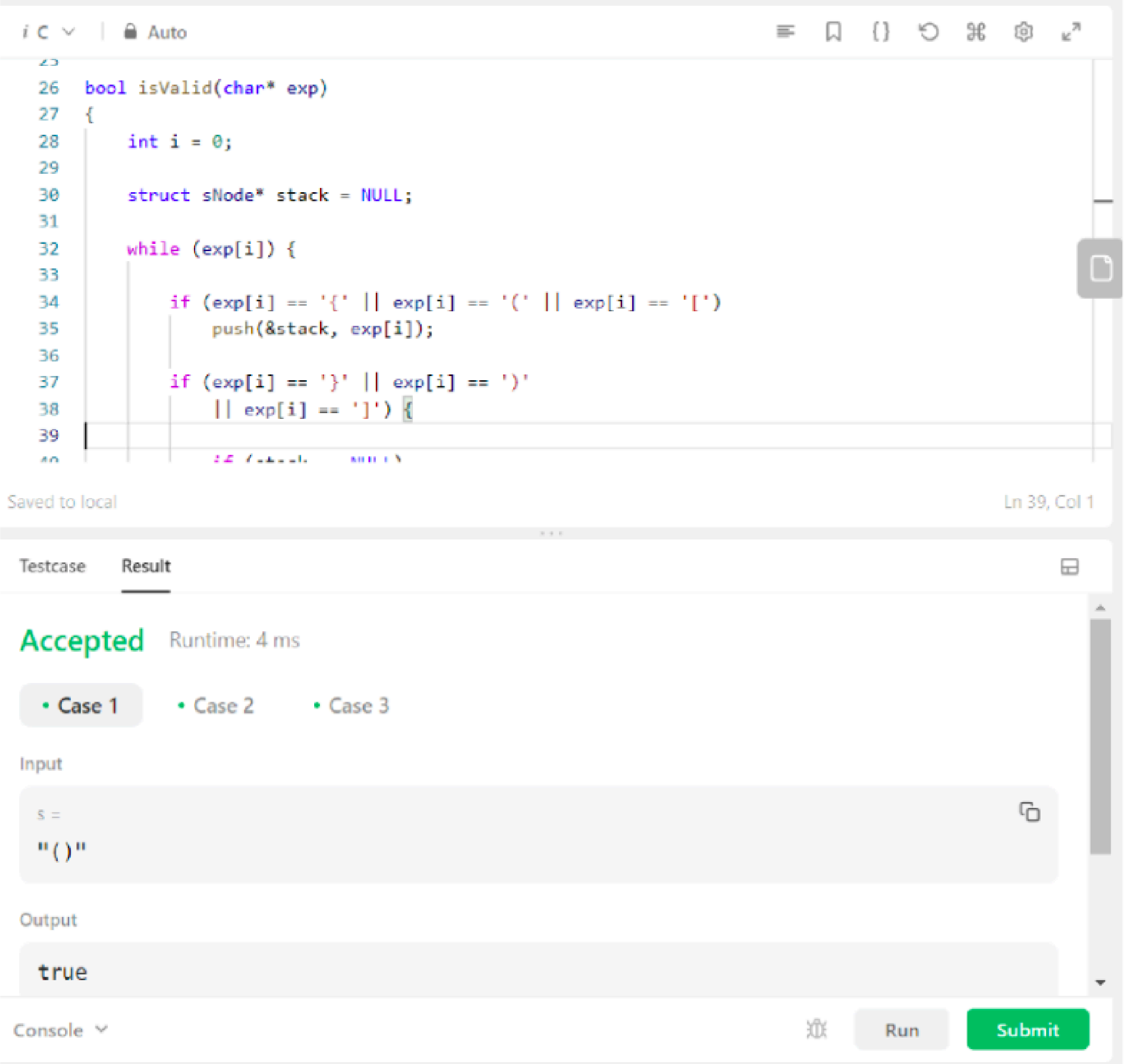


```
/tmp/1KOKpxfvK5.o
Enter any infix expression : A+B*C/D-F+A^E
The corresponding postfix expression is : ABCD/*FAE^+-+
```

2b) Leetcode Question - Valid parentheses

```
bool isValid(char* s) {
int len = strlen(s);
char stack[len];
int top = -1;
char a;

for(int i = 0; i < len; i++) {
if(s[i] == '(' || s[i] == '[' || s[i] == '{')
stack[++top] = s[i];
else {
if(top == -1)
return false;
else {
a = stack[top];
 if((a == '(' && s[i] == ')') || (a == '[' && s[i] == ']') || (a == '{' && s[i] == '}'))
top--;
else
return false;
}
}

}

if(top == -1)

return true;

else
return false;

}
```

OUTPUT:



```c
bool isValid(char* exp)
{
    int i = 0;

    struct sNode* stack = NULL;

    while (exp[i]) {

        if (exp[i] == '{' || exp[i] == '(' || exp[i] == '[')
            push(&stack, exp[i]);

        if (exp[i] == '}' || exp[i] == ')'
            || exp[i] == ']') {

```

Accepted Runtime: 4 ms

• Case 1 • Case 2 • Case 3

Input

s =

"()"

