# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

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

**LAB REPORT**
**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**Prajwal P (1BM22CS200)**

**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 Prajwal P **(1BM22CS200)**, 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**
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

## Index Sheet

**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. |

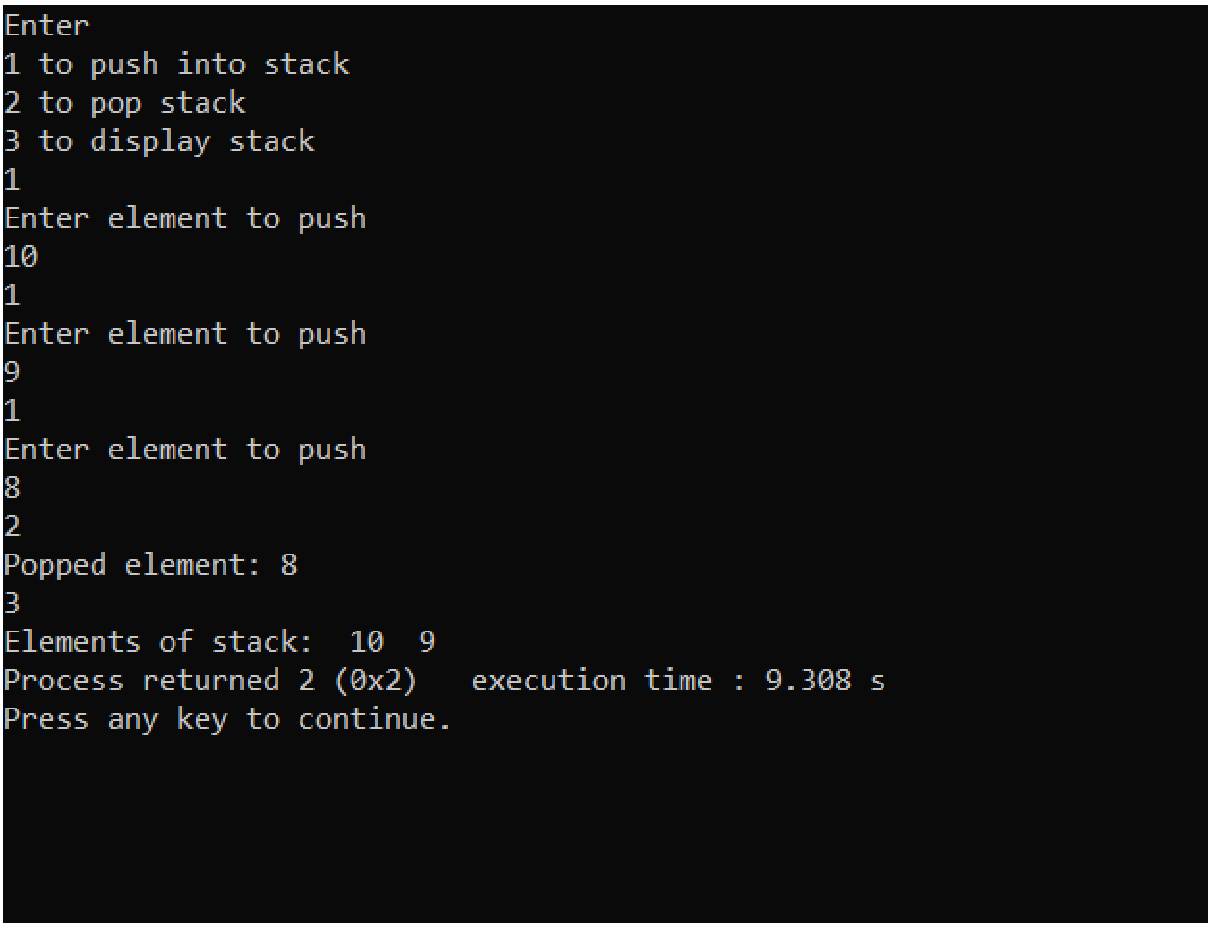
**Lab program 1:**

**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>
#include<stdlib.h>
#define max 10
void push(int a);
int pop();
void display(int a[]);
int num[max],top =-1;

void main(){
   int n,a;
   printf("Enter\n1 to push into stack\n2 to pop stack\n3 to display stack\n");
   scanf("%d",&n);
   while(n!=3){
      switch(n){
      case 1: printf("Enter element to push\n");
             scanf("%d",&a);
             push(a);
             break;
      case 2: a=pop();
             printf("Popped element: %d\n",a);
             break;
      default: printf("Wrong input");
      }
      scanf("%d",&n);
   }
   if(n==3){
      display(num);
   }
}
void push(int a){
   if(top>max-1){
      printf("Stack overflow");
      exit(0);
   }
   else{
      ++top;
      num[top]=a;
   }
}

int pop(){
   if(top==-1){
      printf("Stack underflow:");
```

```
        exit(0);
    }
    else{
        return num[top];
        --top;

    }
}

void display(int a[]){
    printf("Elements of stack: ");
    for(int i=0;i<top;i++){
        printf(" %d ",num[i]);
    }
}
```

**Output:**



```
Enter
1 to push into stack
2 to pop stack
3 to display stack
1
Enter element to push
10
1
Enter element to push
9
1
Enter element to push
8
2
Popped element: 8
3
Elements of stack:  10  9
Process returned 2 (0x2)   execution time : 9.308 s
Press any key to continue.
```

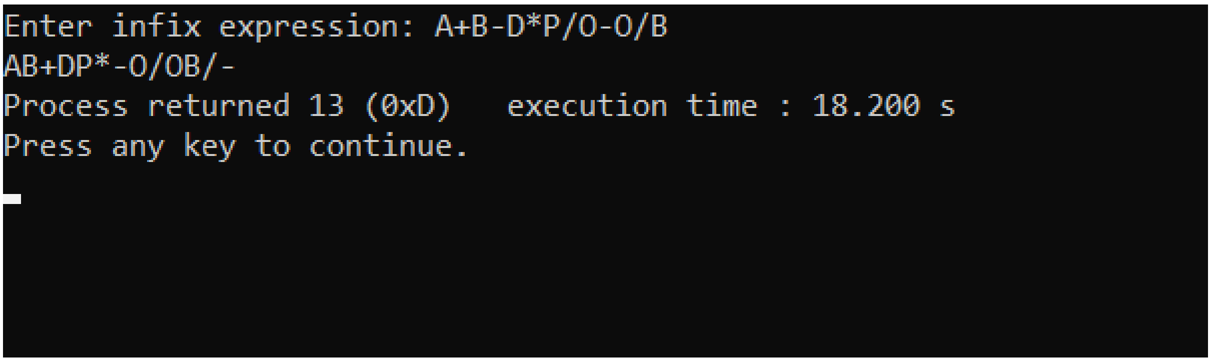
**Lab Program 2:**

**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<ctype.h>
#define max 20
void push(char a);
char pop();
char stack[max],top =-1;
int pre(char a);

void main(){
   char infix[max],a;
   char post[max];
   printf("Enter infix expression: ");
   scanf("%s",infix);
   int j=0;
   push('(');
   for(int i=0;i<strlen(infix);i++){
      if(isalnum(infix[i])){
         post[j]=infix[i];
         j+=1;
      }
     else if((infix[i]=='+' || infix[i]=='-' || infix[i]=='/' || infix[i]=='*')){
            if(pre(infix[i])>pre(stack[top])){
               push(infix[i]);
            }
            else if(pre(infix[i])<=pre(stack[top])){

               while(1){
               a=pop();
               if(a=='('){
                  push(a);
                  break;
               }
               post[j]=a;
               j+=1;
            }
            push(infix[i]);
         }


   }
}
while(top!=-1){
   char y=pop();
```

```
    if(y=='('){
        break;
       }
    post[j]=y;
    j+=1;


}
post[j]='\0';
printf("%s",post);

}

void push(char a){
    if(top>max-1){
        printf("Stack overflow");
        exit(0);
    }
    else{
        ++top;
        stack[top]=a;
    }
}

char pop(){
    if(top==-1){
        printf("Stack underflow:");
        exit(0);
    }
    else{
        return stack[top--];

    }
}

int pre(char a){
        if(a=='^'){
            return 3;
        }
        else if( a=='*' || a=='/'){
            return 2;
        }
        else if(a=='+' || a=='-'){
            return 1;
        }
        else{
            return 0;
        }
}
```

**Output:**



```
Enter infix expression: A+B-D*P/O-O/B
AB+DP*-O/OB/-
Process returned 13 (0xD)   execution time : 18.200 s
Press any key to continue.
```

