VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Ritesh Mohan Nayak (1BM23CS350)

in partial fullfillment 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
September 2024-January 2025

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 Ritesh Mohan Nayak **(1BM23CS350)**, 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 year 2024-25. 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 Bagalkot
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
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.

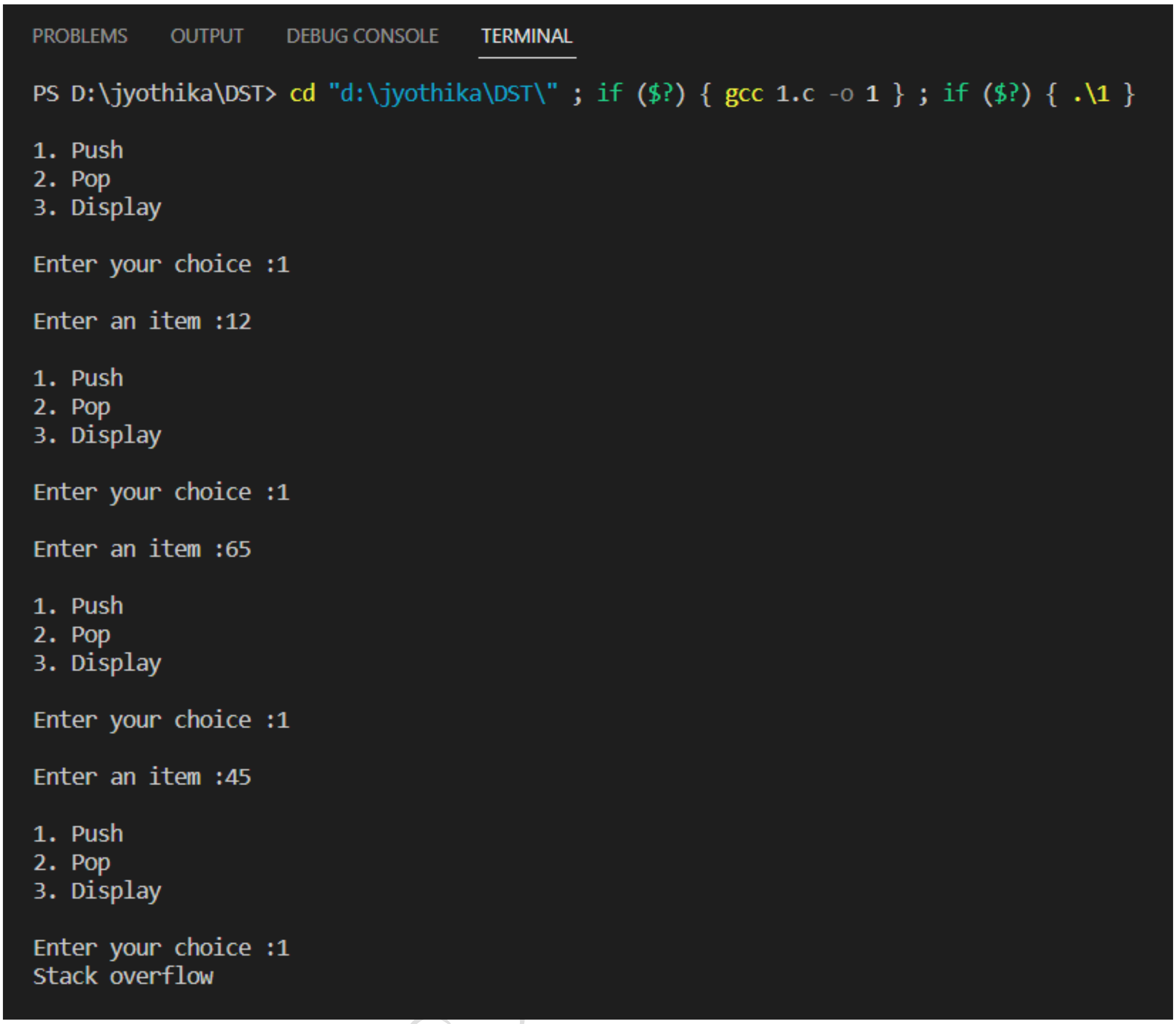
Lab program 1:

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.