Output

```
true
```

LAB 3

3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations

a) Insert

b) Delete

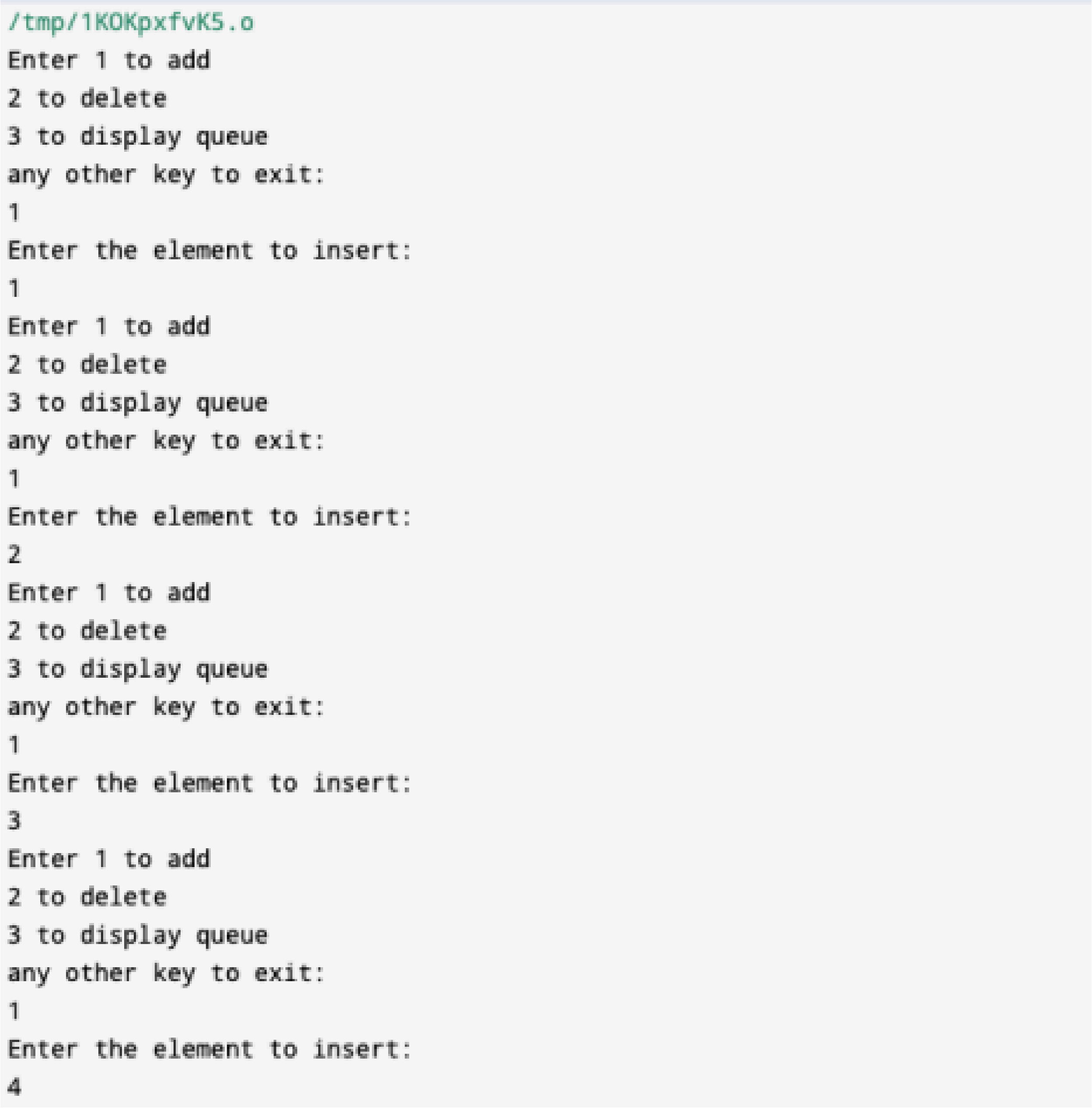
c) Display

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

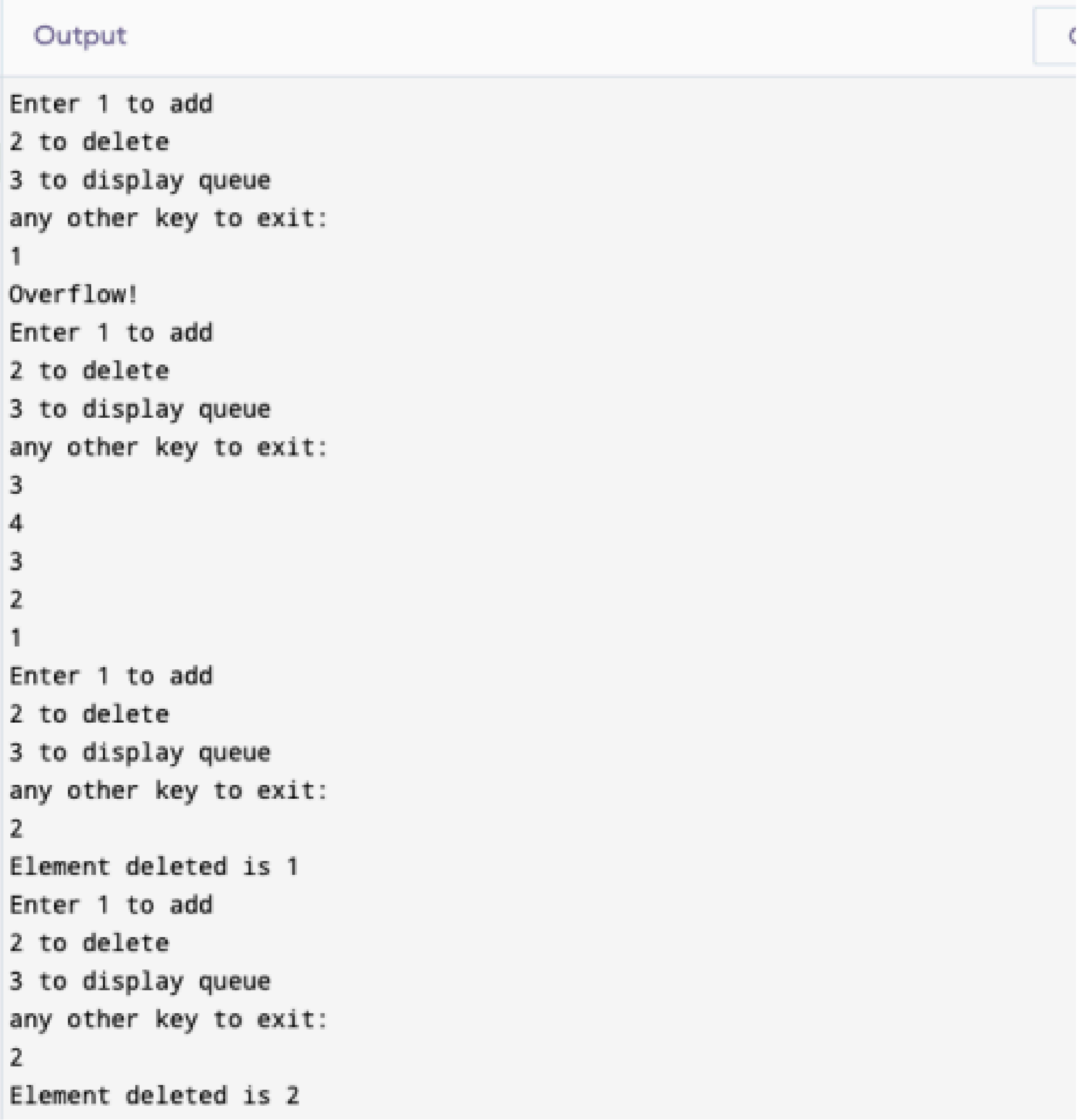
```
#include <stdio.h>
#include <stdlib.h>
#define N 4
int q[N];
int REAR = -1;
int FRONT = -1;
void enq();
void deq();
void display();
void enq() {
if (REAR == N - 1) {
printf("Overflow!\n");
} else {
int item;
printf("Enter the element to insert:\n");
scanf("%d", &item);
if (REAR == -1 && FRONT == -1) {
REAR++;
q[REAR]=item;
FRONT++;
}
```

```
else{
REAR++;
q[REAR] = item;
}
}
}
void deq() {
int val;
if (FRONT == -1 || FRONT > REAR) {
printf("Queue empty!\n");
} else {
    val = q[FRONT];
    FRONT++;
    printf("Element deleted is %d\n", val);
    }
    }
    void display() {
    int i;
    for (i = REAR; i >= FRONT; i--) {
    printf("%d\n", q[i]);
    }
    }
    int main() {
    int choice;
    while (1) {
    printf("Enter 1 to add, 2 to delete, 3 to display queue, any
    other key to exit:\n"); scanf("%d", &choice);
    switch (choice) {
    case 1:
    enq();
    break;
    case 2:
    deq();
    break;
    case 3:
    display();
    break;
    default:
    printf("Invalid key entered\n");
    exit(1);
    }
    }
    return 0;
    }
```

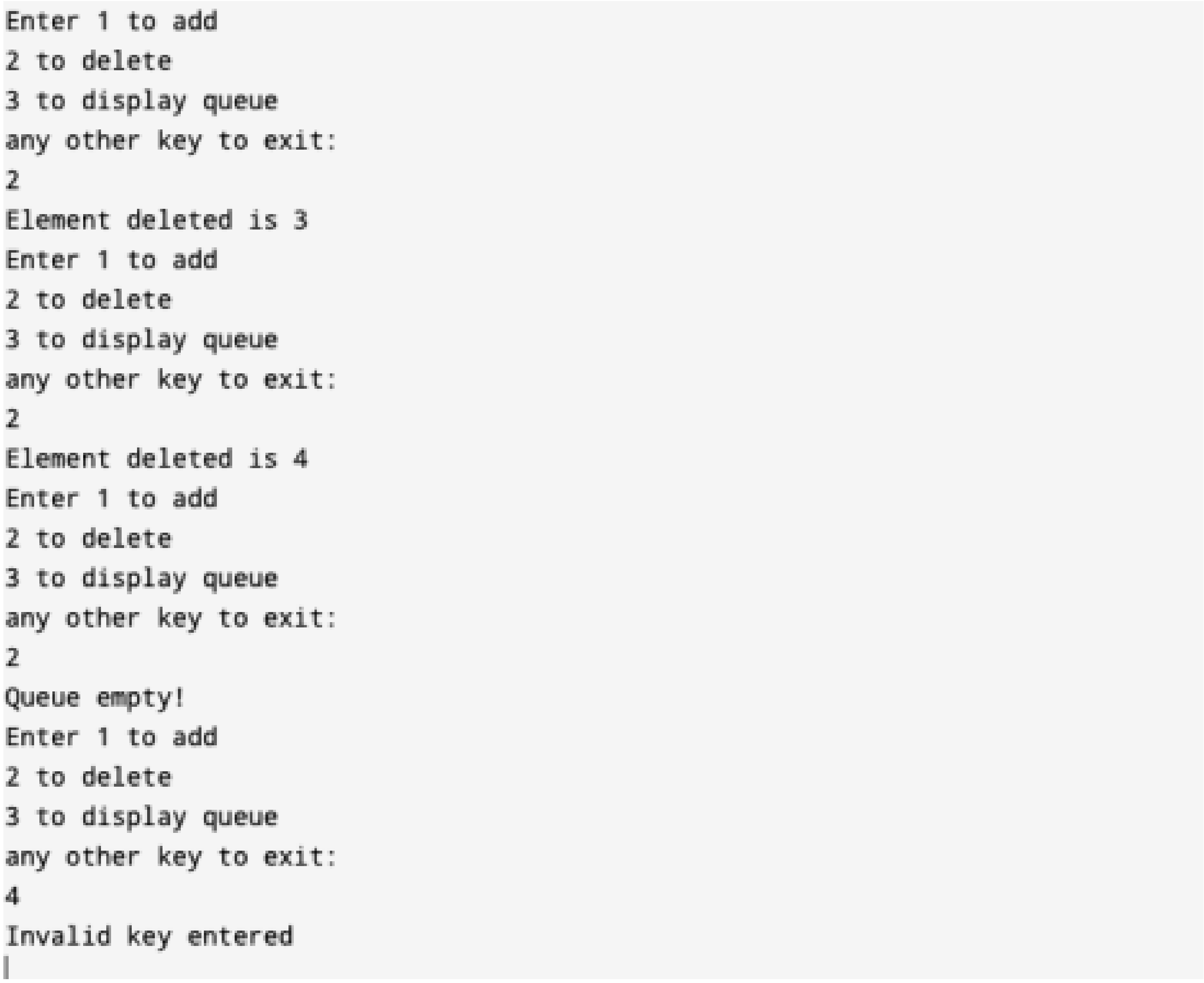
Output



```
/tmp/1KOKpxfvK5.o
Enter 1 to add
2 to delete
3 to display queue
any other key to exit:
1
Enter the element to insert:
1
Enter 1 to add
2 to delete
3 to display queue
any other key to exit:
1
Enter the element to insert:
2
Enter 1 to add
2 to delete
3 to display queue
any other key to exit:
1
Enter the element to insert:
3
Enter 1 to add
2 to delete
3 to display queue
any other key to exit:
1
Enter the element to insert:
4
```



```
Enter 1 to add
2 to delete
3 to display queue
any other key to exit:
1
Overflow!
Enter 1 to add
2 to delete
3 to display queue
any other key to exit:
3
4
3
2
1
Enter 1 to add
2 to delete
3 to display queue
any other key to exit:
2
Element deleted is 1
Enter 1 to add
2 to delete
3 to display queue
any other key to exit:
2
Element deleted is 2
```



```
Enter 1 to add
2 to delete
3 to display queue
any other key to exit:
2
Element deleted is 3
Enter 1 to add
2 to delete
3 to display queue
any other key to exit:
2
Element deleted is 4
Enter 1 to add
2 to delete
3 to display queue
any other key to exit:
2
Queue empty!
Enter 1 to add
2 to delete
3 to display queue
any other key to exit:
4
Invalid key entered
```

3b) WAP to simulate the working of a circular 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

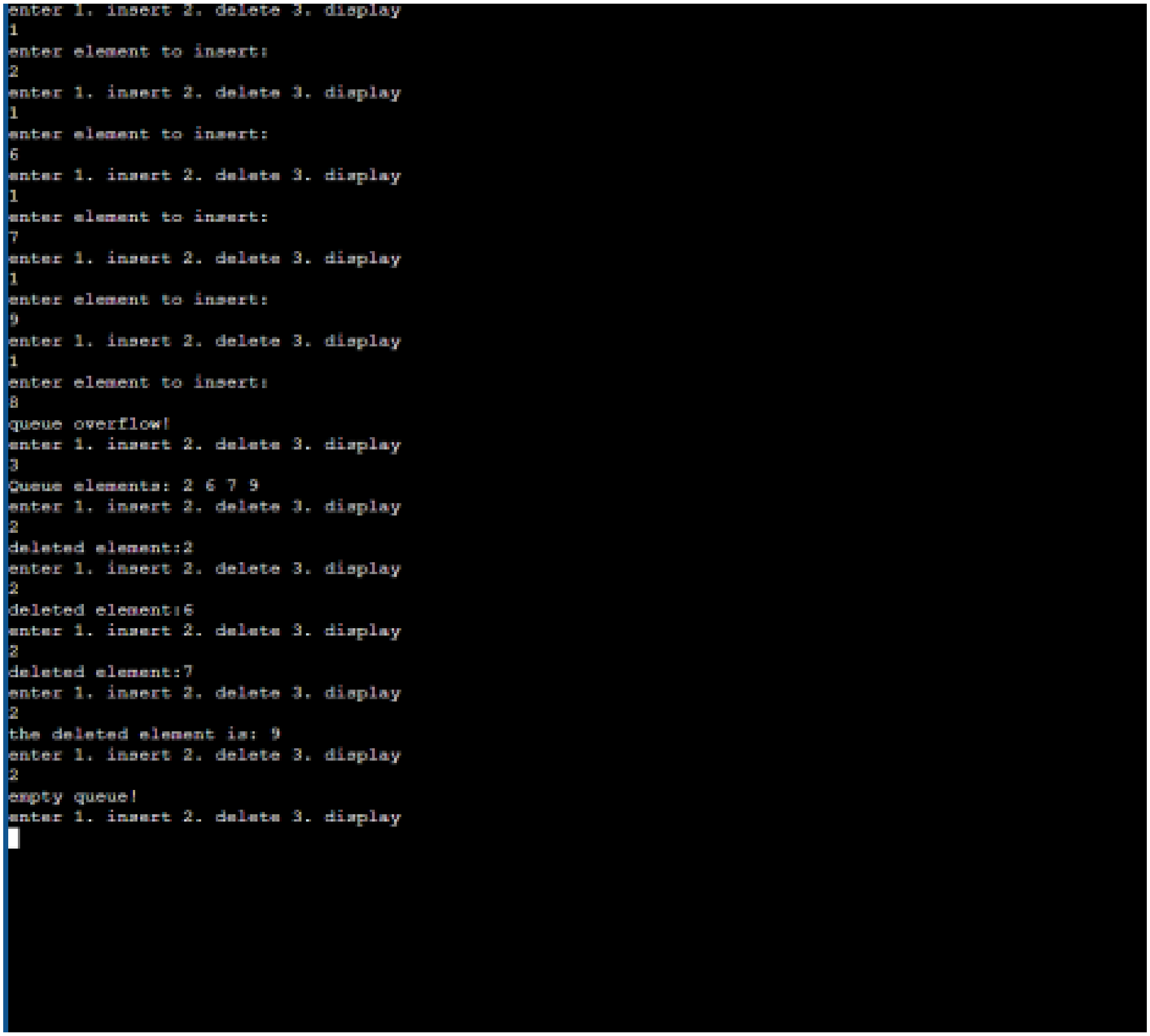
```
b) #include <stdio.h>
#include <stdlib.h>
#define N 4
int q[N];
int REAR=-1;
int FRONT=-1;
void enq();
void deq();
void display();

void enq(){
int item;
printf("enter element to insert:\n");
scanf("%d",&item);
if(FRONT==-1 && REAR==-1){
FRONT=REAR=0;
q[REAR]=item;
}
```

```
else if((REAR+1)%N==FRONT){
printf("queue overflow!\n");
}
else{
REAR=(REAR+1)%N;
q[REAR]=item;
}
}
void deq(){
if(FRONT==-1 && REAR==-1){
printf("empty queue!\n");
}
else if(FRONT==REAR){
printf("the deleted element is: %d\n",q[FRONT]);
FRONT=REAR=-1;
}
else{
printf("deleted element:%d\n",q[FRONT]);
FRONT=(FRONT+1)%N;
}
}
void display(){
```

```
int i;



if (FRONT == -1 && REAR == -1) {
printf("Queue is empty\n");
}
else {
printf("Queue elements: ");
i = FRONT;
while (i != REAR) {
printf("%d ", q[i]);
i = (i + 1) % N;
}
printf("%d", q[REAR]); // Print the last element
}
printf("\n");
}

void main(){
int choice;
while(1){
printf("enter 1. insert 2. delete 3. display\n");
scanf("%d",&choice);
switch(choice){
```

```
case 1: enq();
break;



case 2: deq();
break;
case 3: display();
break;
default: printf("invalid entry\n");
exit(0);
}
}
}
```