**Lab Program 3**

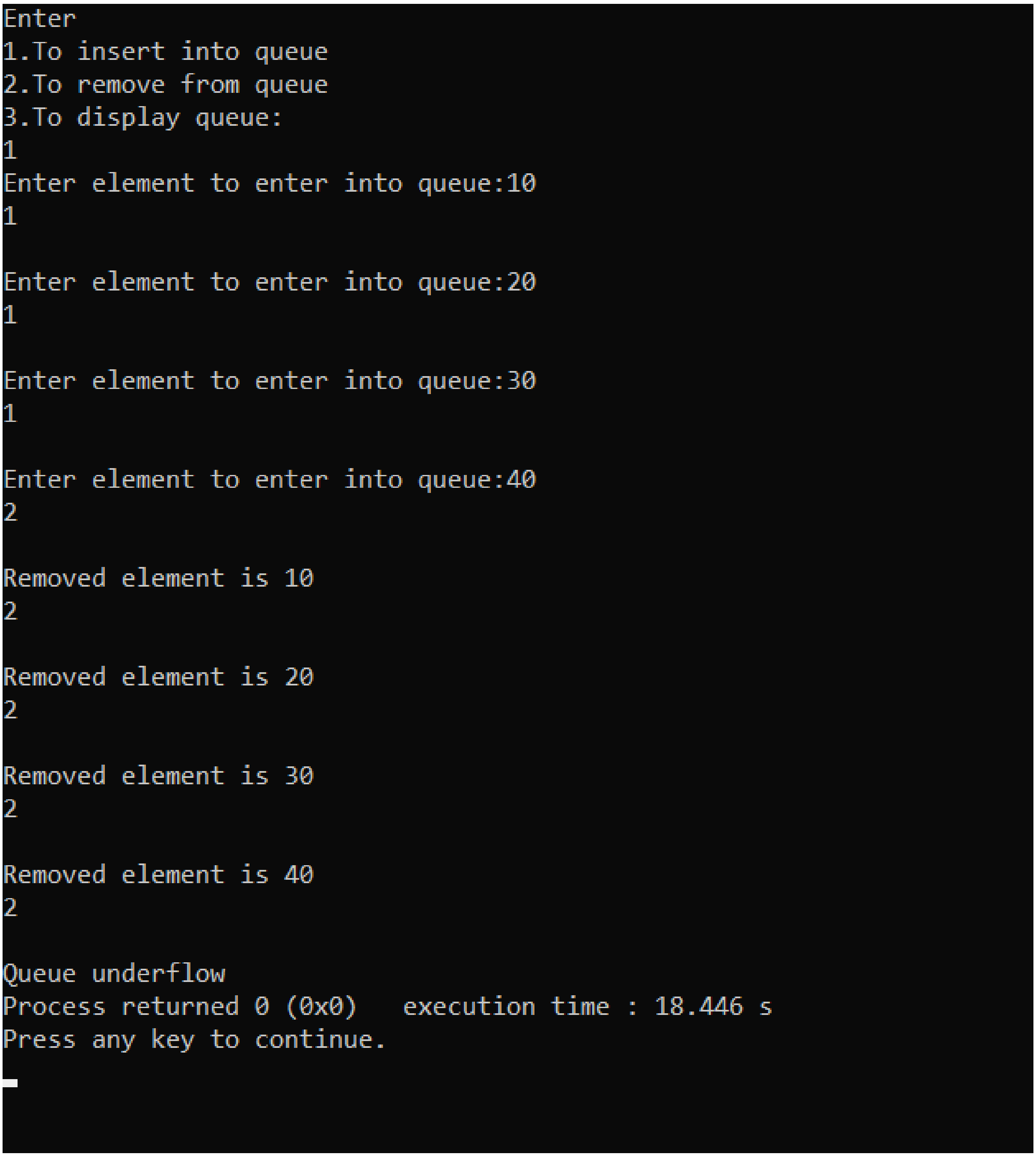
**3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions**

```
#include<stdio.h>
#include<stdlib.h>
#define max 5
int queue[max];
int front=-1, rear=-1;

void enqueue(int a);
int dequeue();
void display();

void main(){
   int n,m;
   printf("Enter\n1.To insert into queue\n2.To remove from queue\n3.To display queue:\n");
   scanf("%d",&n);
   while(n!=3){
      switch(n){
         case 1: printf("Enter element to enter into queue:");
               scanf("%d",&m);
               enqueue(m);
               break;

         case 2: m=dequeue();
               printf("Removed element is %d\n",m);
               break;

         default: printf("Invalid input");
      }
      scanf("%d",&n);
      printf("\n");
   }
   if(n==3){
```

```
            display();
        }
}

void enqueue(int a){
    if(rear==max-1){
        printf("Queue overflow");

    }
    rear+=1;
    queue[rear]=a;
}

int dequeue(){
    if(rear==-1 || front==rear){
        printf("Queue underflow");
        exit(0);
    }
    else{
        front+=1;
        return queue[front];
    }
}

void display(){

    printf("Queue:\n");
    for(int i=front+1;i<=rear;i++){
        printf("%d\t",queue[i]);
        }
}
```

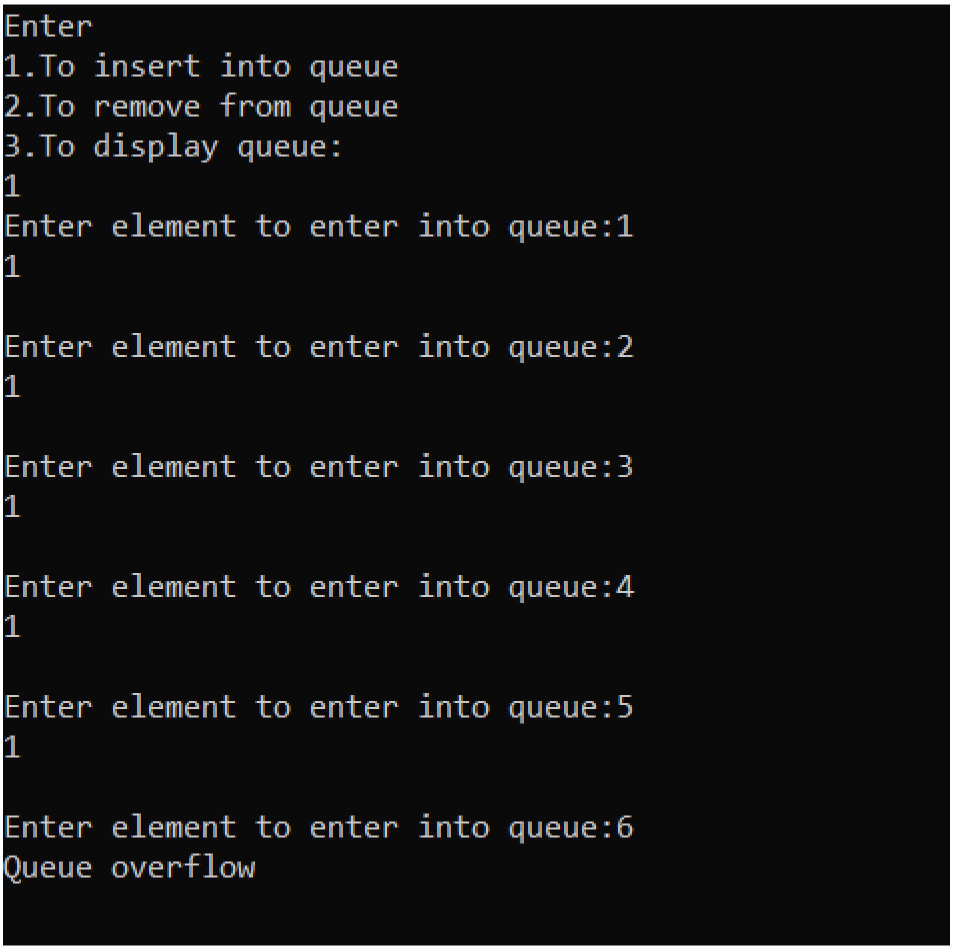
**Output:**

Queue Underflow



```
Enter
1.To insert into queue
2.To remove from queue
3.To display queue:
1
Enter element to enter into queue:10
1

Enter element to enter into queue:20
1

Enter element to enter into queue:30
1

Enter element to enter into queue:40
2

Removed element is 10
2

Removed element is 20
2

Removed element is 30
2

Removed element is 40
2

Queue underflow
Process returned 0 (0x0)   execution time : 18.446 s
Press any key to continue.
```

Queue Overflow



```
Enter
1.To insert into queue
2.To remove from queue
3.To display queue:
1
Enter element to enter into queue:1
1

Enter element to enter into queue:2
1

Enter element to enter into queue:3
1

Enter element to enter into queue:4
1

Enter element to enter into queue:5
1

Enter element to enter into queue:6
Queue overflow
```

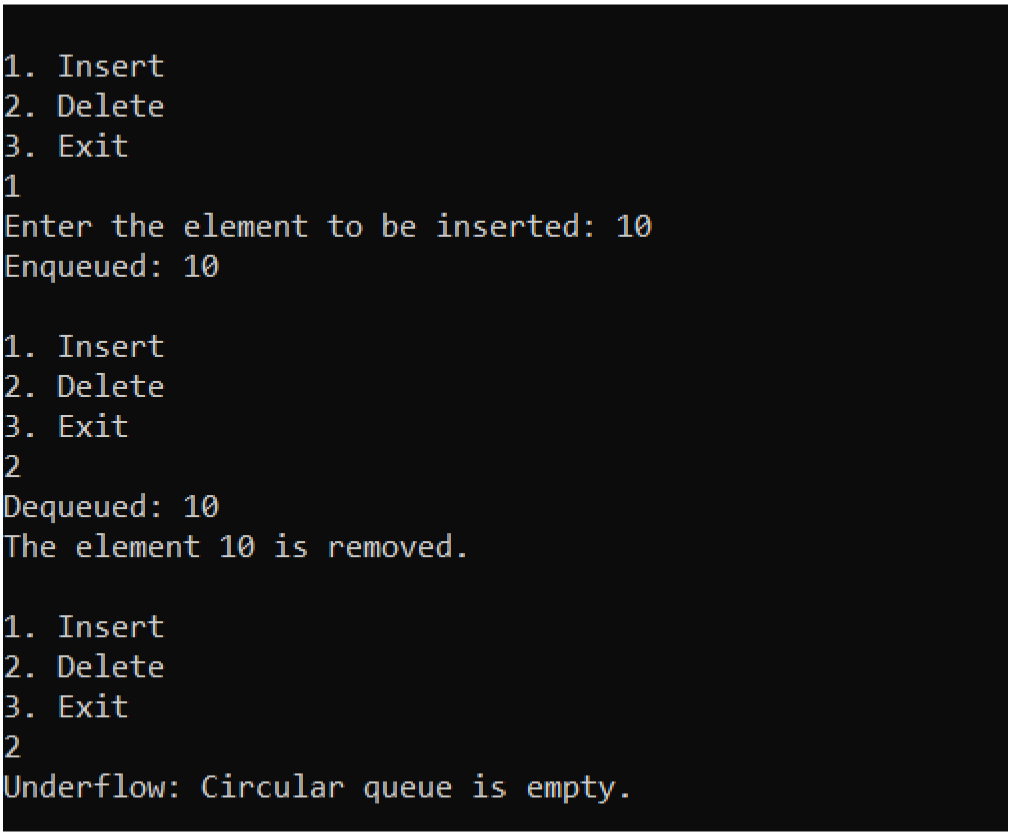
**WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions**