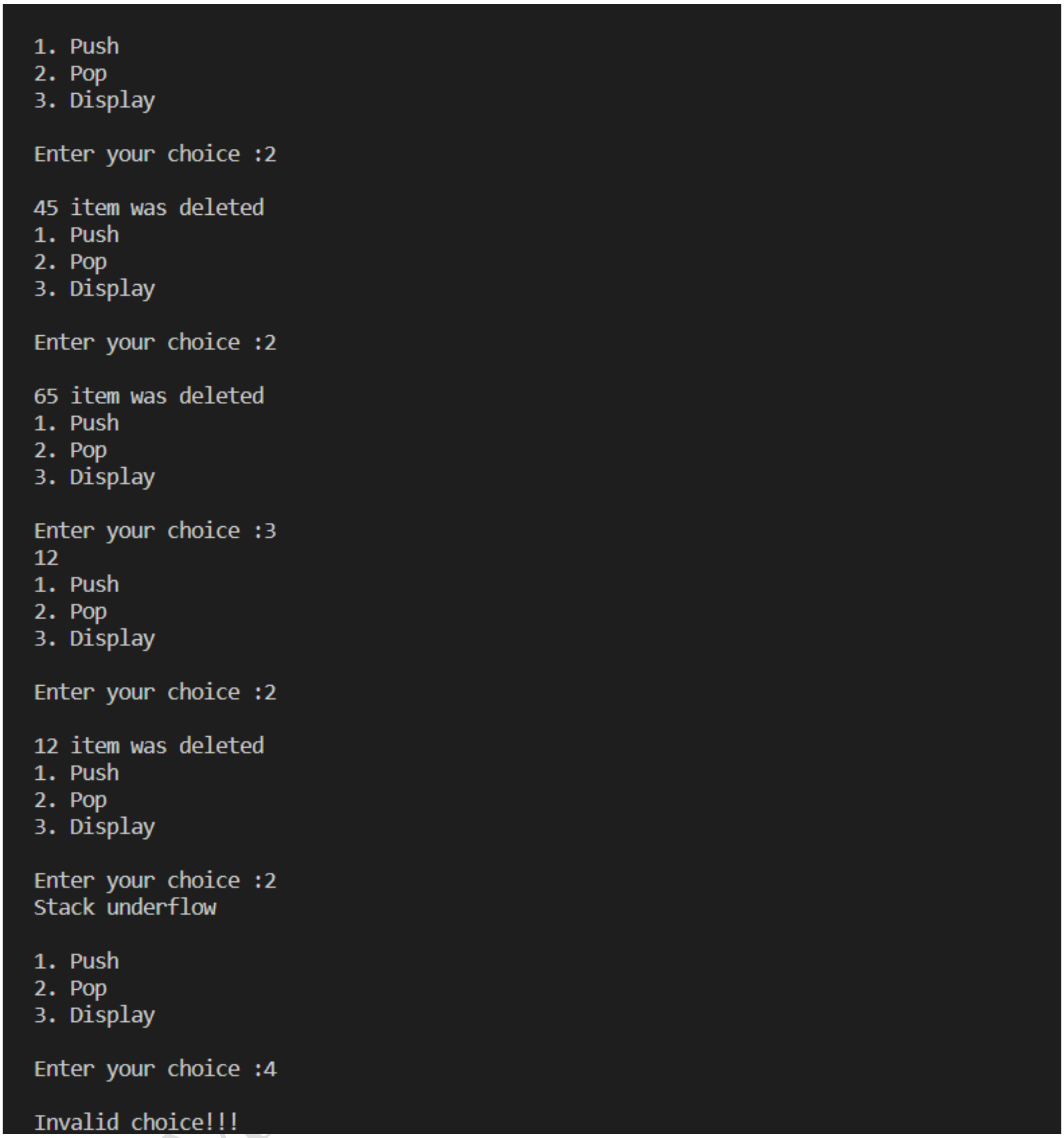
```
#include <stdio.h>
#include<stdlib.h>
#define STACK_SIZE 5
void push(int st[],int *top)
{
        int item;
        if(*top==STACK_SIZE-1)
                printf("Stack overflow\n");
        else
        {
                printf("\nEnter an item :");
                scanf("%d",&item);
                (*top)++;
                st[*top]=item;
        }
}
void pop(int st[],int *top)
{
        if(*top==-1)
                printf("Stack underflow\n");
        else
        {
                printf("\n%d item was deleted",st[(*top)--]);
        }
}
void display(int st[],int *top)
```

```
{
        int i;
        if(*top==-1)
                printf("Stack is empty\n");
        for(i=0;i<=*top;i++)
                printf("%d\t",st[i]);
}
void main()
{
        int st[10],top=-1, c,val_del;
        while(1)
        {
                printf("\n1. Push\n2. Pop\n3. Display\n");
                printf("\nEnter your choice :");
                scanf("%d",&c);
                switch(c)
                {
                        case 1: push(st,&top);
                                break;
                        case 2: pop(st,&top);
                                break;
                        case 3: display(st,&top);
                                break;
                        default: printf("\nInvalid choice!!!");
                                exit(0);
                }
        }
}
```

Output:



```
PS D:\jyothika\DST> cd "d:\jyothika\DST\" ; if ($?) { gcc 1.c -o 1 } ; if ($?) { .\1 }

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :12

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :65

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :45

1. Push
2. Pop
3. Display

Enter your choice :1
Stack overflow
```



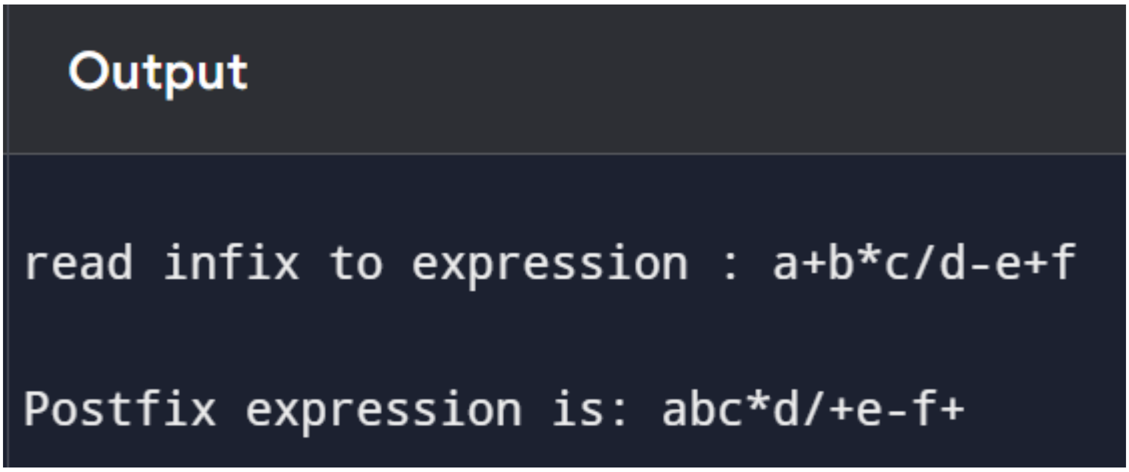
```
1. Push
2. Pop
3. Display

Enter your choice :2

45 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2

65 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :3
12
1. Push
2. Pop
3. Display

Enter your choice :2

12 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2
Stack underflow

1. Push
2. Pop
3. Display

Enter your choice :4

Invalid choice!!!
```