OUTPUT:



```
enter 1. insert 2. delete 3. display
1
enter element to insert:
2
enter 1. insert 2. delete 3. display
1
enter element to insert:
6
enter 1. insert 2. delete 3. display
1
enter element to insert:
7
enter 1. insert 2. delete 3. display
1
enter element to insert:
9
enter 1. insert 2. delete 3. display
1
enter element to insert:
8
queue overflow!
enter 1. insert 2. delete 3. display
3
Queue elements: 2 6 7 9
enter 1. insert 2. delete 3. display
2
deleted element:2
enter 1. insert 2. delete 3. display
2
deleted element:6
enter 1. insert 2. delete 3. display
2
deleted element:7
enter 1. insert 2. delete 3. display
2
the deleted element is: 9
enter 1. insert 2. delete 3. display
2
empty queue!
enter 1. insert 2. delete 3. display
```

LAB 4

4) a) 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>

typedef struct Node {
int data;
struct Node *next;
}Node;

void InsertAtBeginning( Node **head_ref,int new_data);
void InsertAtEnd( Node **head_ref,int new_data);
void Insert( Node **prev_node,int new_data,int pos);
void PrintList(Node * next);

void InsertAtBeginning( Node **head_ref,int new_data)
{
Node *new_node=(struct Node*)malloc(sizeof( Node));
new_node->data=new_data;
new_node->next=*head_ref;
*head_ref=new_node;
}
```

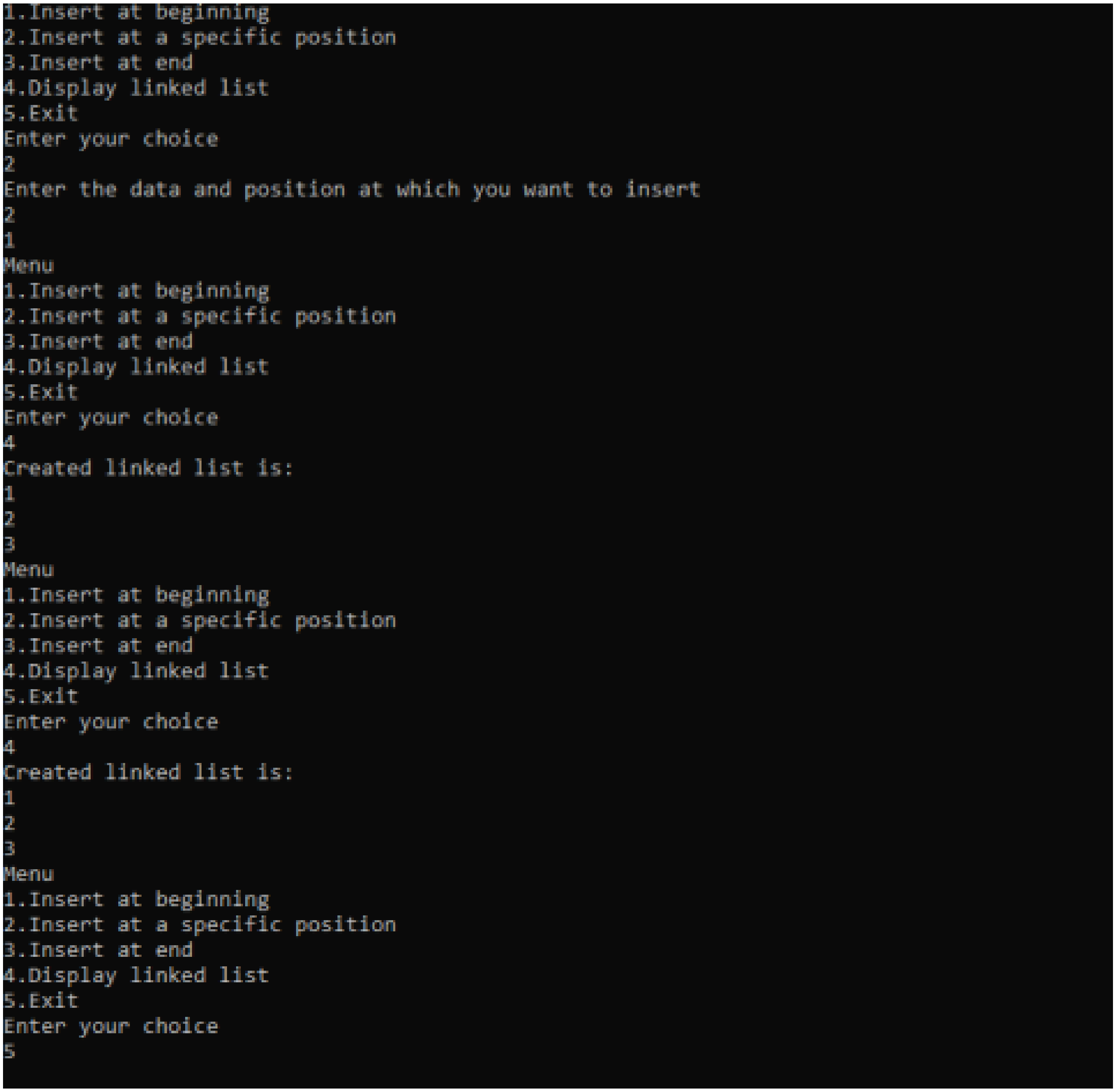
```
void InsertAtEnd(Node **head_ref,int new_data)
{
Node *new_node=(struct Node*)malloc(sizeof( Node));
Node *last=*head_ref;
new_node->data=new_data;
new_node->next=NULL;
if (*head_ref==NULL)
{
*head_ref=new_node;
return ;
}
while (last->next!=NULL)
last=last->next;
last->next=new_node;

}

void Insert(Node **head_ref,int new_data,int pos)
{
if (*head_ref ==NULL)
{
printf("Cannot be NULL\n");
return;
}
Node *temp = *head_ref;
Node *newNode = ( Node *) malloc (sizeof ( Node));
newNode->data = new_data;
newNode->next = NULL;

while (--pos>0)




{
```

```
temp = temp->next;
}
newNode->next = temp->next;
temp->next = newNode;
}


void PrintList(Node *node)
{
while (node!=NULL)
{
printf("%d\n",node->data);
node=node->next;
}
}


int main()
{
int ch,new,pos;
Node* head=NULL;
while(ch!=5)
{
printf("Menu\n");
printf("1.Insert at beginning\n");
printf("2.Insert at a specific position\n");
printf("3.Insert at end\n");
printf("4.Display linked list\n");
printf("5.Exit\n");
printf("Enter your choice\n");
```

```
scanf("%d",&ch);
switch(ch)
{
case 1:
{
printf("Enter the data you want to insert at beginning\n");
scanf("%d",&new);
InsertAtBeginning(&head,new);
break;
}
case 2:
{
printf("Enter the data and position at which you want to insert \n");
scanf("%d%d",&new,&pos);
Insert(&head,new,pos);
break;
}
case 3:
{
printf("Enter the data you want to insert at end\n");
scanf("%d",&new);
InsertAtEnd(&head,new);
break;
}
case 4:
{
printf("Created linked list is:\n");
PrintList(head);
break;
}
case 5:
```

```
{
return 0;
break;
}
case 6:
{
printf("Invalid data!");
break;
} }
} return 0;}
```

OUTPUT:

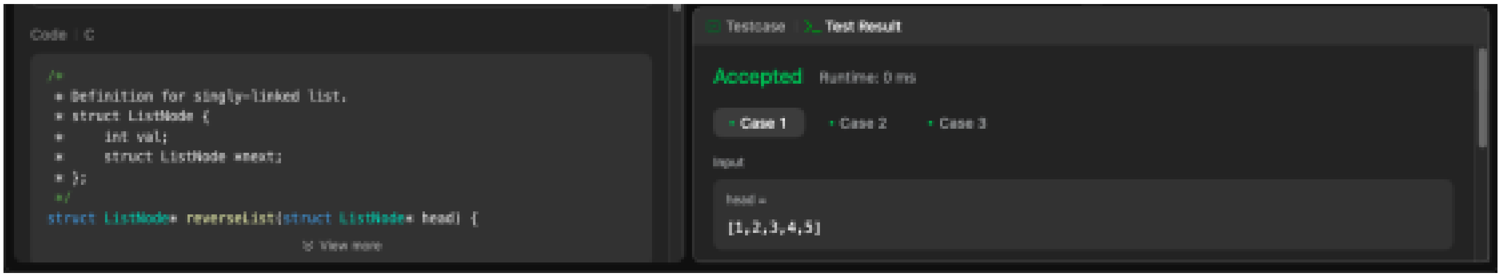


```
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
2
Enter the data and position at which you want to insert
2
1
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
4
Created linked list is:
1
2
3
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
4
Created linked list is:
1
2
3
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
5
```

4b) Leetcode Question-Reverse a Singly Linked List

```
struct ListNode* reverseList(struct ListNode* head) {

struct ListNode* prev = NULL;

struct ListNode* temp;

struct ListNode* n;

temp=head;


while (temp != NULL) {

n = temp->next;

temp->next = prev;

prev = temp;

temp = n;

}


return prev;
 }
```

OUTPUT:



```
/*
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* reverseList(struct ListNode* head) {
```

Accepted Runtime: 0 ms

head =

[1,2,3,4,5]

LAB 5

5 a)

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.

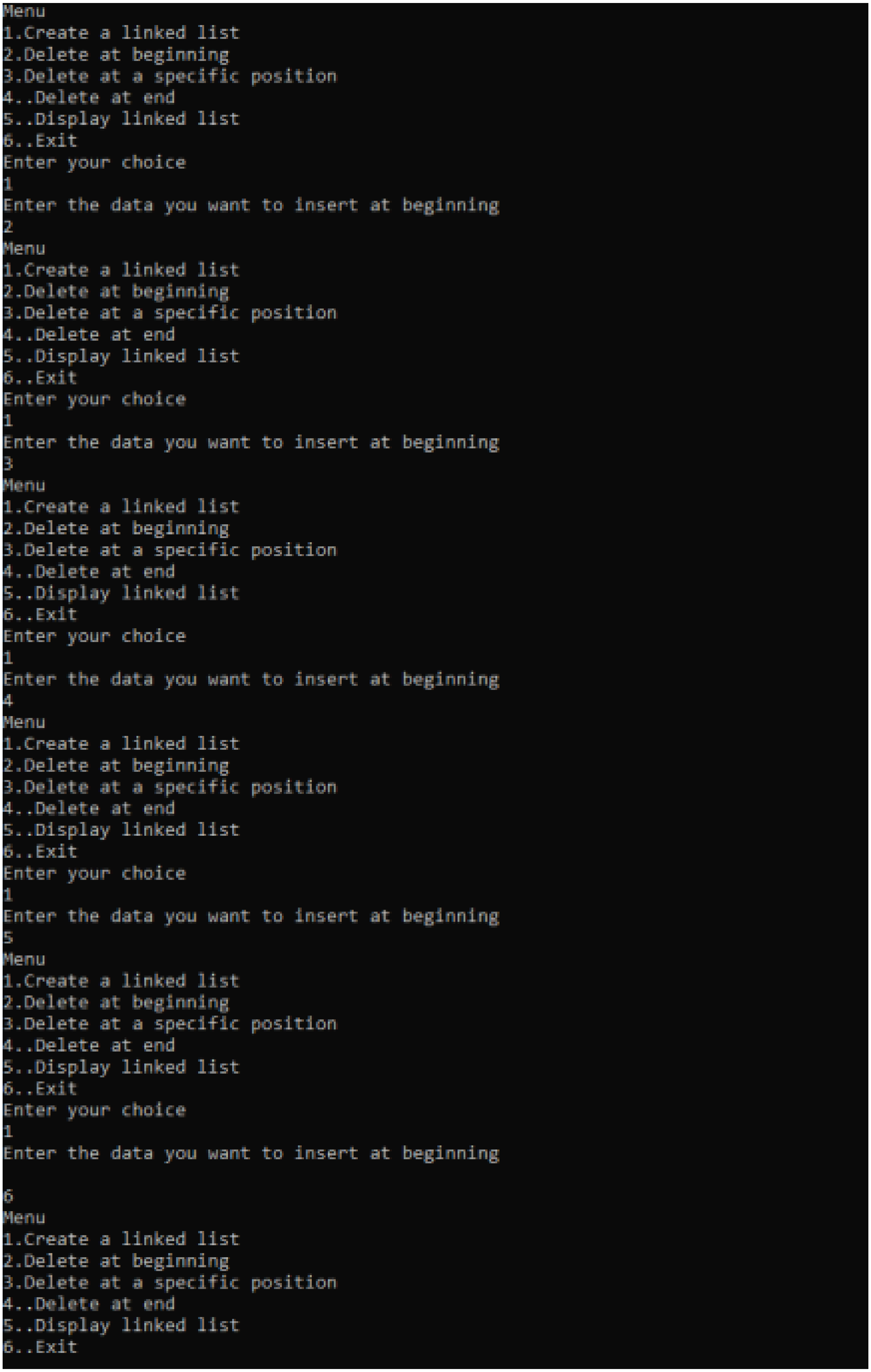
```
#include <stdio.h>
#include<stdlib.h>


typedef struct Node {
int data;
struct Node *next;
}Node;

void InsertAtBeginning( Node **head_ref,int new_data);
void DeleteAtBeginning( Node **head_ref);
void DeleteAtEnd( Node **head_ref);
void Delete( Node **prev_node,int pos);
void PrintList(Node * next);




void InsertAtBeginning( Node **head_ref,int new_data)
```

```
{
Node *new_node=(struct Node*)malloc(sizeof( Node));
new_node->data=new_data;
new_node->next=*head_ref;
*head_ref=new_node;
}

void DeleteAtBeginning( Node **head_ref)
{
Node *ptr;
if(head_ref == NULL)
{
printf("\nList is empty");
}
else
{
ptr = *head_ref;
*head_ref = ptr->next;
free(ptr);
printf("\n Node deleted from the beginning ...");

}

}


void DeleteAtEnd(Node **head_ref)
{
Node *ptr,*ptr1;
```

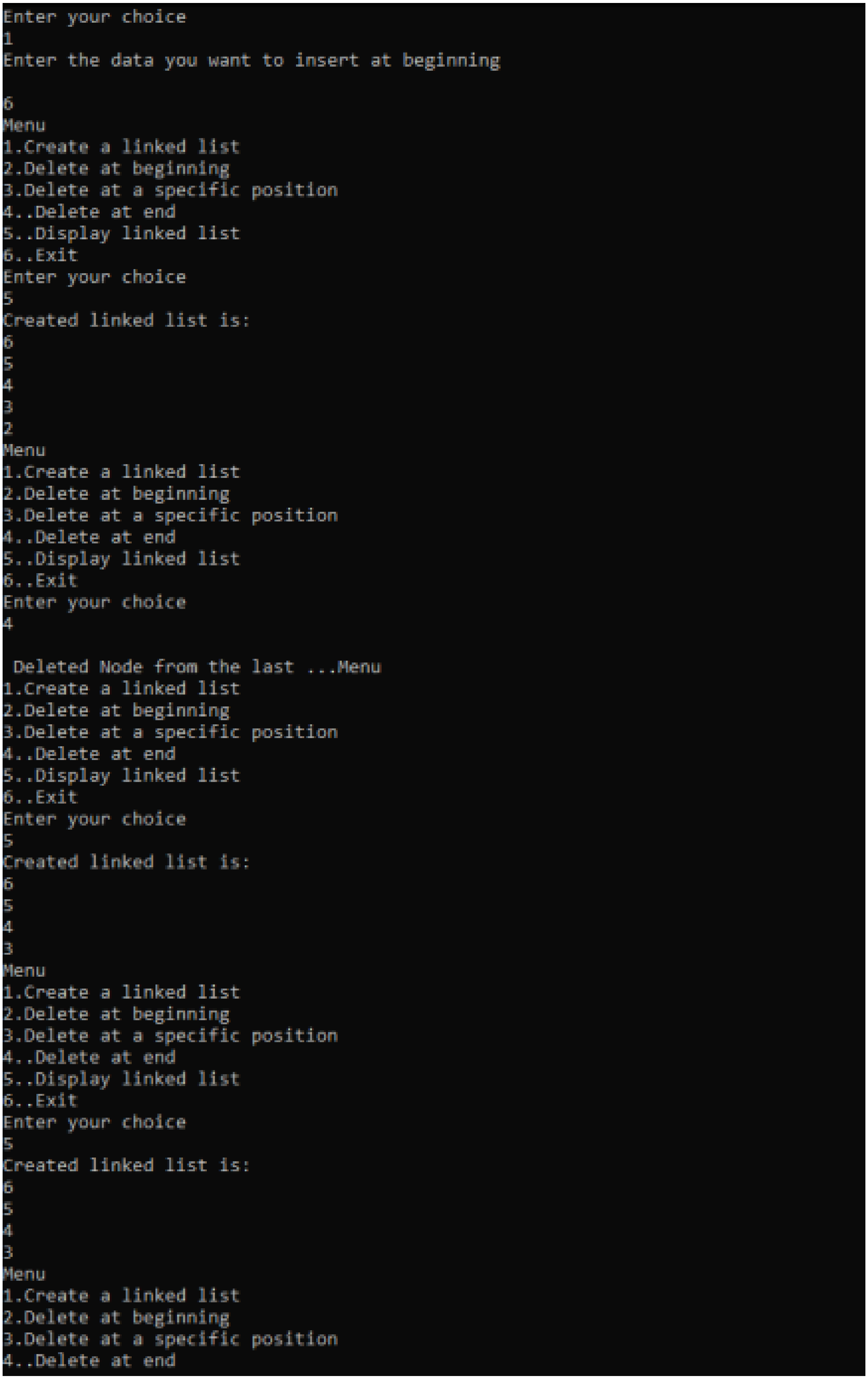
```
if(*head_ref == NULL)
{
printf("\nlist is empty");
}
else if((*head_ref)-> next == NULL)
{
free(*head_ref);
*head_ref= NULL;
printf("\nOnly node of the list deleted ...");
}
else
{
ptr = *head_ref;
while(ptr->next != NULL)
{
ptr1 = ptr;
ptr = ptr ->next;

}

ptr1->next = NULL;

free(ptr);

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

}

}
void Delete(Node **head_ref, int pos)
{
Node *temp = *head_ref, *prev;

if (temp == NULL)
{
printf("\nList is empty");
return;
}

if (pos == 1)
{
*head_ref = temp->next;
free(temp);
printf("\nDeleted node with position %d", pos);
return;
}



for (int i = 0; temp != NULL && i < pos - 1; i++)
```

```
{
prev = temp;
temp = temp->next;
}

if (temp == NULL)
{
printf("\nPosition out of range");
return;
}

prev->next = temp->next;
free(temp);
printf("\nDeleted node with position %d", pos);
}
void PrintList(Node *node)
{
while (node!=NULL)
{
printf("%d\n",node->data);
node=node->next;
}
}


int main()
{
int ch,new,pos;
Node* head=NULL;
while(ch!=6)
{
printf("Menu\n");
printf("1.Create a linked list\n");
printf("2.Delete at beginning\n");
printf("3.Delete at a specific position\n");
printf("4..Delete at end\n");
printf("5..Display linked list\n");
printf("6..Exit\n");
printf("Enter your choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1:
{
printf("Enter the data you want to insert at beginning\n");
scanf("%d",&new);
InsertAtBeginning(&head,new);
break;
}
case 2:
{
DeleteAtBeginning(&head);
break;
}
case 3:
{
printf("Enter the position at which you want to delete \n");
scanf("%d",&pos);
Delete(&head,pos);
break;