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

#define MAX 6

int cq[MAX];
int front = -1, rear = -1;

bool is_full() {
   return (rear + 1) % MAX == front;
}

bool is_empty() {
   return front == -1 && rear == -1;
```

```
}

void insert(int item) {
   if (is_full()) {
      printf("Overflow: Circular queue is full.\n");
      // Handle overflow appropriately, e.g., return without enqueueing
      return;
   }

   if (is_empty()) {
      front = rear = 0;
   } else {
      rear = (rear + 1) % MAX;
   }

   cq[rear] = item;
   printf("Enqueued: %d\n", item);
}

int dequeue() {
   if (is_empty()) {
      printf("Underflow: Circular queue is empty.\n");
      return -1;
   }

   int deletedItem = cq[front];

   if (front == rear) {
      front = rear = -1;
   } else {
      front = (front + 1) % MAX;
   }

   printf("Dequeued: %d\n", deletedItem);
   return deletedItem;
}

int main() {
   int n, ele;
   do {
      printf("\n1. Insert\n2. Delete\n3. Exit\n");
      scanf("%d", &n);
      switch (n) {
         case 1:
            printf("Enter the element to be inserted: ");
            scanf("%d", &ele);
            insert(ele);
            break;
         case 2:
            {
                int deletedItem = dequeue();
                if (deletedItem != -1) {
                    printf("The element %d is removed.\n", deletedItem);
                }
            }
            break;
        case 3:
            printf("Thanks\n");
            break;
        default:
            printf("Please enter the right option.\n");
    }
} while (n != 3);

return 0;
}
```

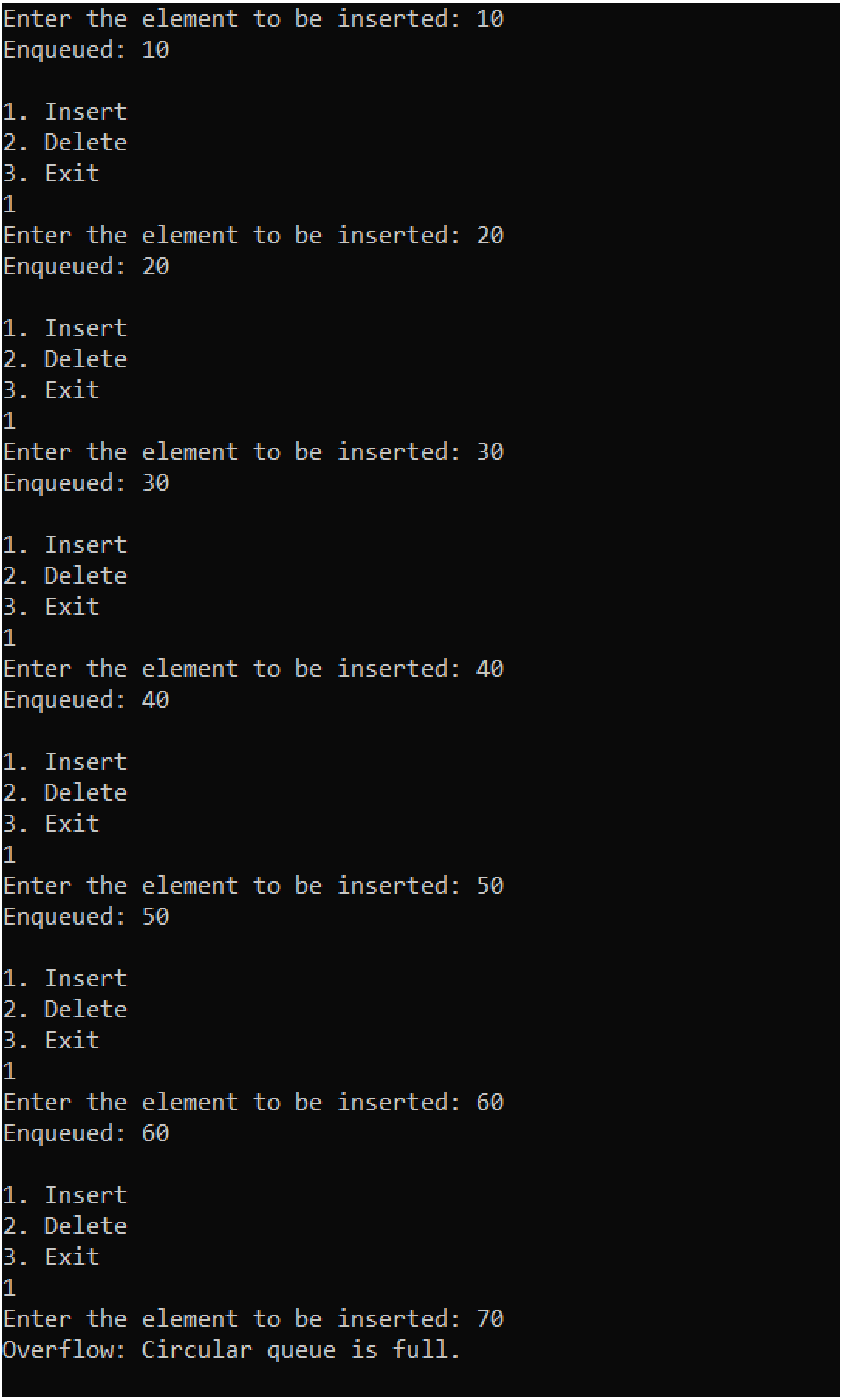
**Output:**

Circular Queue Underflow



```
1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 10
Enqueued: 10

1. Insert
2. Delete
3. Exit
2
Dequeued: 10
The element 10 is removed.

1. Insert
2. Delete
3. Exit
2
Underflow: Circular queue is empty.
```

Circular Queue Overflow



```
Enter the element to be inserted: 10
Enqueued: 10

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 20
Enqueued: 20

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 30
Enqueued: 30

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 40
Enqueued: 40

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 50
Enqueued: 50

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 60
Enqueued: 60

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 70
Overflow: Circular queue is full.
```

**Lab Program 4**

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

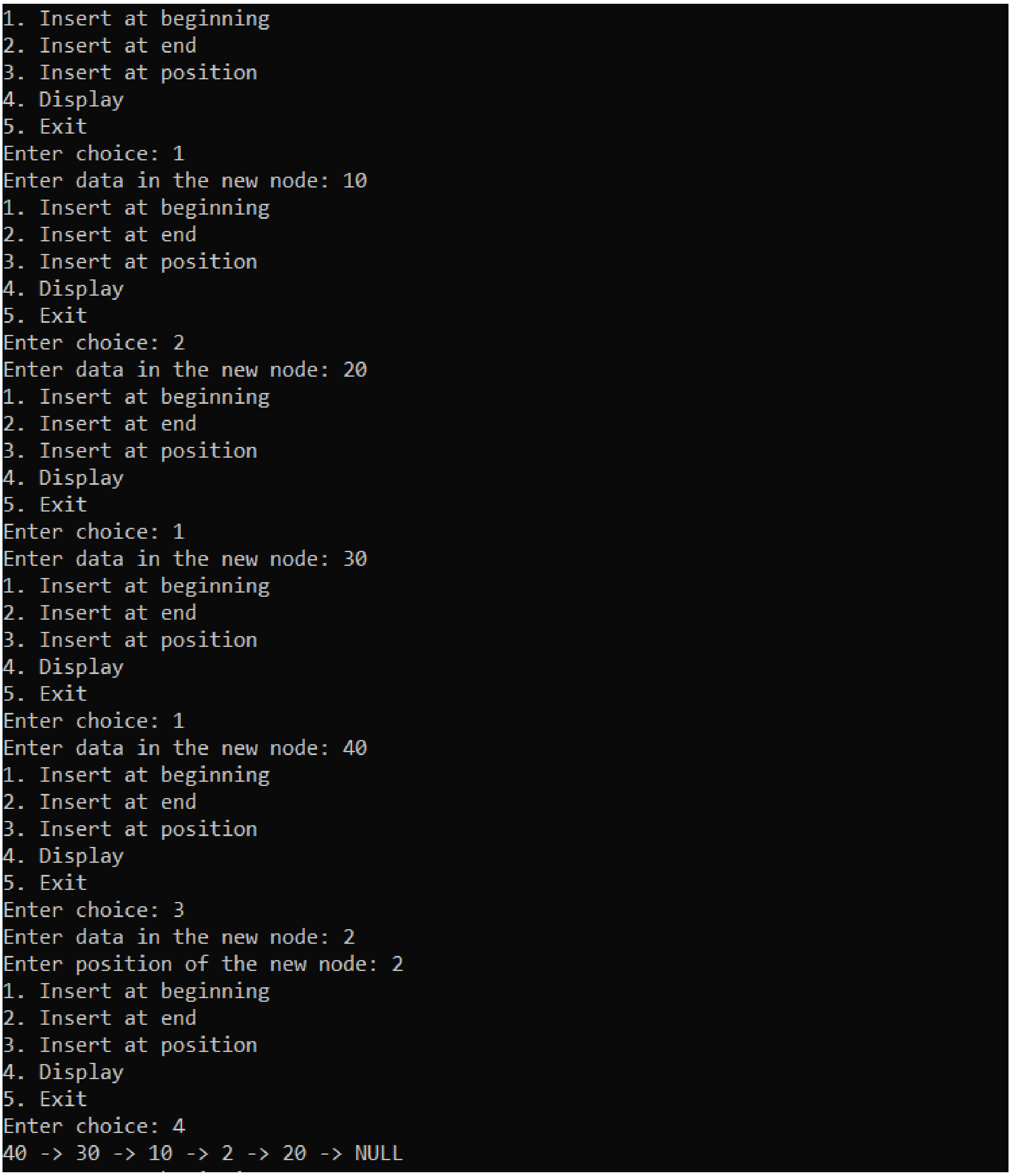
**a) Create a linked list.**

**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>

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

Node* head = NULL;

void push();
void append();
void insert();
void display();

int main() {
    int choice;
    while (1) {
        printf("1. Insert at beginning\n");
        printf("2. Insert at end\n");
        printf("3. Insert at position\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                push();
                break;
            case 2:
                append();
                break;
            case 3:
                insert();
                break;
            case 4:
                display();
                break;
            default:
                printf("Exiting the program");
                return 0;
        }
    }
}

void push() {
    Node* temp = (Node*)malloc(sizeof(Node));
    int new_data;
    printf("Enter data in the new node: ");
    scanf("%d", &new_data);
    temp->data = new_data;
    temp->next = head;
    head = temp;
}

void append() {
    Node* temp = (Node*)malloc(sizeof(Node));
    int new_data;
    printf("Enter data in the new node: ");
    scanf("%d", &new_data);
    temp->data = new_data;
    temp->next = NULL;
    if (head == NULL) {
        head = temp;
        return;
    }
    Node* temp1 = head;
    while (temp1->next != NULL) {
        temp1 = temp1->next;
    }
    temp1->next = temp;
}

void insert() {
    Node* temp = (Node*)malloc(sizeof(Node));
    int new_data, pos;
    printf("Enter data in the new node: ");
    scanf("%d", &new_data);
    printf("Enter position of the new node: ");
    scanf("%d", &pos);
    temp->data = new_data;
    temp->next = NULL;
    if (pos == 0) {
        temp->next = head;
        head = temp;
        return;
    }
    Node* temp1 = head;
    while (pos--) {
        temp1 = temp1->next;
    }
    Node* temp2 = temp1->next;
    temp->next = temp2;
    temp1->next = temp;
}

void display() {
    Node* temp1 = head;
    while (temp1 != NULL) {
        printf("%d -> ", temp1->data);
        temp1 = temp1->next;
    }
    printf("NULL\n");
}
```