2.Infix to Postfix Conversions

Code:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#define size 20

struct stack
{
   int top;
   char data[size];
};

typedef struct stack STACK;

void push(STACK*S, char item)
{
   S-> data[++(S-> top)]=item;
}

char pop(STACK*S)
{
   return S-> data[(S->top)--];
}

int preced(char symbol)
{
   switch (symbol)
```

```
    {
        case '^' : return 3;
        case'*' :
        case'/' : return 2;
        case'+' :
        case'-' : return 1;
        default : return 0;
    }
}

void infixtopostfix(char infix[10], STACK*S)
{
    char postfix[10],symbol,temp;
    int a,b=0;
    for(a=0;infix[a]!='\0';a++)
    {
        symbol=infix[a];
        if(isalnum(symbol)){
            postfix[b++]=symbol;
        }else{
            switch(symbol){
                case'(': push(S,symbol);
                        break;
                case')': temp = pop(S);
                        while(temp!='(')
                        {
                            postfix[b++]=temp;
                            temp=pop(S);
                        }
                    break;
                case '+' :
                case '-' :
                case '*' :
                case '/' :
                case '^' : while (S->top != -1 && preced(S->data[S->top]) >= preced(symbol) && S->data[S->top] != '('){
                        postfix[b++] = pop(S);
                    }
                    push(S, symbol);
                    break;
            }
        }
    }
        while (S->top != -1) {
        postfix[b++] = pop(S);
    }
    postfix[b] = '\0';
    printf("\nPostfix expression is: %s\n", postfix);
}
int main ()
{
    char infix[10];
    STACK S;
    S.top=-1;
    printf("\nread infix to expression : ");
    scanf("%s",infix);
    infixtopostfix(infix,&S);

    return 0;