}
case4:
{
DeleteAtEnd(&head);
break;
}
case5:
{
printf("Created linked list is:\n");
PrintList(head);
break;
}
case6:
{
return0;
break;
}
default:
{
printf("Invaliddata!");
break;
}
}
}
return0;
}
```

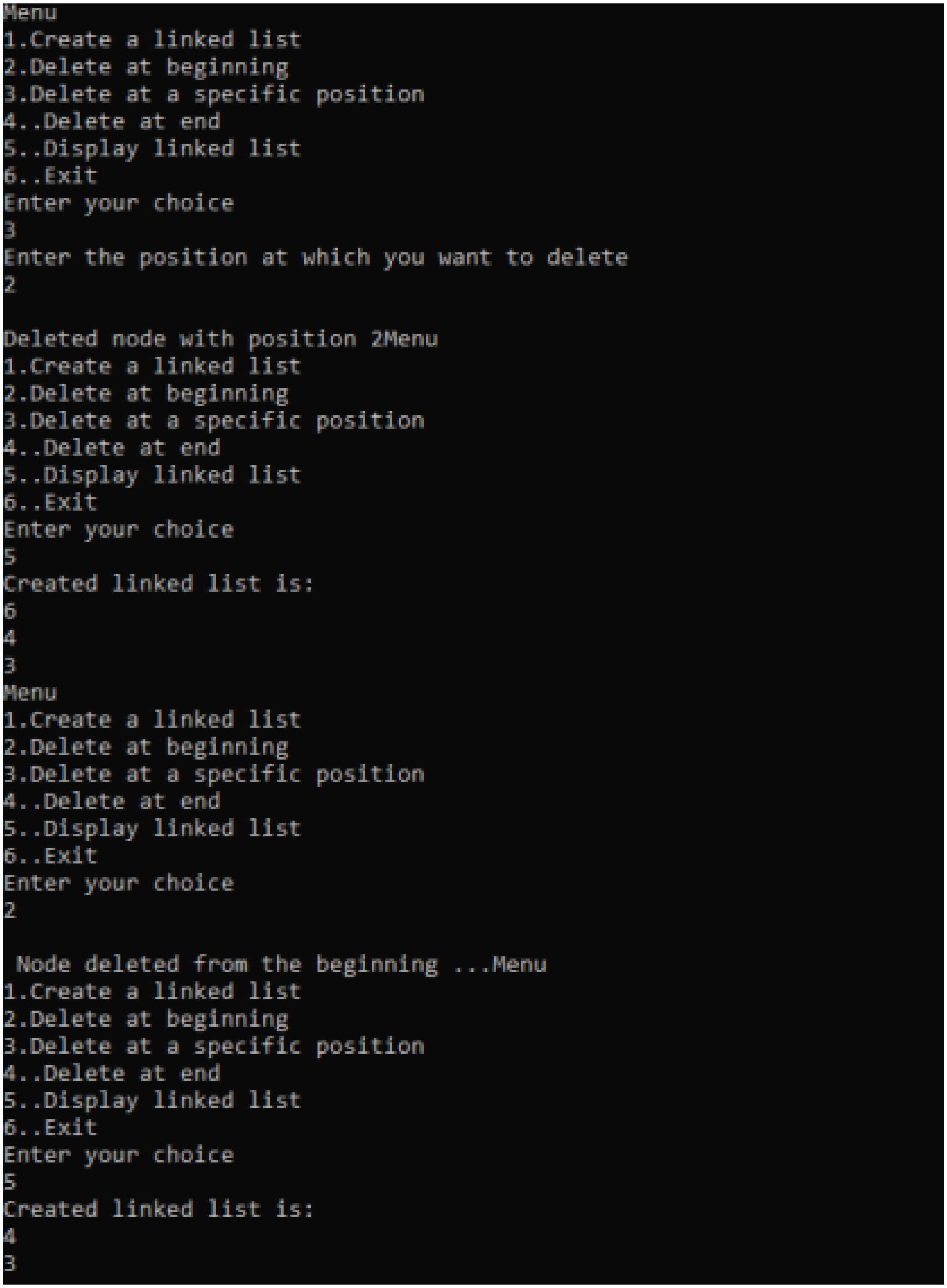
OUTPUT:



```
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
2
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
3
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
4
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
5
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning

6
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
```



```
Enter your choice
1
Enter the data you want to insert at beginning

6
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
6
5
4
3
2
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
4

 Deleted Node from the last ...Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
6
5
4
3
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
6
5
4
3
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
```



```
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
3
Enter the position at which you want to delete
2

Deleted node with position 2Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
6
4
3
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
2

 Node deleted from the beginning ...Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
4
3
```

LAB 6

6) a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

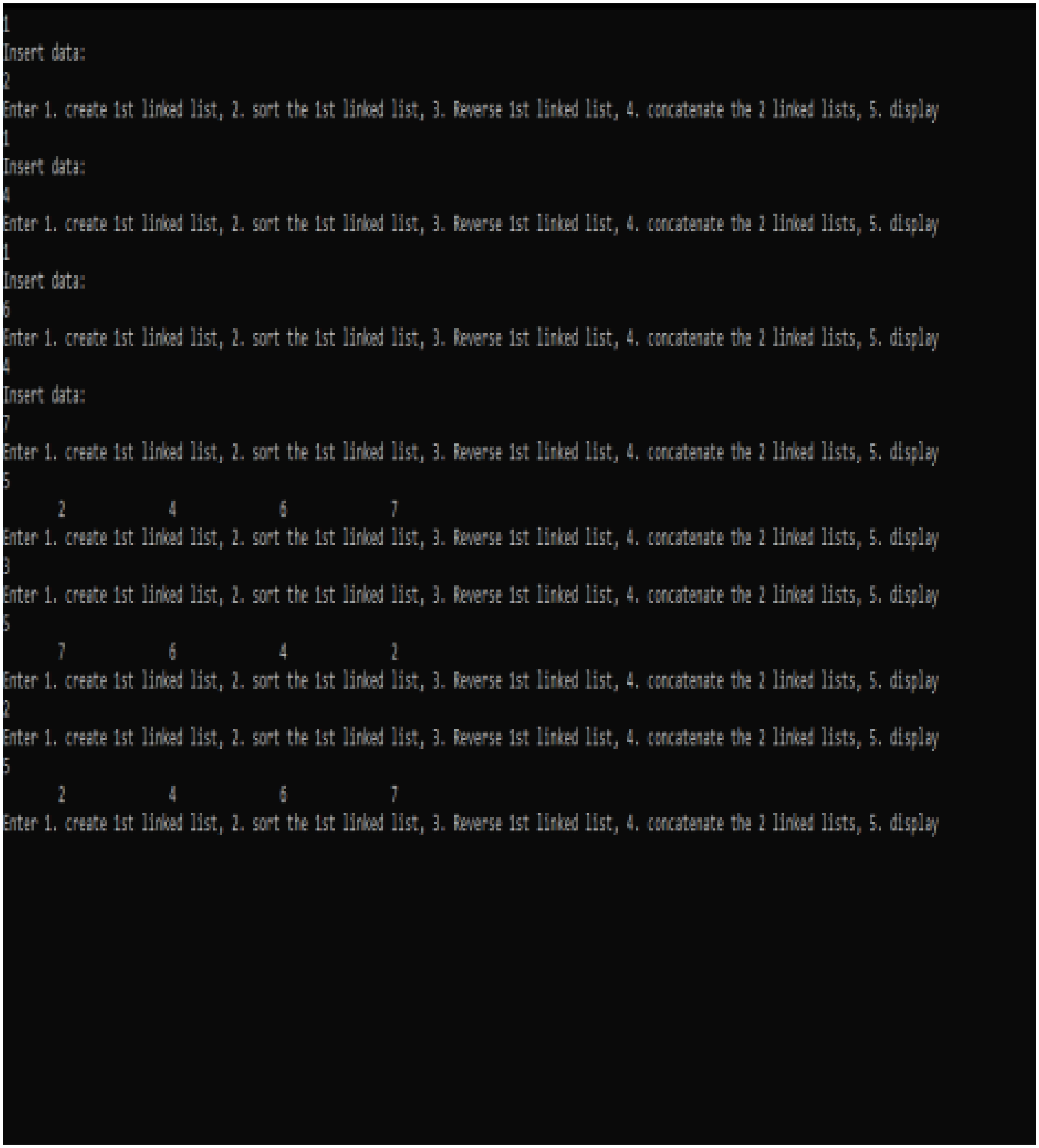
b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

a)

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node* next;
};
struct node* head=NULL;
struct node* head2=NULL;
void insert(int item)
{
   struct node* newnode=(struct node*)malloc(sizeof(struct node));
   newnode->data=item;
   if(head==NULL)
   {
      head=newnode;
      newnode->next=NULL;
   }
   else
   {
      struct node* temp=head;
      while(temp->next!=NULL)
      {
         temp=temp->next;
      }
      temp->next=newnode;
      newnode->next=NULL;
   }
}
void insert2(int item)
{
   struct node* newnode=(struct node*)malloc(sizeof(struct node));
   newnode->data=item;
   if(head2==NULL)
   {
      head2=newnode;
      newnode->next=NULL;
   }
   else
   {
      struct node* temp=head2;
      while(temp->next!=NULL)
      {
         temp=temp->next;
      }
      temp->next=newnode;
      newnode->next=NULL;
   }
}
void reverse()
{
   struct node* curr=head;
   struct node* prev=NULL;
   struct node* next=NULL;
   while(curr != NULL)
   {
      next=curr->next;
      curr->next=prev;
      prev=curr;
      curr=next;
   }
   head=prev;
}
void sort()
{
   struct node* i;
   struct node* j;
   for(i=head;i!=NULL;i=i->next)
   {
      for(j=i->next;j!=NULL;j=j->next)
      {
         if(i->data>j->data)
         {
            int temp=i->data;
            i->data=j->data;
            j->data=temp;
         }
      }
   }
}
void concatenate()
{
   if(head==NULL)
   {
      head=head2;
   }
   struct node* temp=head;
   while(temp->next!=NULL)
   {
            temp=temp->next;
   }
   temp->next=head2;
}
```

```
void display()
{
   struct node* temp=head;
   while(temp!=NULL)
   {
      printf("%d ->",temp->data);
      temp=temp->next;
   }
   printf("\n");
}
void main()
{
   int choice=0;
   int item=0;
   int c=0;
   int i=0;
   while(choice!=6)
  {
   printf(" Enter 1. to insert\n Enter 2 to reverse \n Enter 3 to sort \n Enter 4 to concatenate \n
Enter 5 to display\n Enter 6 to exit\n");
   scanf("%d",&choice);
   switch(choice)
   {
   case 1:
      printf("Enter data to insert\n");
      scanf("%d",&item);
      insert(item);
      break;
   case 2:
      reverse();
      break;
   case 3:
      sort();
      break;
   case 4:
    while(c!=2)
    {
         printf("For the second link list Enter 1. to insert\n enter 2 to exit\n");
        scanf("%d",&c);
        if(c==1)
        {
            printf("Enter data to insert\n");
            scanf("%d",&i);
            insert2(i);
        }
    }
    concatenate();
    break;
  case 5:
    display();
    break;
  case 6:
    break;
  }
}
```

OUTPUT:

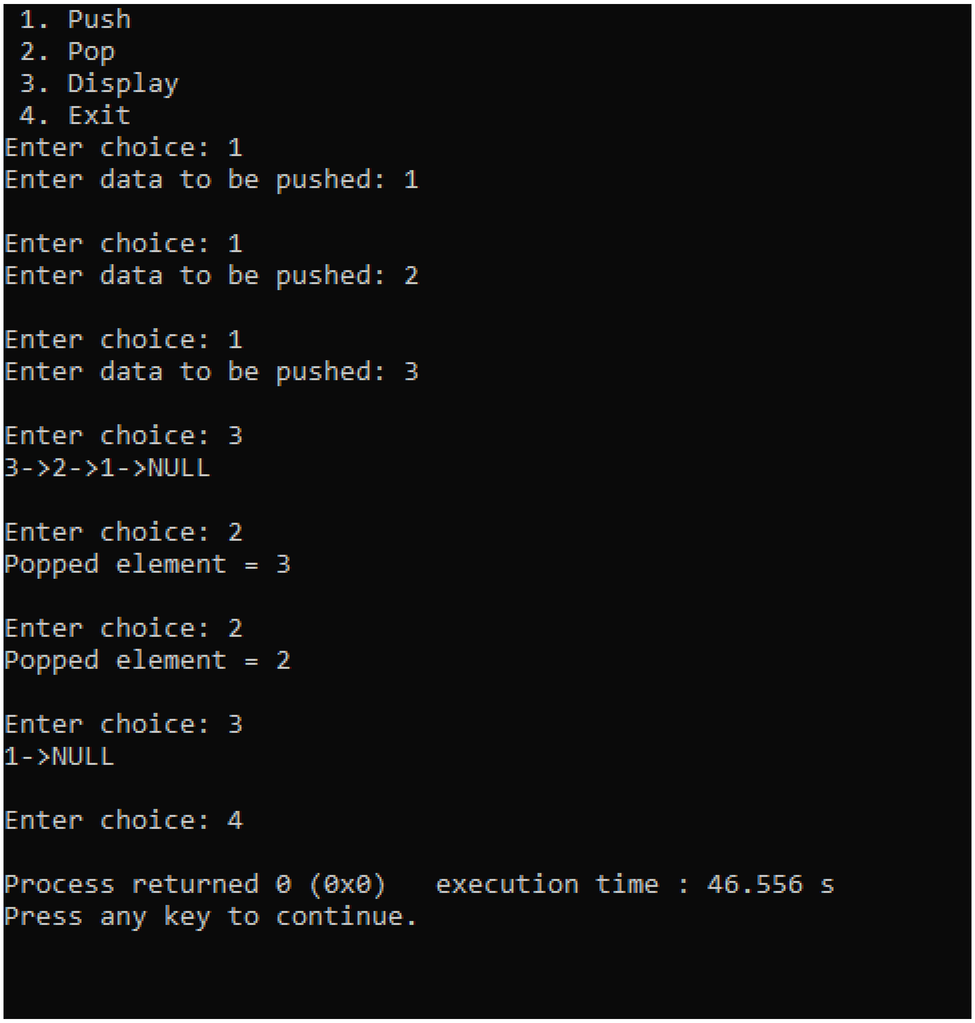


```
1
Insert data:
2
Enter 1. create 1st linked list, 2. sort the 1st linked list, 3. Reverse 1st linked list, 4. concatenate the 2 linked lists, 5. display
1
Insert data:
4
Enter 1. create 1st linked list, 2. sort the 1st linked list, 3. Reverse 1st linked list, 4. concatenate the 2 linked lists, 5. display
1
Insert data:
6
Enter 1. create 1st linked list, 2. sort the 1st linked list, 3. Reverse 1st linked list, 4. concatenate the 2 linked lists, 5. display
4
Insert data:
7
Enter 1. create 1st linked list, 2. sort the 1st linked list, 3. Reverse 1st linked list, 4. concatenate the 2 linked lists, 5. display
5
        2               4               6               7
Enter 1. create 1st linked list, 2. sort the 1st linked list, 3. Reverse 1st linked list, 4. concatenate the 2 linked lists, 5. display
3
Enter 1. create 1st linked list, 2. sort the 1st linked list, 3. Reverse 1st linked list, 4. concatenate the 2 linked lists, 5. display
5
        7               6               4               2
Enter 1. create 1st linked list, 2. sort the 1st linked list, 3. Reverse 1st linked list, 4. concatenate the 2 linked lists, 5. display
2
Enter 1. create 1st linked list, 2. sort the 1st linked list, 3. Reverse 1st linked list, 4. concatenate the 2 linked lists, 5. display
5
        2               4               6               7
Enter 1. create 1st linked list, 2. sort the 1st linked list, 3. Reverse 1st linked list, 4. concatenate the 2 linked lists, 5. display
```

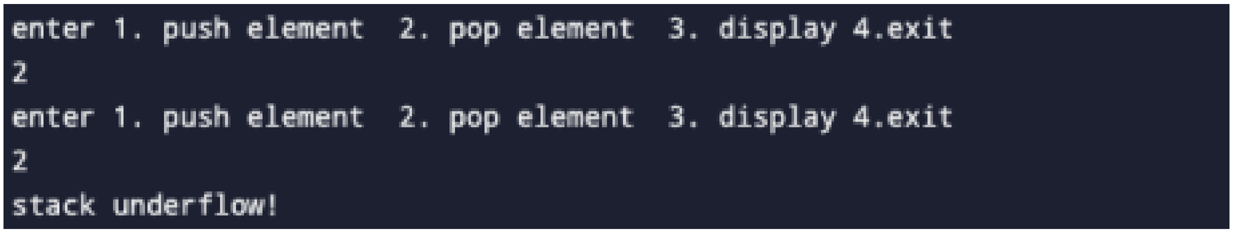
b)

```
#include <stdio.h>
#include <stdlib.h>

struct node{
    int data;
    struct node *next;
};
struct node *head, *temp, *newnode, *p;

void push(){
   newnode=(struct node *)malloc(sizeof(struct node));
   printf("enter data:");
  scanf("%d",&newnode->data);
   if(head==NULL){
       head=temp=newnode;
   }
  else{
       newnode->next=temp;
       head=newnode;
      temp=newnode;
   }
}
void pop(){
    if(head==NULL){
       printf("stack underflow!\n");
    }
    else{




    p=head;
    head=head->next;
    p->next=0;
    free(p);
    }
}
void display(){
    temp=head;
    while(temp!=NULL){
        printf("%d\n",temp->data);
        temp=temp->next;
    }
    temp=head=newnode;
}

int main(){
    head=NULL;
    int c;
    while(1){
    printf("enter 1. push element 2. pop element 3. display 4.exit\n");
    scanf("%d",&c);

    switch(c){
        case 1: push();
                break;
        case 2: pop();
                break;
        case 3: display();
                break;
        case 4: exit(1);
    }
  }
}
```

OUTPUT:



```
 1. Push
 2. Pop
 3. Display
 4. Exit
Enter choice: 1
Enter data to be pushed: 1

Enter choice: 1
Enter data to be pushed: 2

Enter choice: 1
Enter data to be pushed: 3

Enter choice: 3
3->2->1->NULL

Enter choice: 2
Popped element = 3

Enter choice: 2
Popped element = 2

Enter choice: 3
1->NULL

Enter choice: 4

Process returned 0 (0x0)   execution time : 46.556 s
Press any key to continue.
```



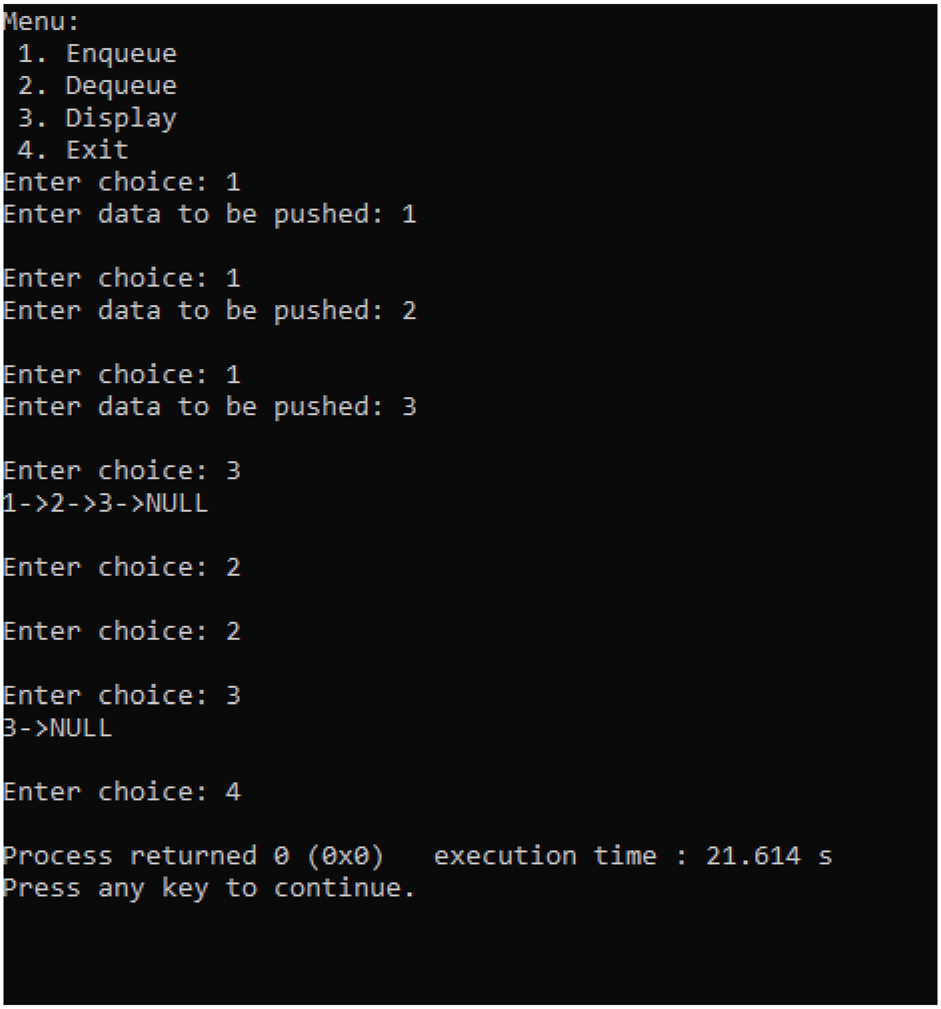
```
enter 1. push element  2. pop element  3. display 4.exit
2
enter 1. push element  2. pop element  3. display 4.exit
2
stack underflow!
```

QUEUE

```
#include <stdio.h>
#include <stdlib.h>

struct node{
   int data;
   struct node *next;
};
struct node *front, *rear, *newnode, *temp, *p;

void enqueue(){
  newnode=(struct node *)malloc(sizeof(struct node));
  printf("enter data:");
  scanf("%d",&newnode->data);
  if(front==NULL && rear==NULL){
     front=rear=newnode;
  }
  else{
     rear->next=newnode; //O(1)
     rear=rear->next; //rear=newnode;
  }
}
void dequeue(){ //delete from beginning
   if(front==NULL){
      printf("queue underflow\n");
   }
   else{
   printf("dequeued element: %d\n",front->data);
   p=front;
   front=front->next;
   p->next=NULL;
   free(p);




   }
}
void display(){
   temp=front; //temp pointer to traverse and display
   if(rear==0 && front==0){
      printf("Queue is empty\n");
   }
   else{
      while(temp!=NULL){
          printf("%d\n",temp->data);
          temp=temp->next;
      }
   }
}

int main(){
   front=NULL;
   rear=NULL; //tail
   int c;
   while(1){
   printf("enter 1. enqueue 2. dequeue 3. display 4.exit\n");
   scanf("%d",&c);

   switch(c){
      case 1: enqueue();
```

```
                break;
        case 2: dequeue();
                break;
        case 3: display();
                break;
        case 4: exit(1);




}
}
}
```

OUTPUT:



```
Menu:
 1. Enqueue
 2. Dequeue
 3. Display
 4. Exit
Enter choice: 1
Enter data to be pushed: 1

Enter choice: 1
Enter data to be pushed: 2

Enter choice: 1
Enter data to be pushed: 3

Enter choice: 3
1->2->3->NULL

Enter choice: 2

Enter choice: 2

Enter choice: 3
3->NULL

Enter choice: 4

Process returned 0 (0x0)   execution time : 21.614 s
Press any key to continue.
```