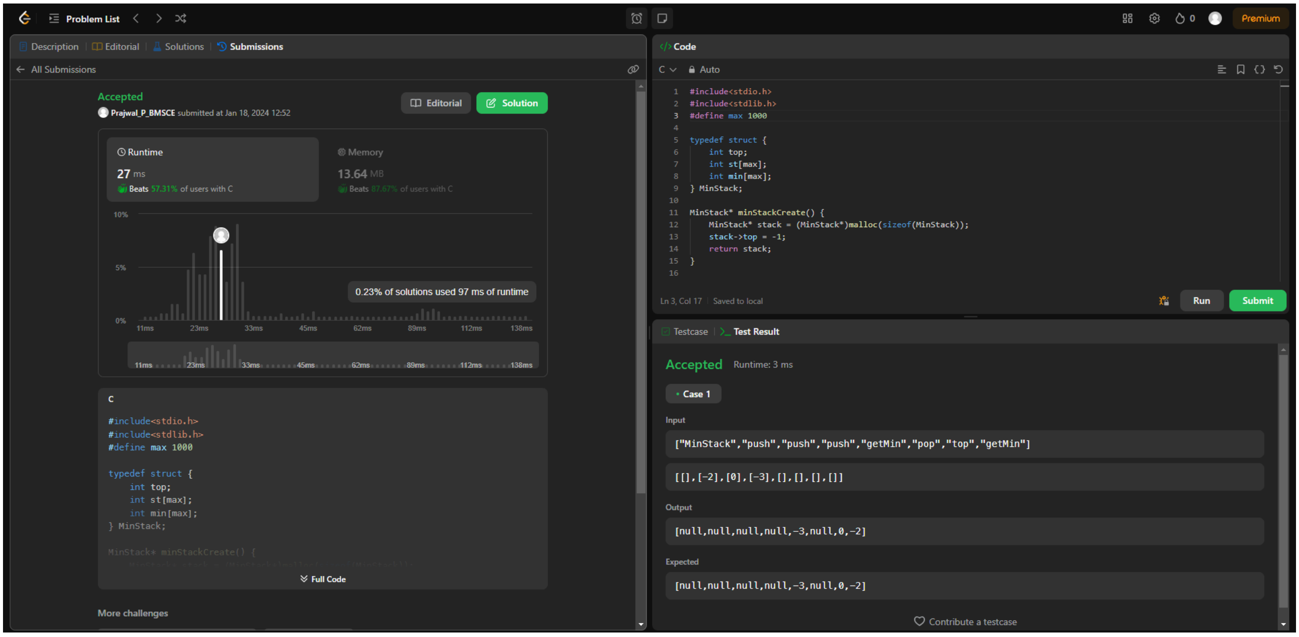
**Output:**



```
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 1
Enter data in the new node: 10
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 2
Enter data in the new node: 20
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 1
Enter data in the new node: 30
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 1
Enter data in the new node: 40
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 3
Enter data in the new node: 2
Enter position of the new node: 2
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 4
40 -> 30 -> 10 -> 2 -> 20 -> NULL
```

**Minstack Question [LeetCode]**

```
#include<stdio.h>
#include<stdlib.h>
#define max 1000

typedef struct {
    int top;
    int st[max];
    int min[max];
} MinStack;

MinStack* minStackCreate() {
    MinStack* stack = (MinStack*)malloc(sizeof(MinStack));
    stack->top = -1;
    return stack;
}

void minStackPush(MinStack* obj, int val) {
    if(obj->top == max-1){
        printf("Stack Full\n");
        return;
    }
    obj->st[++obj->top] = val;

    if(obj->top > 0)
    {
        if(obj->min[obj->top - 1] < val)
            obj->min[obj->top] = obj->min[obj->top - 1];
        else
            obj->min[obj->top] = val;
    }
    else
        obj->min[obj->top] = val;
}

void minStackPop(MinStack* obj) {
    if(obj->top == -1)
    {
        printf("Stack empty\n");
        return;
    }
    else {
        obj->top -= 1;
    }
}

int minStackTop(MinStack* obj) {
    if(obj->top == -1)
    {
        printf("Stack empty\n");
        return -1;
    }
    return obj->st[obj->top];
}

int minStackGetMin(MinStack* obj) {
    if(obj->top == -1)
    {
        printf("min Stack empty\n");
        return -1;
    }
    return obj->min[obj->top];
}

void minStackFree(MinStack* obj) {
    free(obj);
}
```



Accepted

Prajwal_P_BMSCE submitted at Jan 18, 2024 12:52

Runtime

27 ms

Beats 57.31% of users with C

Memory

13.64 MB

Beats 87.67% of users with C

0.23% of solutions used 97 ms of runtime

C

```
#include<stdio.h>
#include<stdlib.h>
#define max 1000

typedef struct {
    int top;
    int st[max];
    int min[max];
} MinStack;

MinStack* minStackCreate() {
```

```
#include<stdio.h>
#include<stdlib.h>
#define max 1000

typedef struct {
    int top;
    int st[max];
    int min[max];
} MinStack;

MinStack* minStackCreate() {
    MinStack* stack = (MinStack*)malloc(sizeof(MinStack));
    stack->top = -1;
    return stack;
}

```

Ln 3, Col 17 | Saved to local

Accepted Runtime: 3 ms

Case 1

Input

```
["MinStack","push","push","push","getMin","pop","top","getMin"]
```

```
[[],[-2],[0],[-3],[],[],[],[]]
```

Output

```
[null,null,null,null,-3,null,0,-2]
```

Expected

```
[null,null,null,null,-3,null,0,-2]
```

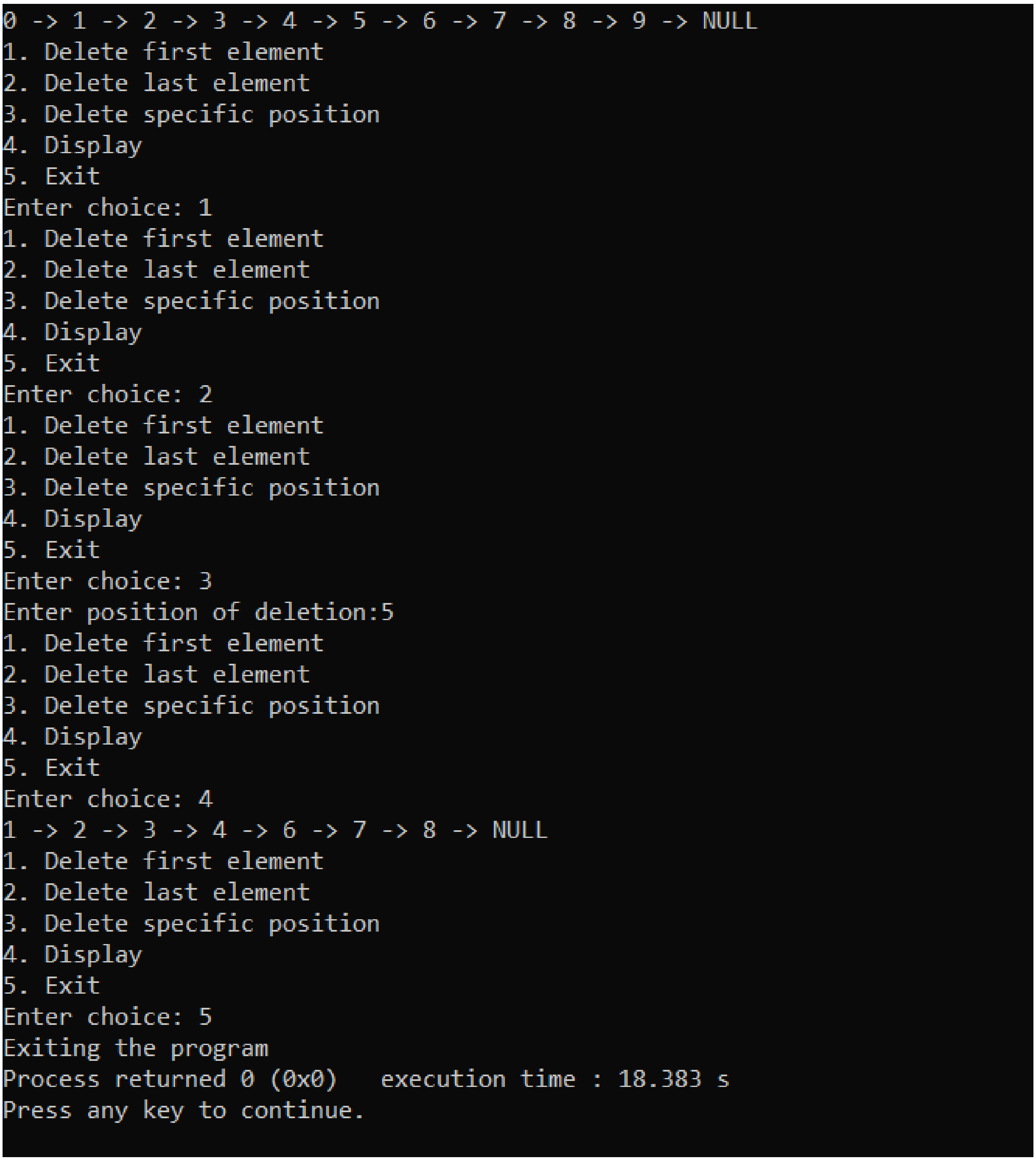
## Lab Program 5

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

- **Create a linked list.**
- **Deletion of first element, specified element and last element in the list.**
- **Display the contents of the linked list.**

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

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

Node* head=NULL;

void display();
void create();
void fdelete();
void ldelete();
void idelete();


int main() {
   create();
   int choice;
   display();
   while (1) {
      printf("1. Delete first element\n");
      printf("2. Delete last element\n");
      printf("3. Delete specific position\n");
      printf("4. Display\n");
      printf("5. Exit\n");
      printf("Enter choice: ");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            fdelete();
            break;
         case 2:
            ldelete();
            break;
         case 3:
            idelete();
            break;
         case 4:
            display();
            break;
         default:
                printf("Exiting the program");
                return 0;
        }
    }
}


void fdelete(){
    if(head==NULL){
        printf("Empty list");
    }
    Node *ptr=head;
    head=head->next;
    free(ptr);
}

void ldelete(){
    if(head==NULL){
        printf("Empty list");
    }
    Node *ptr=head;
    Node *ptr1=head;
    while(ptr->next!=NULL){
        ptr1=ptr;
        ptr=ptr->next;
    }
    ptr1->next=NULL;
    free(ptr);
}

void idelete(){
    printf("Enter position of deletion:");
    int pos;
    scanf("%d",&pos);
     if(head==NULL){
        printf("Empty list");
    }
    Node *ptr=head;
    Node *ptr1=head;
    pos=pos-1;
    while(pos--){
        ptr1=ptr;
        ptr=ptr->next;
    }
    ptr1->next=ptr->next;
    free(ptr);
}

void display() {
    Node* temp1 = head;
    while (temp1 != NULL) {
        printf("%d -> ", temp1->data);
        temp1 = temp1->next;
    }
    printf("NULL\n");
}

void create(){
    int A[]={0,1,2,3,4,5,6,7,8,9};
    struct Node *t, *last;
    head = (struct Node*)malloc(sizeof(struct Node));
    head->data = A[0];
    head->next = NULL;
    last = head;
    for(int i=1; i<10; i++){
        t = (struct Node*)malloc(sizeof(struct Node));
        t->data = A[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }

}
```

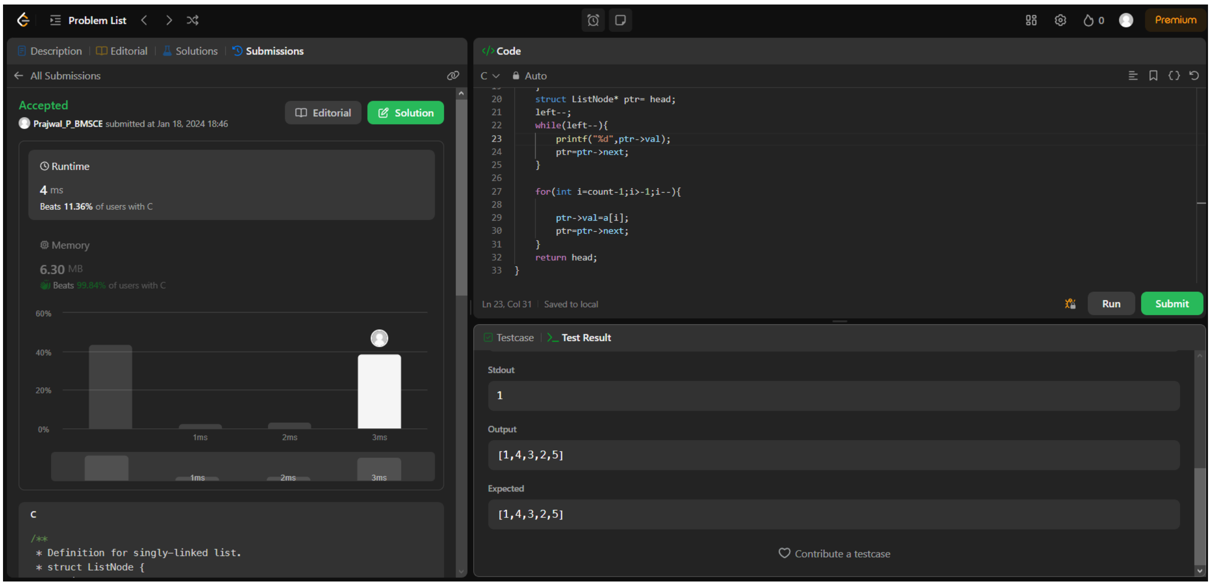
**Output:**



```
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> NULL
1. Delete first element
2. Delete last element
3. Delete specific position
4. Display
5. Exit
Enter choice: 1
1. Delete first element
2. Delete last element
3. Delete specific position
4. Display
5. Exit
Enter choice: 2
1. Delete first element
2. Delete last element
3. Delete specific position
4. Display
5. Exit
Enter choice: 3
Enter position of deletion:5
1. Delete first element
2. Delete last element
3. Delete specific position
4. Display
5. Exit
Enter choice: 4
1 -> 2 -> 3 -> 4 -> 6 -> 7 -> 8 -> NULL
1. Delete first element
2. Delete last element
3. Delete specific position
4. Display
5. Exit
Enter choice: 5
Exiting the program
Process returned 0 (0x0)   execution time : 18.383 s
Press any key to continue.
```

## Reverse Linked List Question [LeetCode]

```c
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
    struct ListNode* ptrl= head;
    int temp=left-1;
    while(temp--){
        ptrl=ptrl->next;
    }
    int count=right-left+1;
    int* a = (int*)malloc(count * sizeof(int));
    for(int i=0;i<count;i++){
        a[i]=ptrl->val;
        ptrl=ptrl->next;
    }
    struct ListNode* ptr= head;
    left--;
    while(left--){
        printf("%d",ptr->val);
        ptr=ptr->next;
    }

    for(int i=count-1;i>-1;i--){

        ptr->val=a[i];
        ptr=ptr->next;
    }
    return head;
}
```



Accepted

Prajwal_P_BMSCE submitted at Jan 18, 2024 18:46

Runtime

4 ms

Beats 11.36% of users with C

Memory

6.30 MB

Beats 99.84% of users with C

C

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
```

```c
    struct ListNode* ptr= head;
    left--;
    while(left--){
        printf("%d",ptr->val);
        ptr=ptr->next;
    }

    for(int i=count-1;i>-1;i--){

        ptr->val=a[i];
        ptr=ptr->next;
    }
    return head;
}
```

Ln 23, Col 31 | Saved to local

Stdout

```
1
```

Output

```
[1,4,3,2,5]
```

Expected

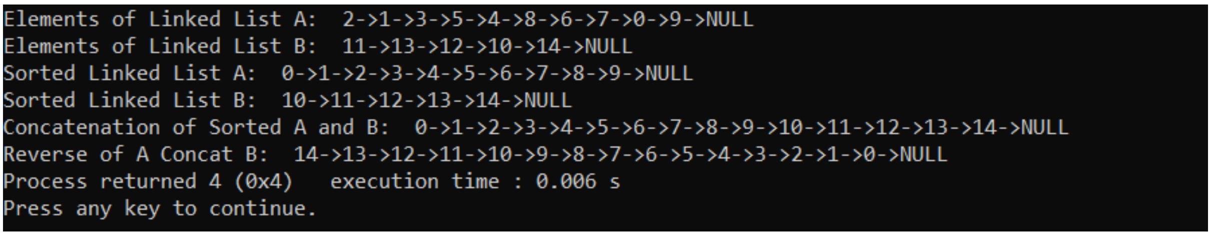
```
[1,4,3,2,5]
```

## Lab Program 6:

**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;
};
struct node *head,*headB;
void createA(){

   int A[]={2,1,3,5,4,8,6,7,0,9};
   printf("Elements of Linked List A:  ");
   for(int i=0;i<10;i++){
      printf("%d->",A[i]);
   }
   printf("NULL");
   struct node *t, *last;
   head = (struct node*)malloc(sizeof(struct node));
   head->data = A[0];
   head->next = NULL;
   last = head;
   for(int i=1; i<10; i++){
      t = (struct node*)malloc(sizeof(struct node));
      t->data = A[i];
      t->next = NULL;
      last->next = t;
      last = t;
   }
}

void createB(){

   int B[]={11,13,12,10,14};
   printf("\nElements of Linked List B:  ");
   for(int i=0;i<5;i++){
      printf("%d->",B[i]);
   }
   printf("NULL");
   struct node *t, *last;
   headB = (struct node*)malloc(sizeof(struct node));
   headB->data = B[0];
   headB->next = NULL;
   last = headB;
   for(int i=1; i<5; i++){
        t = (struct node*)malloc(sizeof(struct node));
        t->data = B[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }
}

struct node* sort(struct node *head) {
    struct node *ptr1,*ptr2;
    ptr1=head;
    int temp,count=1;
    while(ptr1->next!=NULL){
        count++;
        ptr1=ptr1->next;
    }
    while(--count){
        ptr1=head;
        while(ptr1->next!=NULL){
            ptr2=ptr1;
            ptr1=ptr1->next;
            if(ptr1->data<ptr2->data){
                temp=ptr2->data;
                ptr2->data=ptr1->data;
                ptr1->data=temp;
            }
        }
    }
    return head;
}

struct node* concat(struct node *head, struct node *headB){
    struct node *ptr=head;
    while(ptr->next!=NULL){
                ptr=ptr->next;
    }
    ptr->next=headB;
    return head;
}
struct node* reverse(struct node *head){
    struct node *pre, *cur, *after;
    pre = NULL;
    after = cur = head;
    while(after != NULL)
    {
        after = after->next;
        cur->next = pre;
        pre = cur;
        cur = after;
    }
    head = pre;
    return head;
}
void main(){
    createA();
    createB();
    head = sort(head);
    struct node *p=head;
    printf("\nSorted Linked List A:  ");
    while(p!=NULL){
            printf("%d->",p->data);
            p = p->next;
    }
    printf("NULL");
    headB = sort(headB);
    p = headB;
    printf("\nSorted Linked List B:  ");
    while(p!=NULL){
            printf("%d->",p->data);
            p = p->next;
    }
    printf("NULL");
    p=concat(head,headB);
    printf("\nConcatenation of Sorted A and B:  ");
    while(p!=NULL){
            printf("%d->",p->data);
            p = p->next;
    }
    printf("NULL\n");
    p=reverse(head);
    printf("Reverse of A Concat B:  ");
    while(p!=NULL){
            printf("%d->",p->data);
            p = p->next;
    }
     printf("NULL");
}
```

**Output:**



```
Elements of Linked List A:  2->1->3->5->4->8->6->7->0->9->NULL
Elements of Linked List B:  11->13->12->10->14->NULL
Sorted Linked List A:  0->1->2->3->4->5->6->7->8->9->NULL
Sorted Linked List B:  10->11->12->13->14->NULL
Concatenation of Sorted A and B:  0->1->2->3->4->5->6->7->8->9->10->11->12->13->14->NULL
Reverse of A Concat B:  14->13->12->11->10->9->8->7->6->5->4->3->2->1->0->NULL
Process returned 4 (0x4)   execution time : 0.006 s
Press any key to continue.
```

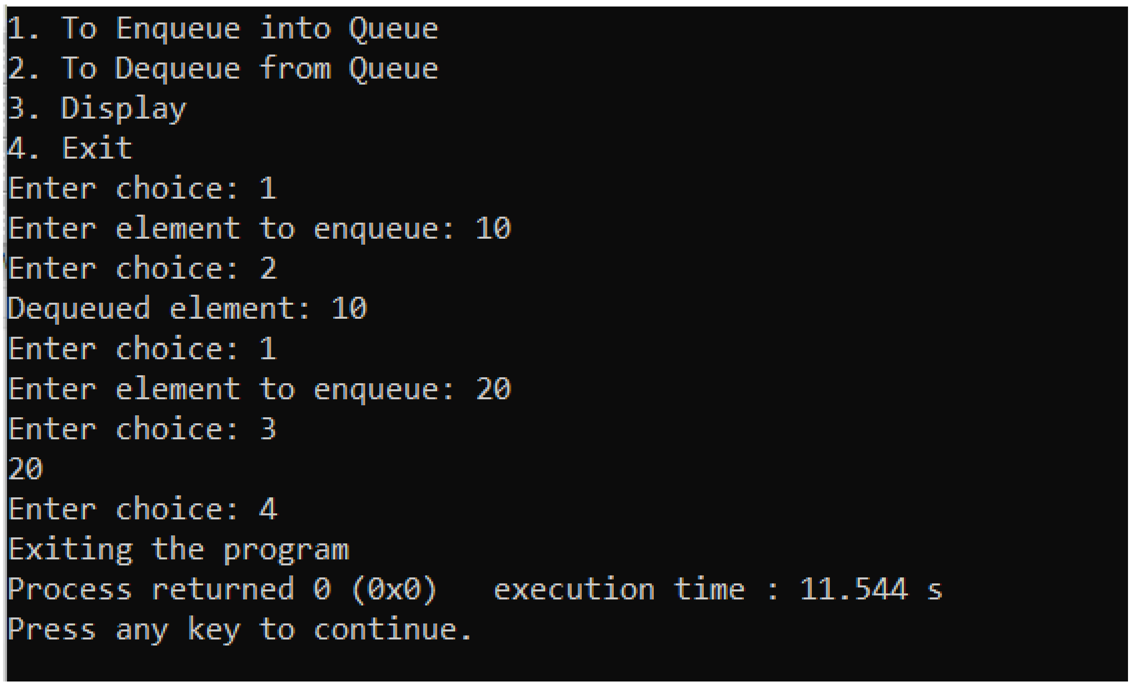
**WAP to Implement Single Link List to simulate Stack & Queue Operations.**

**Queue Implementation**

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{
   int val;
   struct Node *next;
}Node;

Node *head=NULL;

void enqueue(){
   Node *ptr=(Node*)malloc(sizeof(Node));
   printf("Enter element to enqueue: ");
   int num;
   scanf("%d",&num);
   ptr->val=num;
   if(head==NULL){
      head=ptr;
      ptr->next=NULL;
   }
   else{
      Node *ptr2=head;
      while(ptr2->next!=NULL){
         ptr2=ptr2->next;
      }
      ptr2->next=ptr;
   }
}

void display(){
   Node *ptr=head;
   while(ptr!=NULL){
      printf("%d ",ptr->val);
      ptr=ptr->next;
   }
   printf("\n");
}

void dequeue(){
   Node *ptr=head;
   int num=ptr->val;
   head=head->next;
   free(ptr);
   printf("Dequeued element: %d\n",num);
}

int main(){
   int choice;
```

```
    printf("1. To Enqueue into Queue\n");
    printf("2. To Dequeue from Queue\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    printf("Enter choice: ");
    while (1) {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            default:
                printf("Exiting the program");
                return 0;
        }
        printf("Enter choice: ");
    }
}
```