}
```



```
read infix to expression : a+b*c/d-e+f

Postfix expression is: abc*d/+e-f+
```

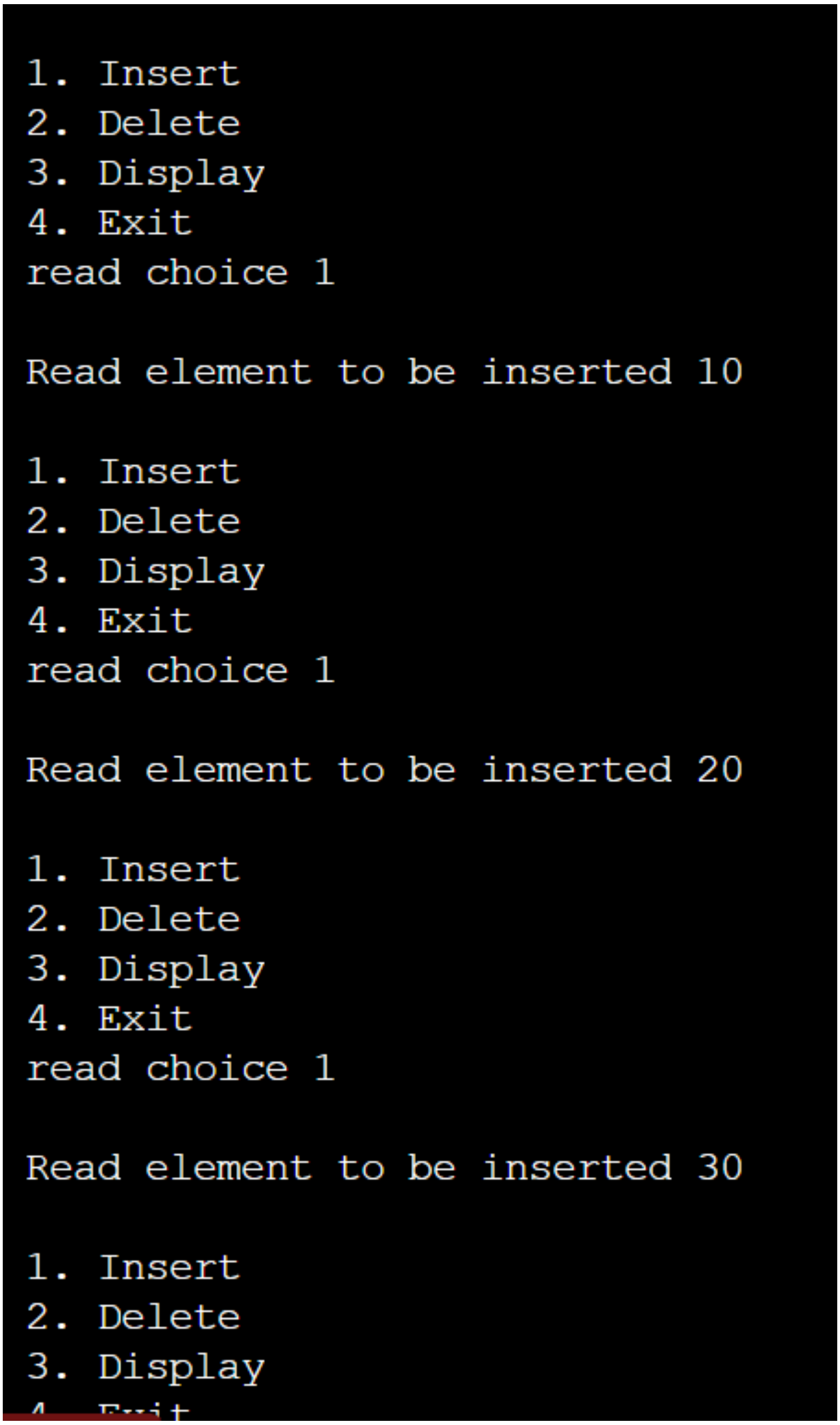
3.Queue Implementation

Code:

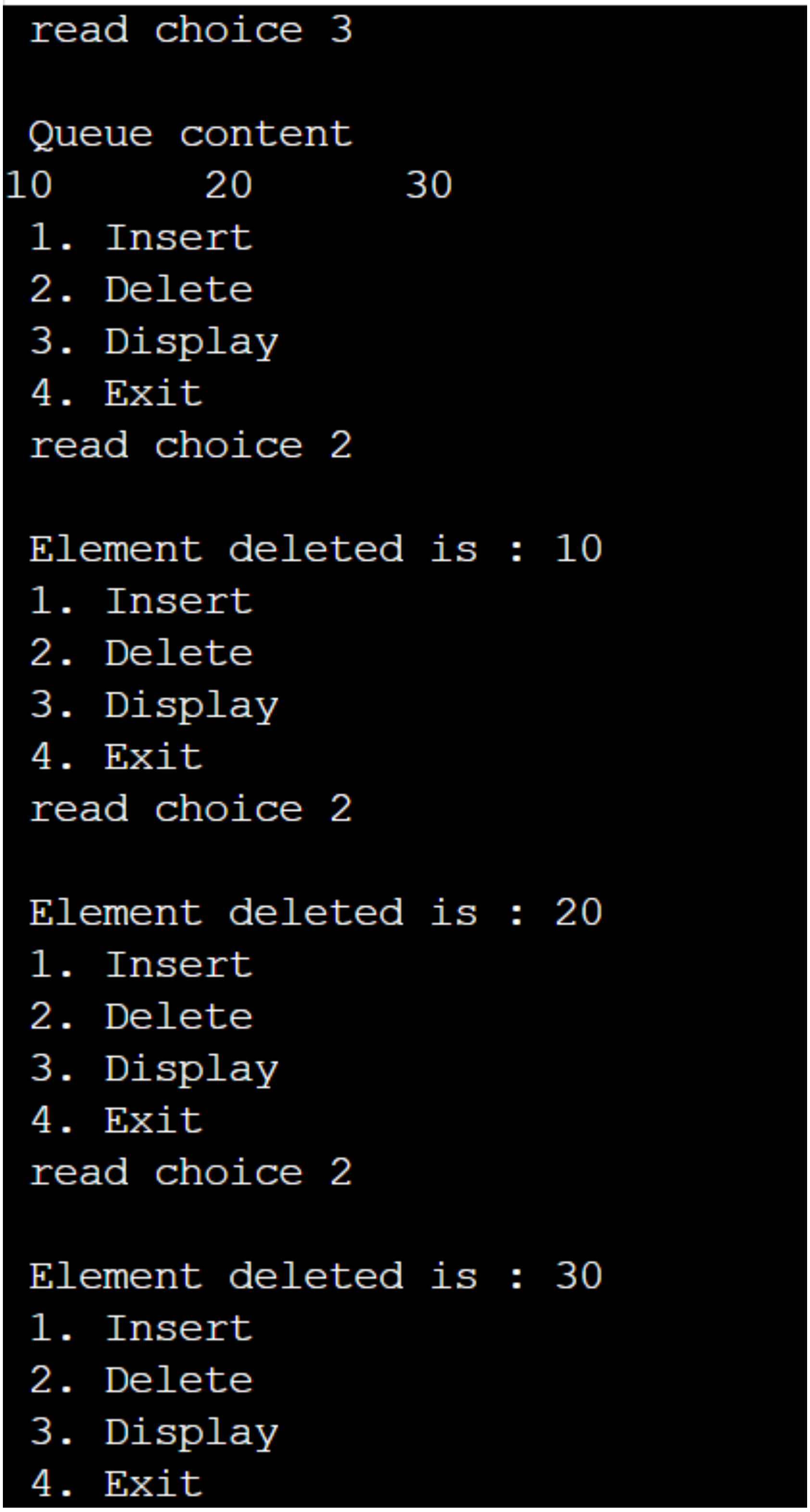
```
#include <stdio.h>

#include <stdlib.h>

#define SIZE 5


int front = -1 , rear = -1 ;

int q[SIZE];


void enqueue (int item)