LAB 7

7) a) 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

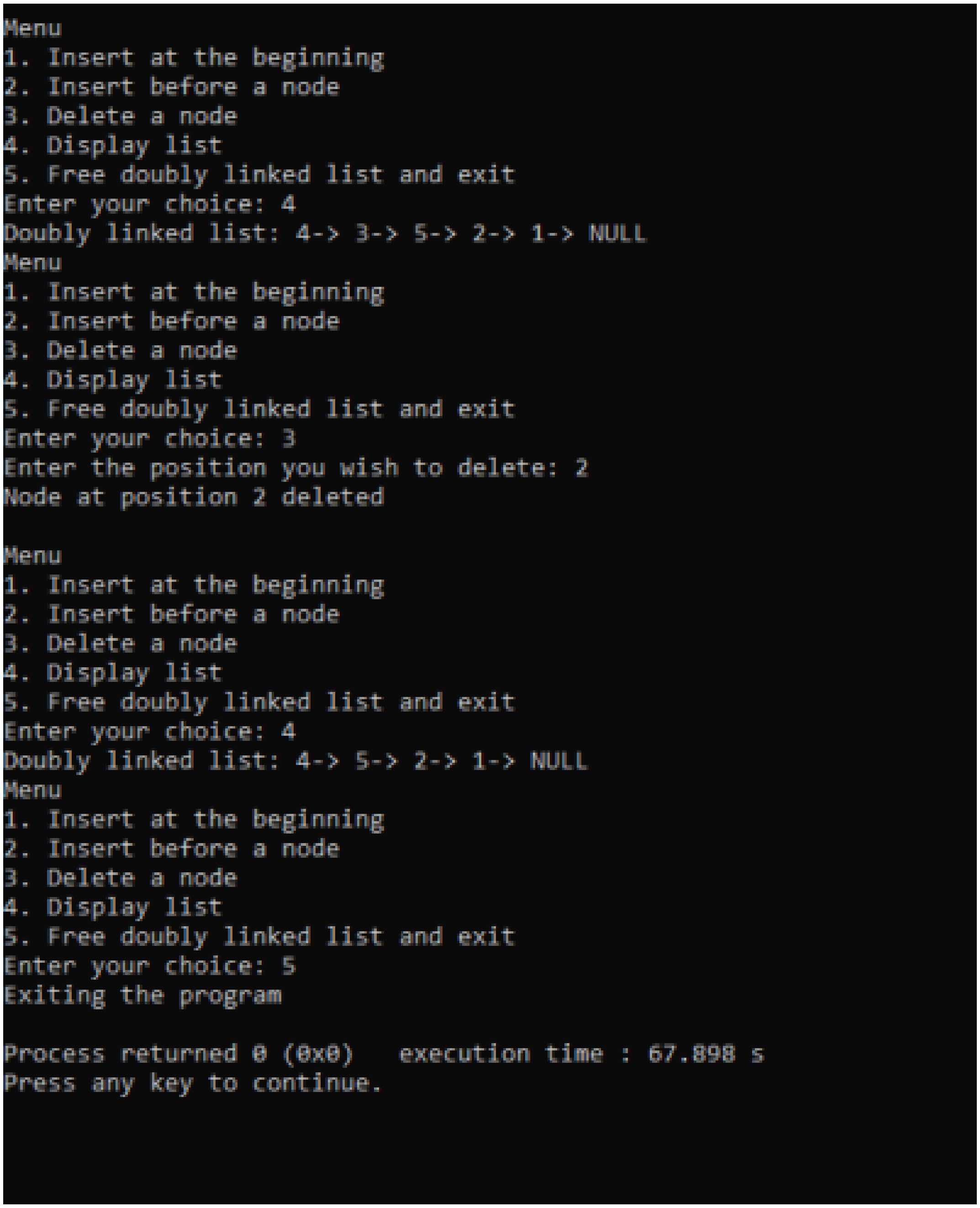
Display the contents of the list

```
#include <stdio.h>
#include <stdlib.h>

struct node{
   int data;
   struct node *next;
   struct node *prev;
};

struct node *head, *temp, *p, *f, *ptr,*newnode;

void create(){
   newnode=(struct node*)malloc(sizeof(struct node));
   printf("enter data:\n");
   scanf("%d",&newnode->data);
   if(head==NULL){
      head=temp=newnode;
      temp->prev=NULL;
```

```
        temp->next=NULL;
    }
    else{
        temp->next=newnode;
        newnode->prev=temp;
        temp=temp->next;
    }
}

void insertLeft(){
    temp=head;
    int pos;
    printf("enter position of node to insert to the left:\n");
    scanf("%d",&pos);
    int i=1;
    if(pos==1){
        newnode=(struct node*)malloc(sizeof(struct node));
        printf("enter data:");
        scanf("%d",&newnode->data);
        newnode->next=temp;
        head=newnode;
        newnode->prev=NULL;
    }
    else{
        while(i<pos){
            p=temp;
            temp=temp->next;
            i++;
        }
        newnode=(struct node*)malloc(sizeof(struct node));
        printf("enter data:\n");

        scanf("%d",&newnode->data);
        newnode->next=temp;
        p->next=newnode;
        newnode->prev=p;
    }
}

voiddelete(){
    temp=head;
    f=temp;
    int val;
    printf("enterthevaluetobedeleted:\n");
    scanf("%d",&val);
    while(temp!=NULL){
        if(val==temp->data){
            if(temp==head){
                temp=temp->next;
                head=temp;
                f->next=NULL;
                free(f);
            }
            else if(temp->next==NULL){
                f=temp;
                temp->prev=NULL;
                free(f);
            }
            else{
                f->next=temp->next;
                temp->next->prev=f;
                temp->next=NULL;
                temp->prev=NULL;




                ptr=temp;
                free(ptr);
            }
        }
        else{
            f=temp;
            temp=temp->next;
        }
    }
}

void display(){
    temp=head;
    while(temp!=NULL){
        printf("\t%d\t",temp->data);
        temp=temp->next;
    }
}

void main(){
    head=NULL;
    while(1){
        printf("enter 1. create a doubly linked list, 2. insert new node to the left, 3.
delete the node based on a specific value, 4. display\n");

        int choice;
        scanf("%d",&choice);
        switch(choice){
```

```
      case 1: create();
          break;
      case 2: insertLeft();




          break;
      case 3: delete();
          break;
      case 4: display();
          break;
      default: exit(1);
    }
  }
}
```

OUTPUT:

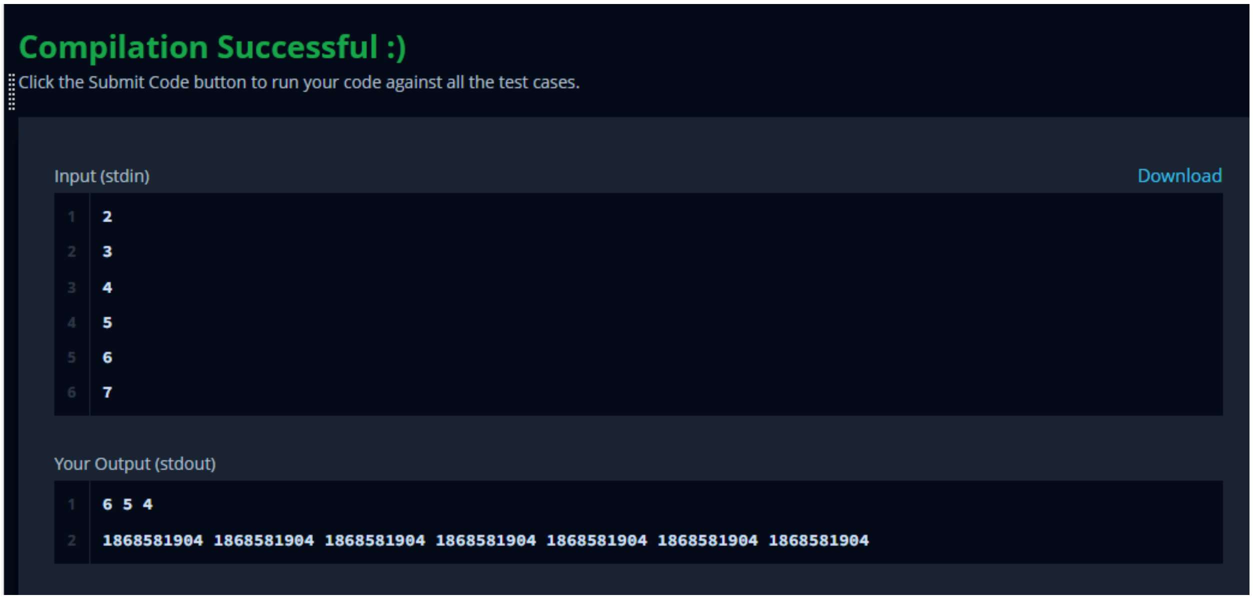


```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 4
Doubly linked list: 4-> 3-> 5-> 2-> 1-> NULL
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 3
Enter the position you wish to delete: 2
Node at position 2 deleted

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 4
Doubly linked list: 4-> 5-> 2-> 1-> NULL
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 5
Exiting the program

Process returned 0 (0x0)   execution time : 67.898 s
Press any key to continue.
```

7) b) Hackerrank Question- Reverse a doubly linked list

```
DoublyLinkedListNode*
    reverse(DoublyLinkedListNode* llist) {
    DoublyLinkedListNode* temp = llist;
    DoublyLinkedListNode* curr = temp;
    DoublyLinkedListNode* prev = NULL;
    DoublyLinkedListNode* nextOne = NULL;
```

```
while(curr != NULL) {
nextOne = curr->next;
curr->next = prev;
prev = curr;
curr = nextOne;
}
return prev;
}
```



Compilation Successful :)

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

```
2
3
4
5
6
7
```

Your Output (stdout)

```
6 5 4
1868581904 1868581904 1868581904 1868581904 1868581904 1868581904 1868581904
```

LAB 8

8 a)

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

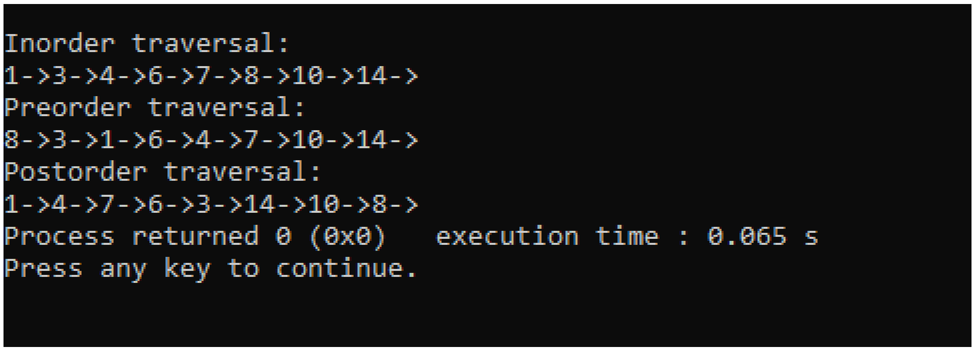
c)To display the elements in the tree.

a)

```
 #include <stdio.h>
#include <stdlib.h>

struct node {
  int data;
  struct node *left, *right;
};

// Create a node
struct node *newNode(int item) {
  struct node *temp = (struct node *)malloc(sizeof(struct node));
  temp->data = item;
  temp->left = temp->right = NULL;
  return temp;
```

```
}

// InorderTraversal
void inorder(structnode*root){
  if (root != NULL) {
    // Traverse left
    inorder(root->left);

    // Traverse root
    printf("%d -> ", root->data);

    // Traverse right
    inorder(root->right);
  }
}

//PreorderTraversal
void preorder(structnode*root){
  if (root != NULL) {
    // Traverse root
    printf("%d -> ", root->data);
    // Traverse left
    preorder(root->left);
    // Traverse right
    preorder(root->right);
  }
}

//PostorderTraversal
void postorder(structnode*root){
  if (root != NULL) {




    // Traverse left
    postorder(root->left);
    // Traverse right
    postorder(root->right);
    // Traverse root
    printf("%d -> ", root->data);
  }
}

// Insert a node
struct node *insert(struct node *node, int data) {
  // Return a new node if the tree is empty
  if (node == NULL) return newNode(data);

  // Traverse to the right place and insert the node
  if (data < node->data)
    node->left = insert(node->left, data);
  else
    node->right = insert(node->right, data);

  return node;
}


// Driver code
int main() {
  struct node *root = NULL;
  root = insert(root, 9);
  root = insert(root, 1);
  root = insert(root, 2);
  root = insert(root, 5);




root = insert(root, 22);
root = insert(root, 11);
root = insert(root, 14);
root = insert(root, 4);

printf("\nInorder traversal: \n");
inorder(root);

printf("\nPreorder traversal: \n");
preorder(root);

printf("\nPostorder traversal: \n");
postorder(root);

}
```