**Output:**

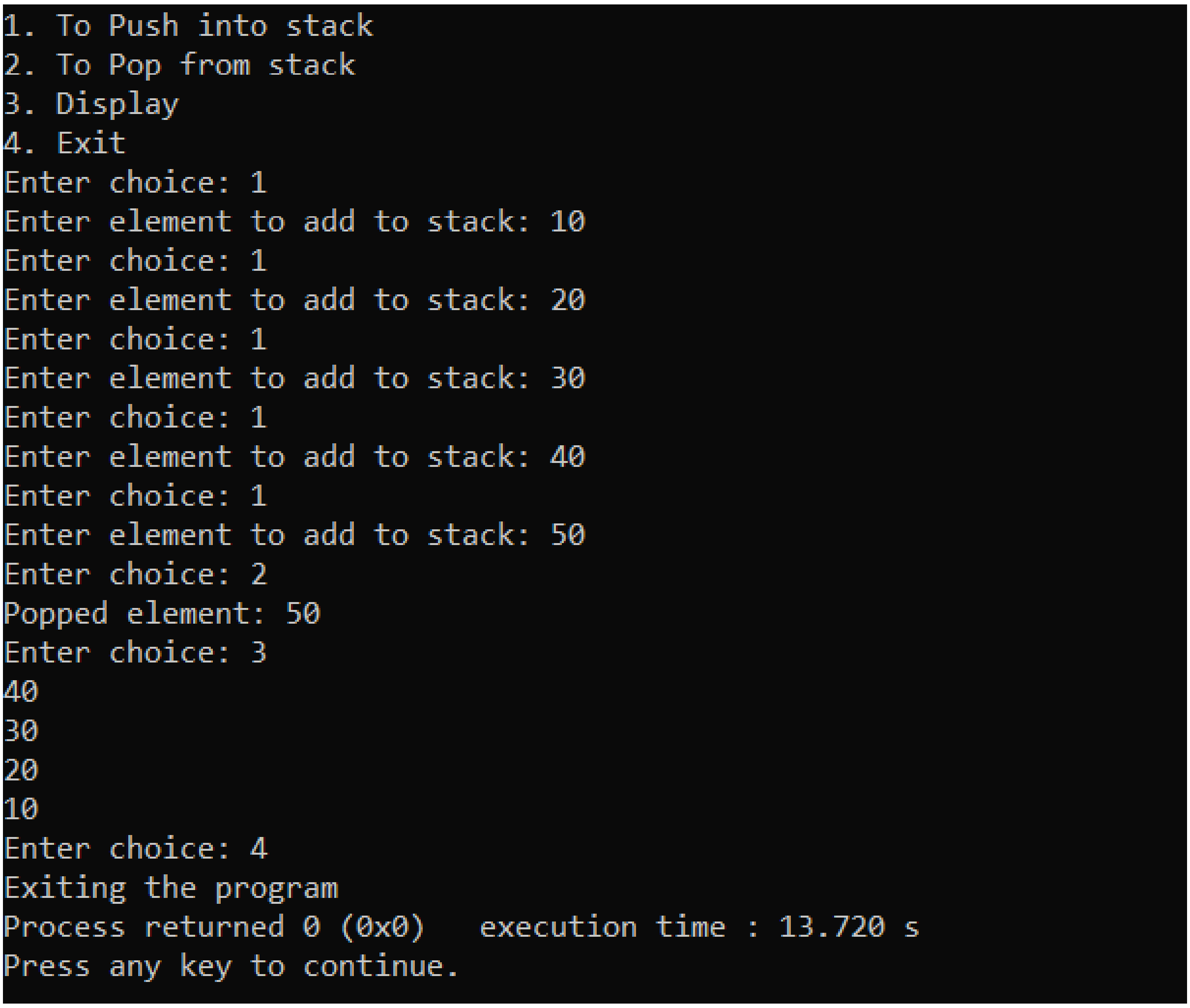


```
1. To Enqueue into Queue
2. To Dequeue from Queue
3. Display
4. Exit
Enter choice: 1
Enter element to enqueue: 10
Enter choice: 2
Dequeued element: 10
Enter choice: 1
Enter element to enqueue: 20
Enter choice: 3
20
Enter choice: 4
Exiting the program
Process returned 0 (0x0)   execution time : 11.544 s
Press any key to continue.
```

**Stack Implementation**

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int val;
    struct Node *next;
}Node;

struct Node *top=NULL;

void pop(){
    if(top==NULL){
        printf("Empty stack");
    }
    struct Node *ptr=top;
    top=top->next;
    int num= ptr->val;
    free(ptr);
    printf("Popped element: %d\n",num);
}

void push() {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    int new_data;
    printf("Enter element to add to stack: ");
    scanf("%d", &new_data);
    temp->val = new_data;
    temp->next = top;
    top=temp;
}

void display() {
    struct Node* temp1 = top;
    while (temp1 != NULL) {
        printf("%d\n", temp1->val);
        temp1 = temp1->next;
    }
}

int main(){
   int choice;
    printf("1. To Push into stack\n");
    printf("2. To Pop from stack\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    printf("Enter choice: ");
    while (1) {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            default:
                printf("Exiting the program");
                return 0;
        }
        printf("Enter choice: ");
    }
}
```

**Output:**



```
1. To Push into stack
2. To Pop from stack
3. Display
4. Exit
Enter choice: 1
Enter element to add to stack: 10
Enter choice: 1
Enter element to add to stack: 20
Enter choice: 1
Enter element to add to stack: 30
Enter choice: 1
Enter element to add to stack: 40
Enter choice: 1
Enter element to add to stack: 50
Enter choice: 2
Popped element: 50
Enter choice: 3
40
30
20
10
Enter choice: 4
Exiting the program
Process returned 0 (0x0)   execution time : 13.720 s
Press any key to continue.
```

## Lab Programs 7:

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

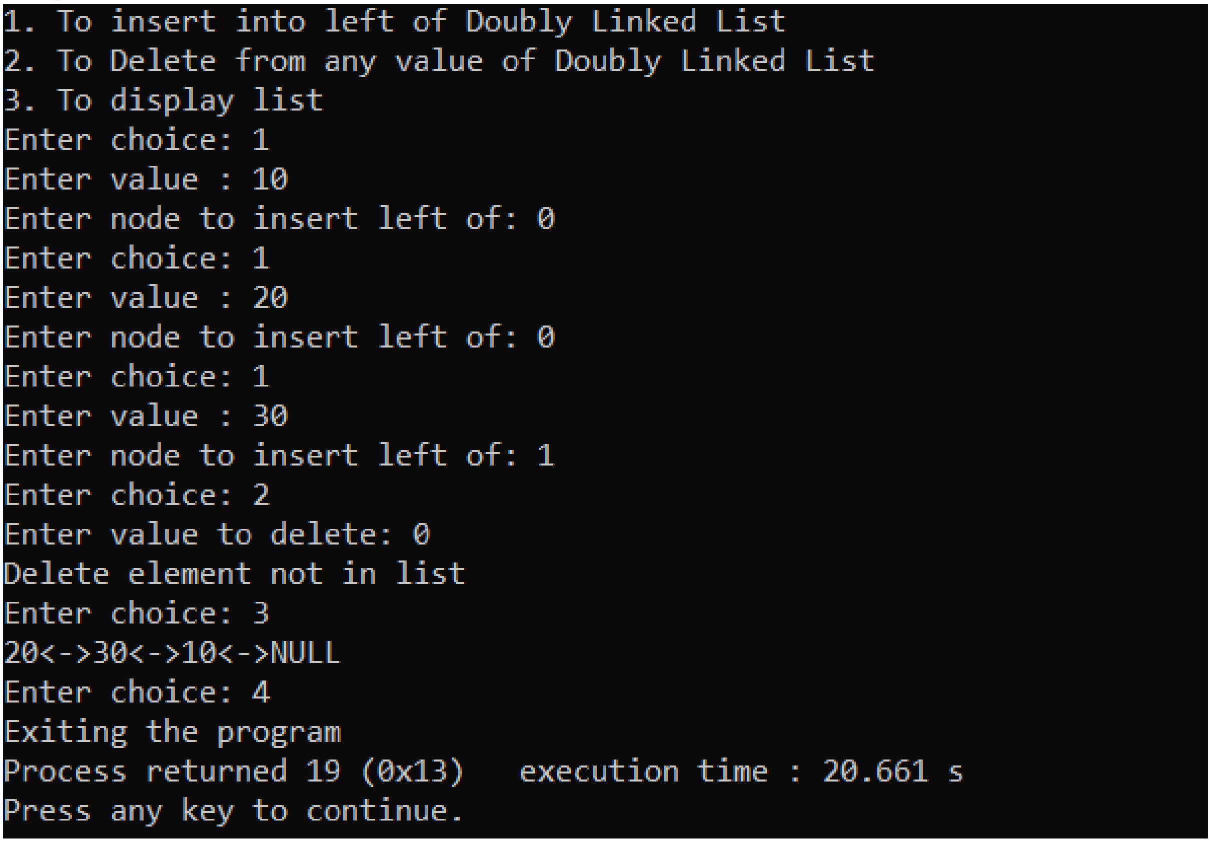
- **Create a doubly linked list.**
- **Insert a new node to the left of the node.**
- **Delete the node based on a specific value**
- **Display the contents of the list**

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{
   int val;
   struct Node *prev;
   struct Node *next;
}Node;

Node *head=NULL;

void insert(){
   int num,pos;
   printf("Enter value : ");
   scanf("%d",&num);
   printf("Enter node to insert left of: ");
   scanf("%d",&pos);

   Node *ptr=(Node*)malloc(sizeof(Node));
   ptr->val=num;

   if(pos==0){
   ptr->next=head;
   ptr->prev=NULL;
      if (head != NULL){
         head->prev = ptr;
      }
   head=ptr;

   }
   Node *ptr1=head;

   if(pos!=0){
      for(int i=0;i<pos;i++){
         ptr1=ptr1->next;
      }
      ptr->next=ptr1;
      ptr->prev=ptr1->prev;
      ptr1->prev->next=ptr;
      ptr1->prev=ptr;
```

```
    }
}

void delete(){
    printf("Enter value to delete: ");
    int loc=-1,len=1,val;
    scanf("%d",&val);
    Node *ptr=head,*ptr2;
    while(ptr->next!=NULL){
        len++;
        ptr=ptr->next;
    }
    ptr=head;
    for(int i=0;i<len;i++){
        if(ptr->val==val){
            loc=i;
        }
        ptr=ptr->next;
    }
    if(loc==-1){
        printf("Delete element not in list\n");
        return;
    }
    if(loc==0){
        printf("Deleted element: %d\n",head->val);
        ptr=head;
        head=head->next;
        free(ptr);
        return;
    }
    if(loc==len){
        ptr = head;
        while (ptr->next != NULL) {
            ptr = ptr->next;
        }
        printf("Deleted element: %d\n", ptr->val);
        ptr->prev->next = NULL;
        free(ptr);
        return;
    }
    ptr=head;
    loc++;
    while(--loc){
        ptr2=ptr;
        ptr=ptr->next;
    }
    printf("Deleted element: %d\n",ptr->val);
    ptr2->next=ptr->next;
    ptr->next->prev=ptr2;
    free(ptr);
```

```
}

void display(){
    Node *ptr=head;
    while(ptr!=NULL){
        printf("%d<->",ptr->val);
        ptr=ptr->next;
    }
    printf("NULL\n");
}

void main(){
   int choice;
    printf("1. To insert into left of Doubly Linked List\n");
    printf("2. To Delete from any value of Doubly Linked List\n");
    printf("3. To display list\n");
    printf("Enter choice: ");
    while (1) {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                insert();
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            default:
                printf("Exiting the program");
                return 0;
        }
        printf("Enter choice: ");
    }
}
```