{

    if(rear==SIZE-1)

    printf("\n Queue is full ");

    else{

        rear=rear+1;
        q[rear]=item;
        if(front==-1)
        front=front+1;
    }
}

void dequeue()
{
    int del;
    if(front==-1)
    printf("\n Queue is empty");
    else{
        del=q[front];
        printf("\n Element deleted is : %d",del);
        if(front==rear)
        {
            front=-1;rear=-1;
        }
        else{
            front=front+1;
        }
    }
}

void display()
{
    int i;
    if(front==-1)
    printf("\n Queue is empty");
    else{
        printf("\n Queue content \n");
        for (i=front;i<=rear;i++)
        printf("%d\t",q[i]);
    }
}

int main()
{
    int item,ch;
    for(;;)
    {
        printf("\n 1. Insert");
        printf("\n 2. Delete");
        printf("\n 3. Display");
        printf("\n 4. Exit");
        printf("\n read choice ");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1 : printf("\n Read element to be inserted ");
                    scanf("%d", & item);
                    enqueue(item);
                    break;

            case 2 : dequeue();
```

```
                break;

            case 3 : display();
                break;

            default : exit (0);
        }
    }
    return 0;
}
```

OUTPUT:



```
1. Insert
2. Delete
3. Display
4. Exit
read choice 1

Read element to be inserted 10

1. Insert
2. Delete
3. Display
4. Exit
read choice 1

Read element to be inserted 20

1. Insert
2. Delete
3. Display
4. Exit
read choice 1

Read element to be inserted 30

1. Insert
2. Delete
3. Display
4. Exit
```



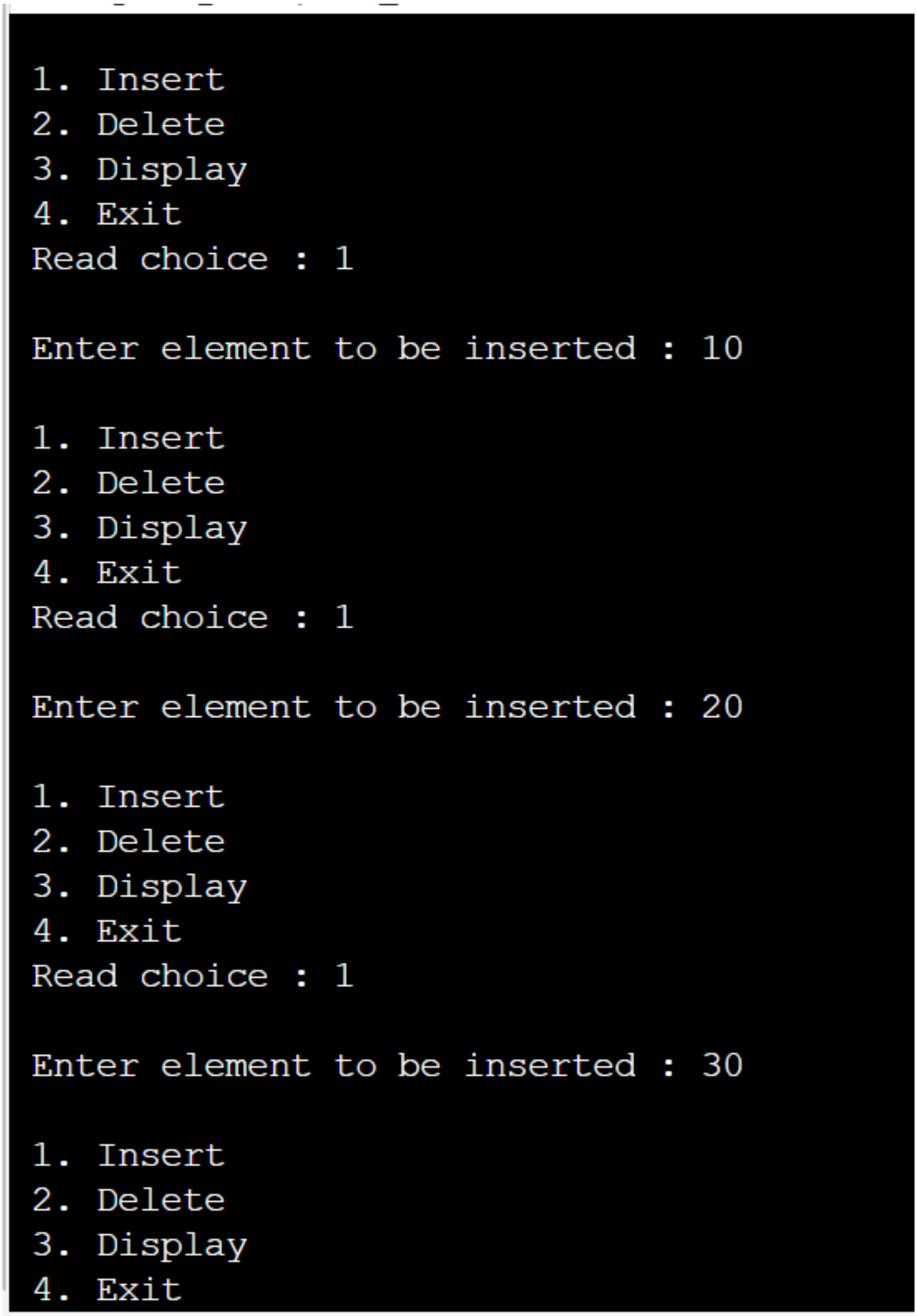
```
 read choice 3

 Queue content
10      20      30
 1. Insert
 2. Delete
 3. Display
 4. Exit
 read choice 2

 Element deleted is : 10
 1. Insert
 2. Delete
 3. Display
 4. Exit
 read choice 2

 Element deleted is : 20
 1. Insert
 2. Delete
 3. Display
 4. Exit
 read choice 2