OUTPUT:

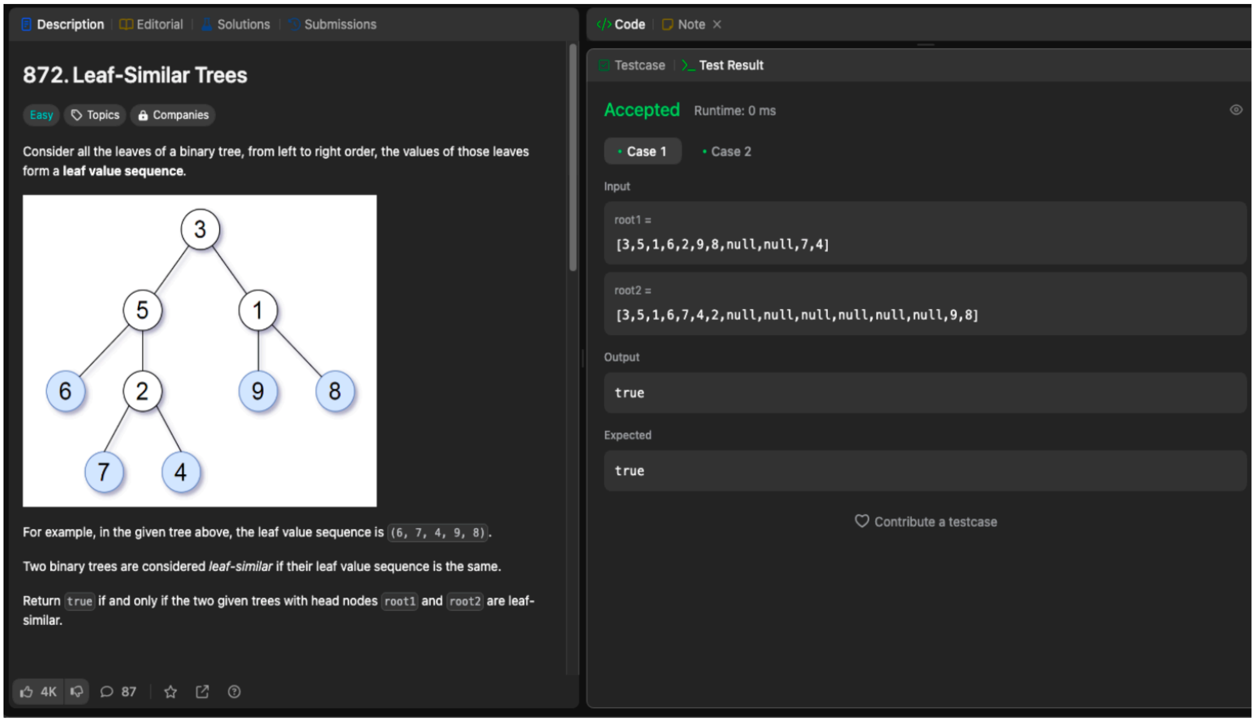


```
Inorder traversal:
1->3->4->6->7->8->10->14->
Preorder traversal:
8->3->1->6->4->7->10->14->
Postorder traversal:
1->4->7->6->3->14->10->8->
Process returned 0 (0x0)   execution time : 0.065 s
Press any key to continue.
```

8 b) Leetcode Question - Leaf-Similar Trees

```
void findLeaves(struct TreeNode* node, int** leafValues, int* size, int*
capacity) {
    if (node == NULL) {
        return;
    }

    if (node->left == NULL && node->right == NULL) {
        if (*size >= *capacity) {
            *capacity *= 2;
            *leafValues = (int*) realloc(*leafValues, *capacity * sizeof(int)); }
        (*leafValues)[(*size)++] = node->val;
    }

    findLeaves(node->left, leafValues, size, capacity);
    findLeaves(node->right, leafValues, size, capacity);
}

bool leafSimilar(struct TreeNode* root1, struct TreeNode* root2) {
    int *leaves1 = (int*) malloc(sizeof(int) * 10);
    int size1 = 0, capacity1 = 10;





    int *leaves2 = (int*) malloc(sizeof(int) * 10);
    int size2 = 0, capacity2 = 10;

    findLeaves(root1, &leaves1, &size1, &capacity1);
    findLeaves(root2, &leaves2, &size2, &capacity2);

    if (size1 != size2) {
        free(leaves1);
        free(leaves2);
        return false;
    }

    for (int i = 0; i < size1; i++) {
        if (leaves1[i] != leaves2[i]) {
            free(leaves1);
            free(leaves2);
            return false;
        }
    }

    free(leaves1);
    free(leaves2);
    return true;
}
```

OUTPUT:



872. Leaf-Similar Trees

Consider all the leaves of a binary tree, from left to right order, the values of those leaves form a **leaf value sequence**.

For example, in the given tree above, the leaf value sequence is `(6, 7, 4, 9, 8)`.

Two binary trees are considered *leaf-similar* if their leaf value sequence is the same.

Return `true` if and only if the two given trees with head nodes `root1` and `root2` are leaf-similar.

Accepted Runtime: 0 ms

Case 1 Case 2

Input

```
root1 =
[3,5,1,6,2,9,8,null,null,7,4]
root2 =
[3,5,1,6,7,4,2,null,null,null,null,null,null,9,8]
```

Output

```
true
```

Expected

```
true
```

LAB 9

9) a) Write a program to traverse a graph using BFS method. b) Write a program to check whether given graph is connected or not using DFS method.

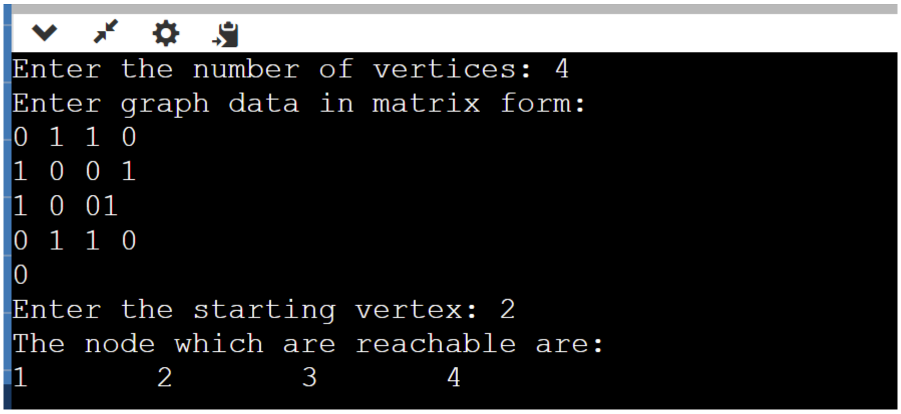
BFS

```
#include <stdio.h>
 int n, i, j, visited[10], queue[10], front = -1, rear = -1;
int adj[10][10];

void bfs(int v)
```

```
{
    for (i = 1; i <= n; i++)
        if (adj[v][i] && !visited[i])
            queue[++rear] = i;
    if (front <= rear)
    {
        visited[queue[front]] = 1;
        bfs(queue[front++]);
    }



}

void main()
{
    int v;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
    {
        queue[i] = 0;
        visited[i] = 0;
    }
    printf("Enter graph data in matrix form: \n");
    for (i = 1; i <= n; i++)
```

```
        for (j = 1; j <= n; j++)
            scanf("%d", &adj[i][j]);
    printf("Enter the starting vertex: ");
    scanf("%d", &v);
    bfs(v);
    printf("The node which are reachable are: \n");
    for (i = 1; i <= n; i++)
        if (visited[i])
            printf("%d\t", i);



        else
            printf("BFS is not possible. Not all nodes are reachable");

}
```

OUTPUT:

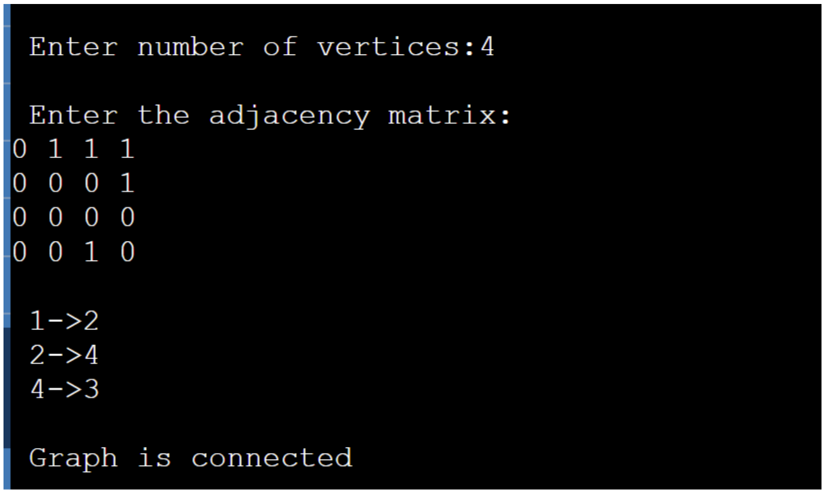


```
Enter the number of vertices: 4
Enter graph data in matrix form:
0 1 1 0
1 0 0 1
1 0 01
0 1 1 0
0
Enter the starting vertex: 2
The node which are reachable are:
1       2       3       4
```

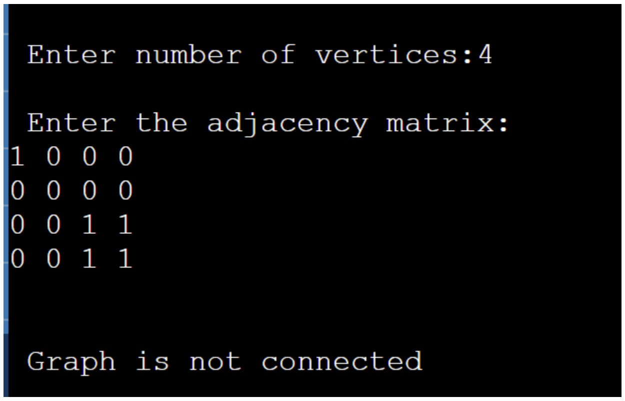
b) DFS

```
#include<stdio.h>

#include<conio.h>

int a[20][20], reach[20], n;

void dfs(int v) {

    int i;
    reach[v] = 1;
    for (i = 1; i <= n; i++)
        if (a[v][i] && !reach[i]) {
            printf("\n %d->%d", v, i);
            dfs(i);
        }
}

int main(int argc, char **argv) {
    int i, j, count = 0;
    printf("\n Enter number of vertices:");
    scanf("%d", &n);
    for (i = 1; i <= n; i++) {
        reach[i] = 0;
        for (j = 1; j <= n; j++)
            a[i][j] = 0;
    }
    printf("\n Enter the adjacency matrix:\n");
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            scanf("%d", &a[i][j]);
    dfs(1);
    printf("\n");




    for (i = 1; i <= n; i++) {
        if (reach[i])
            count++;
    }
    if (count == n)
        printf("\n Graph is connected");
    else
        printf("\n Graph is not connected");
    return 0;
}
```

OUTPUT:



```
 Enter number of vertices:4

 Enter the adjacency matrix:
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0

 1->2
 2->4
 4->3

 Graph is connected
```



```
 Enter number of vertices:4

 Enter the adjacency matrix:
1  0  0  0
0  0  0  0
0  0  1  1
0  0  1  1


 Graph is not connected
```