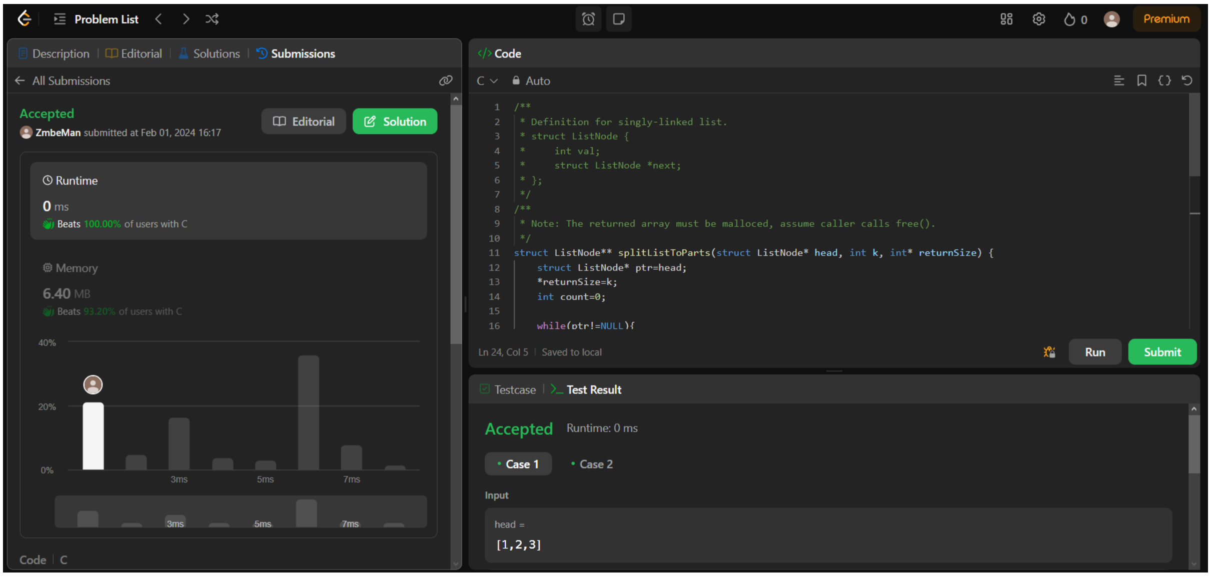
**Output:**



```
1. To insert into left of Doubly Linked List
2. To Delete from any value of Doubly Linked List
3. To display list
Enter choice: 1
Enter value : 10
Enter node to insert left of: 0
Enter choice: 1
Enter value : 20
Enter node to insert left of: 0
Enter choice: 1
Enter value : 30
Enter node to insert left of: 1
Enter choice: 2
Enter value to delete: 0
Delete element not in list
Enter choice: 3
20<->30<->10<->NULL
Enter choice: 4
Exiting the program
Process returned 19 (0x13)   execution time : 20.661 s
Press any key to continue.
```

**Split Linked List into parts [LeetCode]**

```
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize) {
    struct ListNode* ptr=head;
    *returnSize=k;
    int count=0;

    while(ptr!=NULL){
        count++;
        ptr=ptr->next;
    }

    int nums=count/k,a=count%k;

    struct ListNode **L=(struct ListNode**)calloc(k,sizeof(struct ListNode*));

    ptr=head;
    for(int i=0;i<k;i++){
        L[i] = ptr;

        int segmentSize = nums + (a-- > 0 ? 1 : 0);
        for (int j = 1; j < segmentSize; j++) {
            ptr = ptr->next;
        }

        if (ptr != NULL) {
            struct ListNode* next = ptr->next;
            ptr->next = NULL;
            ptr = next;
        }
    }
    return L;
}
```



Accepted

ZmbeMan submitted at Feb 01, 2024 16:17

Runtime: 0 ms, Beats 100.00% of users with C

Memory: 6.40 MB, Beats 93.20% of users with C

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize) {
    struct ListNode* ptr=head;
    *returnSize=k;
    int count=0;

    while(ptr!=NULL){
```

Ln 24, Col 5 | Saved to local

Accepted Runtime: 0 ms

Input

head =
[1,2,3]

**Lab Program 8:**

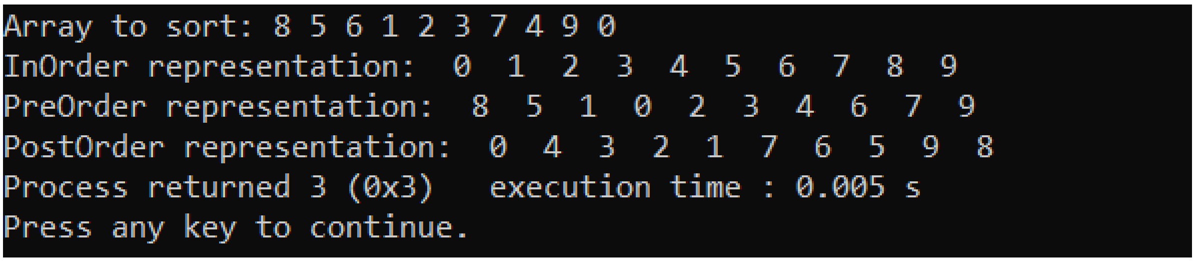
**Write a program**

- **To construct a binary Search tree.**
- **To traverse the tree using all the methods i.e., in-order, preorder and post order**
- **To display the elements in the tree**

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

struct Tree{
   int val;
   struct Tree *left;
   struct Tree *right;
};

struct Tree *root=NULL;

struct Tree* newNode(int a){
   struct Tree *ptr=(struct Tree*)malloc(sizeof(struct Tree));
      ptr->val=a;
      ptr->left=ptr->right=NULL;
      return ptr;
}

struct Tree* insert(struct Tree *ptr, int a){
   if(ptr==NULL){
      return newNode(a);
   }
   else if(ptr->val<a){
      ptr->right=insert(ptr->right,a);
   }
   else if(ptr->val>a){
      ptr->left=insert(ptr->left,a);
   }
}

void inOrder(struct Tree *ptr){
   if (ptr != NULL) {
      inOrder(ptr->left);
      printf(" %d ", ptr->val);
      inOrder(ptr->right);
   }
}

void PreOrder(struct Tree *ptr){
   if (ptr != NULL) {
      printf(" %d ", ptr->val);
      PreOrder(ptr->left);
```

```
        PreOrder(ptr->right);
    }
}

void PostOrder(struct Tree *ptr){
    if (ptr != NULL) {
        PostOrder(ptr->left);
        PostOrder(ptr->right);
        printf(" %d ", ptr->val);
    }
}

void main(){
    int arr[]={8,5,6,1,2,3,7,4,9,0};
    printf("Array to sort: %d ",arr[0]);
    root=insert(root,arr[0]);

    for(int i=1;i<10;i++){
        printf("%d ",arr[i]);
        insert(root,arr[i]);
    }printf("\nInOrder representation: ");
    inOrder(root);
    printf("\nPreOrder representation: ");
    PreOrder(root);
    printf("\nPostOrder representation: ");
    PostOrder(root);}
```