 Element deleted is : 30
 1. Insert
 2. Delete
 3. Display
 4. Exit
```

4.Circular Queue Implementation

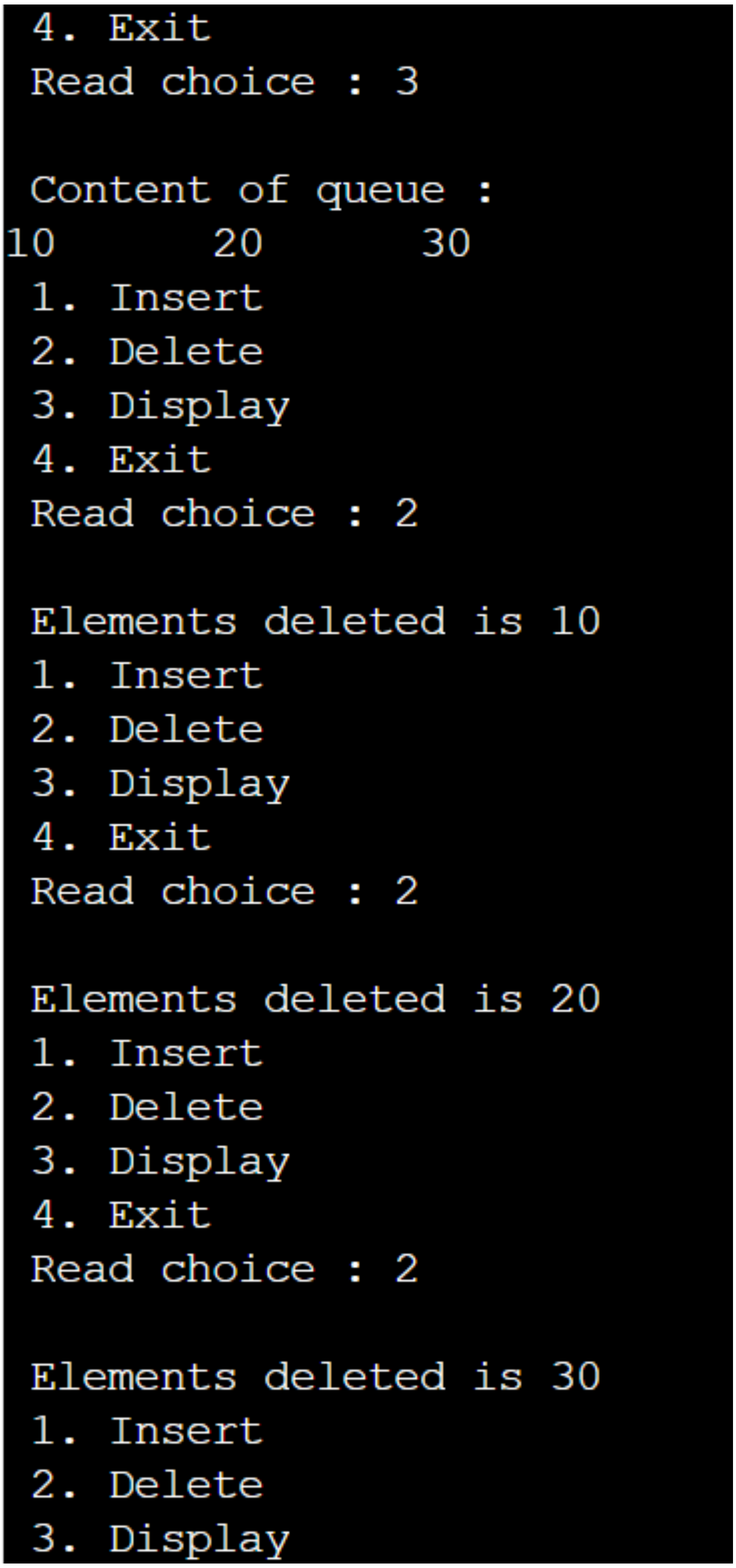
Code:

```
#include <stdio.h>
#include <stdlib.h>
#define size 5

int f=-1;
int r=-1;
int q[size];

void enqueue(int item)
{
   if(f==(r+1)%size)
   printf("Queue is full");
   else{
      r=(r+1)%size;
      q[r]=item;
      if(f==-1)
         f=f+1;
   }
}

void dequeue()
{
   if(f==-1)
   printf("Queue is empty");
   else{
      printf("\n Elements deleted is %d ",q[f]);
      if(f==r)
      {
            f=-1;
            r=-1;
        }
        else{
            f=(f+1)%size;
        }
    }
}

void display()
{
    int i;
    if(f==-1)
        printf("Queue is empty");
    else{
        printf("\n Content of queue : \n");
        for(i=f ; i!=r ; i=(i+1)%size)
        printf("%d \t",q[i]);
        printf("%d \t", q[r]);
    }
}

int main()
{
    int ch , item;
    for(;;)
    {
        printf("\n 1. Insert");
        printf("\n 2. Delete");

        printf("\n 3. Display");

        printf("\n 4. Exit");

        printf("\n Read choice : ");

        scanf("%d",&ch);


        switch(ch)

        {

          case 1 : printf("\n Enter element to be inserted : ");

                  scanf("%d",&item);

                  enqueue(item);

                  break;


          case 2 : dequeue();

                  break;


          case 3 : display();

                  break;


          default : exit(0);

        }

    }

    return 0;

}
```

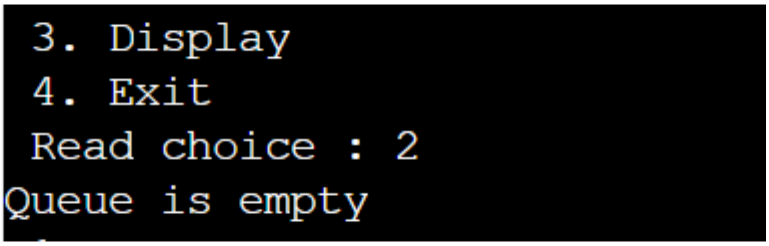
OUTPUT:



```
1. Insert
2. Delete
3. Display
4. Exit
Read choice : 1

Enter element to be inserted : 10

1. Insert
2. Delete
3. Display
4. Exit
Read choice : 1

Enter element to be inserted : 20

1. Insert
2. Delete
3. Display
4. Exit
Read choice : 1

Enter element to be inserted : 30

1. Insert
2. Delete
3. Display
4. Exit
```



```
 4. Exit
 Read choice : 3

 Content of queue :
10      20      30
 1. Insert
 2. Delete
 3. Display
 4. Exit
 Read choice : 2

 Elements deleted is 10
 1. Insert
 2. Delete
 3. Display
 4. Exit
 Read choice : 2

 Elements deleted is 20
 1. Insert
 2. Delete
 3. Display
 4. Exit
 Read choice : 2

 Elements deleted is 30
 1. Insert
 2. Delete
 3. Display
```



```
 3. Display
 4. Exit
 Read choice : 2
Queue is empty
```

5.1) WAP to Implement Singly Linked List with following operations

(10 Marks)

a) Create a linked list.

b) Insertion of a node at first position, at any position and at end of list.

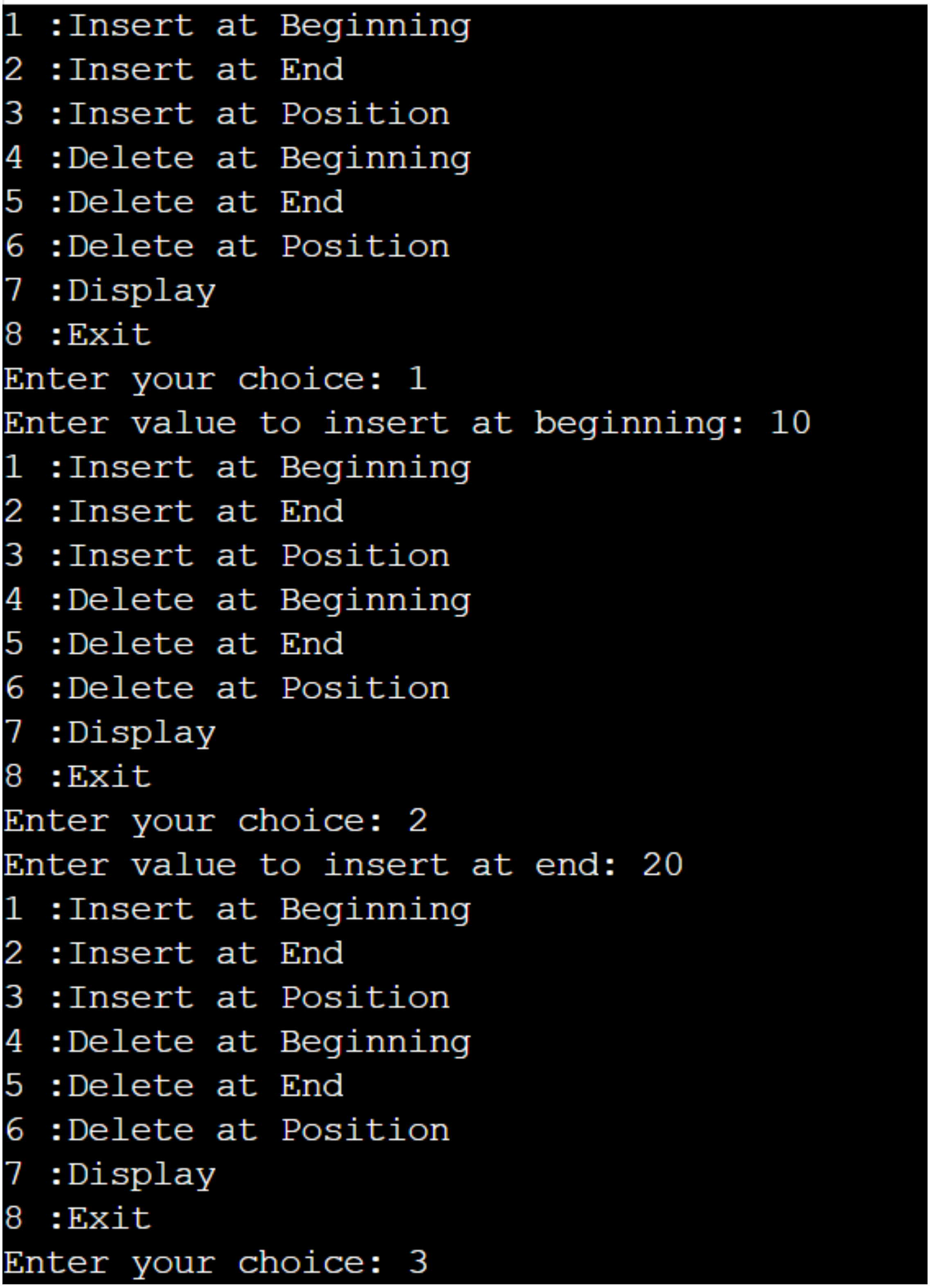
c) Display the contents of the linked list.

Code:

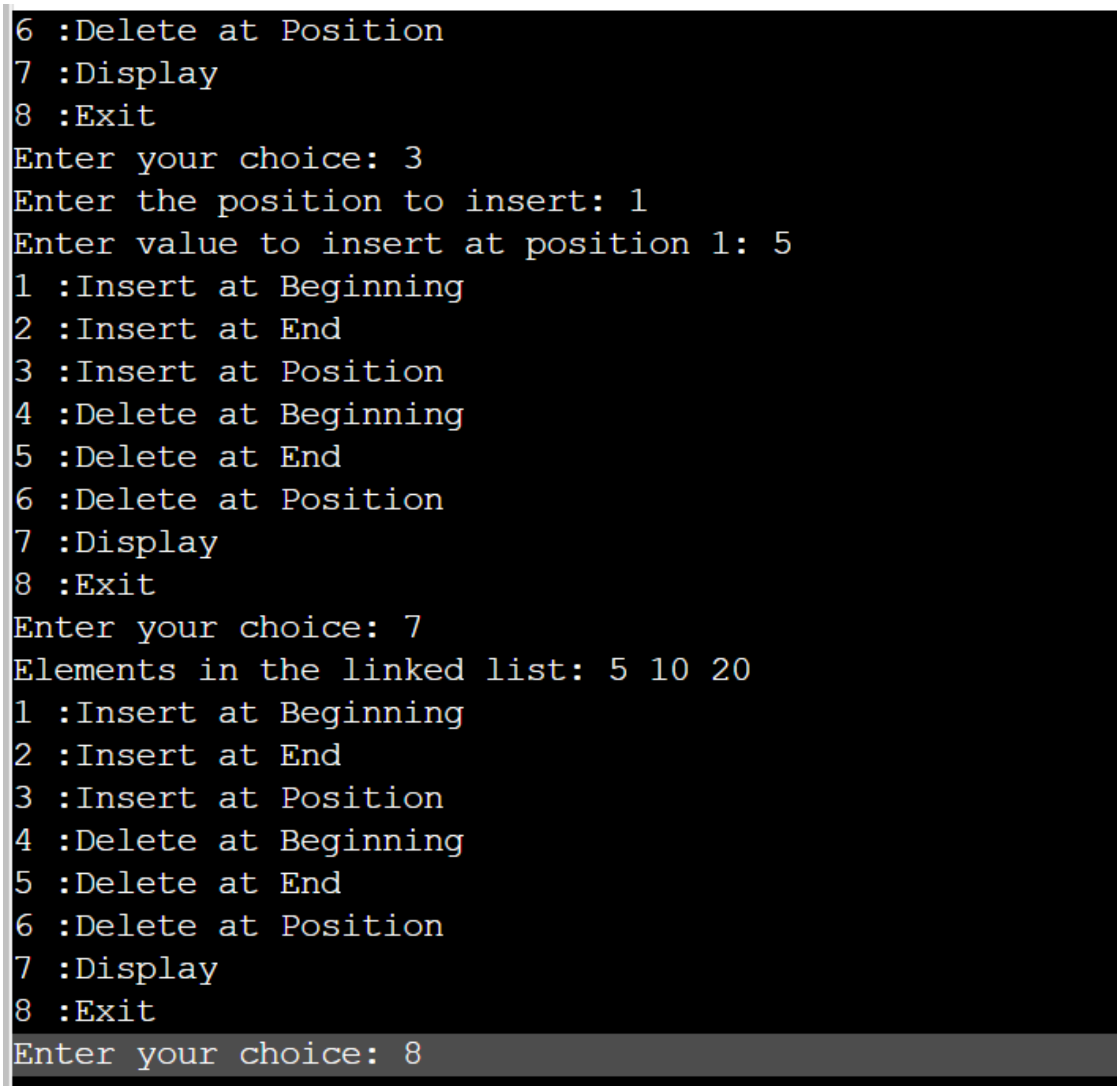
```
#include <stdio.h>