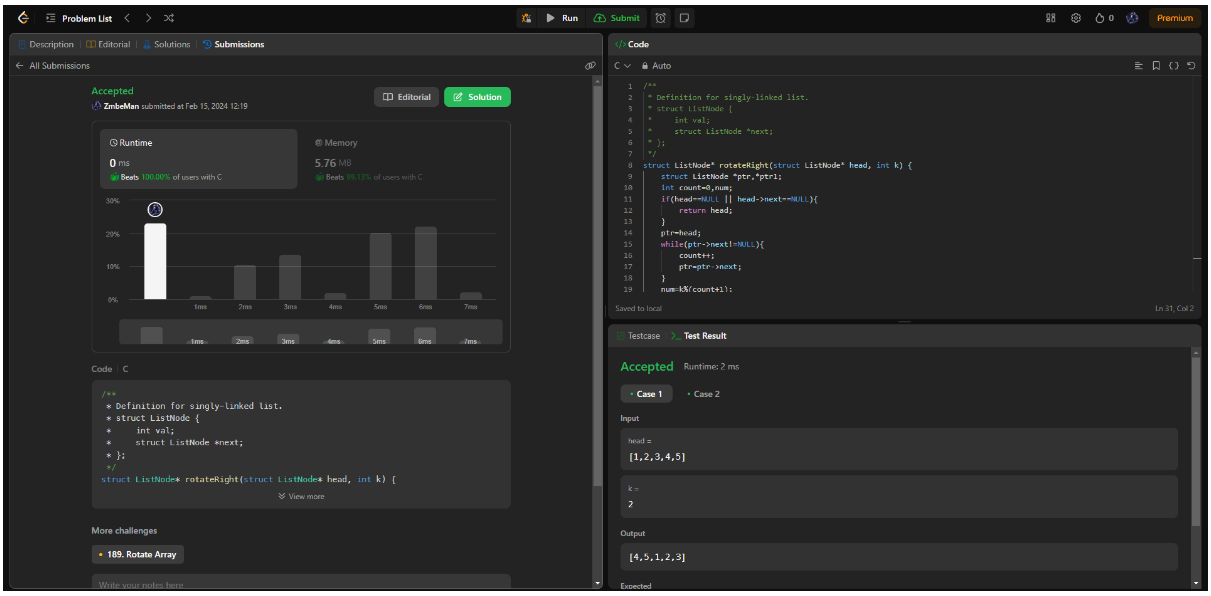
**Output:**



```
Array to sort: 8 5 6 1 2 3 7 4 9 0
InOrder representation:  0  1  2  3  4  5  6  7  8  9
PreOrder representation:  8  5  1  0  2  3  4  6  7  9
PostOrder representation:  0  4  3  2  1  7  6  5  9  8
Process returned 3 (0x3)   execution time : 0.005 s
Press any key to continue.
```

## Rotate List [LeetCode]

```
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize) {
    struct ListNode* ptr=head;
    *returnSize=k;
    int count=0;

    while(ptr!=NULL){
        count++;
        ptr=ptr->next;
    }

    int nums=count/k,a=count%k;

    struct ListNode **L=(struct ListNode**)calloc(k,sizeof(struct ListNode*));

    ptr=head;
    for(int i=0;i<k;i++){
        L[i] = ptr;

        int segmentSize = nums + (a-- > 0 ? 1 : 0);
        for (int j = 1; j < segmentSize; j++) {
            ptr = ptr->next;
        }

        if (ptr != NULL) {
            struct ListNode* next = ptr->next;
            ptr->next = NULL;
            ptr = next;
        }
    }
    return L;
}
```



Accepted
ZmbeMan submitted at Feb 15, 2024 12:19

Runtime
0 ms
Beats 100.00% of users with C

Memory
5.76 MB
Beats 99.13% of users with C

Code | C

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

• 189. Rotate Array

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* rotateRight(struct ListNode* head, int k) {
    struct ListNode *ptr,*ptr1;
    int count=0,num;
    if(head==NULL || head->next==NULL){
        return head;
    }
    ptr=head;
    while(ptr->next!=NULL){
        count++;
        ptr=ptr->next;
    }
    num=k%(count+1);
```

Saved to local

Ln 31, Col 2

Accepted Runtime: 2 ms

• Case 1 • Case 2

Input

head =
[1,2,3,4,5]

k =
2

Output

[4,5,1,2,3]

Expected

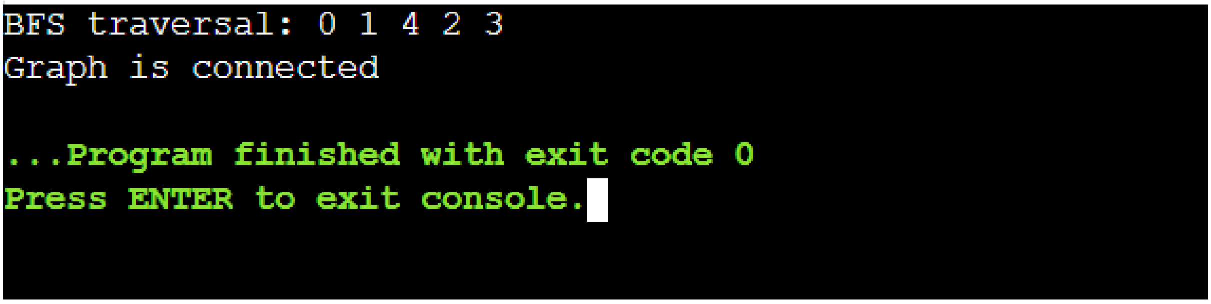
**Lab Program 9:**

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

```
#include<stdio.h>
#define MAX_VERTICES 50
typedef struct Graph_t {
   int V;
   int adj[MAX_VERTICES][MAX_VERTICES];
} Graph;
int DFS_V[50];
Graph* Graph_create(int V)
{
   Graph* g = malloc(sizeof(Graph));
   g->V = V;

   for (int i = 0; i <=V; i++) {
      for (int j = 0; j <=V; j++) {
         g->adj[i][j] = 0;
      }
   }

   return g;
}
void Graph_addEdge(Graph* g, int v, int w)
{
   g->adj[v][w] = 1;
   g->adj[w][v] = 1;
}
void BFS(Graph* g, int root){
   int visited[g->V+1];
   for(int i=0;i<=g->V;i++)
         visited[i]=0;

   int queue[g->V+1];
   int front=0,rear=0;
```

```
    visited[root]=1;

    queue[rear++]=root;
    while(front!=rear){
            root=queue[front++];
        printf("%d ",root);
        for(int i=0;i<=g->V;i++){
            if(g->adj[root][i]==1 && visited[i]!=1){
                visited[i]=1;
                queue[rear++]=i;
            }
        }
    }

}
int DFS(Graph *g,int root){
    for(int i=0;i<=g->V;i++){
        if(g->adj[root][i]==1 && DFS_V[i]!=1){
            DFS_V[i]=1;
            DFS(g,i);
        }
    }
    int count=0;
    for(int i=0;i<=g->V;i++){
        if(DFS_V[i]==1){
         count++;
        }
    }
    return count;
}
int main()
{

    Graph* g = Graph_create(4);
    Graph_addEdge(g, 0, 1);
    Graph_addEdge(g, 0, 4);
    Graph_addEdge(g, 1, 3);
    Graph_addEdge(g, 1, 2);
    Graph_addEdge(g, 2, 3);
    Graph_addEdge(g, 4, 3);
    printf("BFS traversal: ");
    BFS(g,0);
    int count=DFS(g,0);
    if(count==g->V+1){
        printf("\nGraph is connected");
    }
    else{
        printf("\nGraph is disconnected");
    }
}
```

**Output:**



```
BFS traversal: 0 1 4 2 3
Graph is connected

...Program finished with exit code 0
Press ENTER to exit console.
```

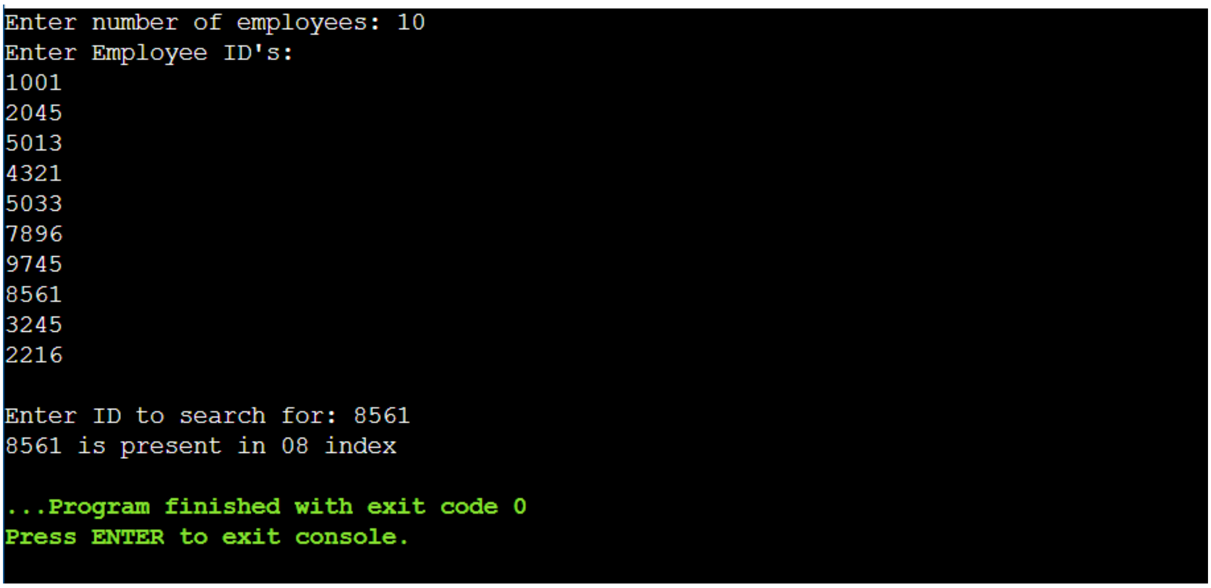
**Lab Program 10:**

**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 -> 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 size 10

int HT[size];
void hash(int a[]){
   int key;
   for(int i=0;i<10;i++){
      HT[i]=-1;
   }
   for(int i=0;i<10;i++){
      key=a[i]%size;
      if(HT[key]==-1){
         HT[key]=a[i];
      }
      else{
         while(HT[key]!=-1){
            key=(key+1)%size;
         }
         HT[key]=a[i];
      }
   }
}
void search(int num){
   int key=num%size;
   int loc=-1;
   if(HT[key]==num){
      loc=key;
   }
```

```
    else{
        while(HT[key]!=num){
            key=(key+1)%size;
        }
        if(HT[key]==num){
            loc=key;
        }
    }
    if(loc==-1){
        printf("%d is absent from HashTable",num);
    }
    else{
        printf("%d is present in 0%d index",num,loc);
    }
}
void main(){
    int n,num;
    printf("Enter number of employees: ");
    scanf("%d",&n);
    if(n!=size){
        printf("Invalid");
    }
    printf("Enter Employee ID's: \n");
    int data[10];
    for(int i=0;i<10;i++){
        scanf("%d",&data[i]);
    }
    printf("\n");
    hash(data);
    printf("Enter ID to search for: ");
    scanf("%d",&num);
    search(num);
}
```

**Output:**



```
Enter number of employees: 10
Enter Employee ID's:
1001
2045
5013
4321
5033
7896
9745
8561
3245
2216

Enter ID to search for: 8561
8561 is present in 08 index

...Program finished with exit code 0
Press ENTER to exit console.
```