#include <stdlib.h>


struct Node {

    int data;

    struct Node *link;

};


typedef struct Node node;

node *start = NULL;


void insertAtBeginning() {

    node *new1 = (node*)malloc(sizeof(node));

    if (new1 == NULL) {

        printf("Memory allocation failed.\n");

        return;

    }


    printf("Enter value to insert at beginning: ");

    scanf("%d", &new1->data);


    new1->link = start;

    start = new1;
```

```
}

void insertAtEnd() {
    node *new1 = (node*)malloc(sizeof(node));
    if (new1 == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }

    printf("Enter value to insert at end: ");
    scanf("%d", &new1->data);

    new1->link = NULL;

    if (start == NULL) {
        start = new1;
    } else {
        node *temp = start;
        while (temp->link != NULL) {
            temp = temp->link;
        }
        temp->link = new1;
    }
}

void insertAtPosition() {
    int pos;
    printf("Enter the position to insert: ");
    scanf("%d", &pos);

    if (pos < 1) {
        printf("Invalid position. Position should be >= 1.\n");
        return;
    }

    node *new1 = (node*)malloc(sizeof(node));
    if (new1 == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }

    printf("Enter value to insert at position %d: ", pos);
    scanf("%d", &new1->data);

    if (pos == 1) {
        new1->link = start;
        start = new1;
    } else {
        node *temp = start;
        for (int i = 1; i < pos - 1; i++) {
            if (temp == NULL) {
                printf("Position out of range.\n");
                free(new1);
                return;
            }
            temp = temp->link;
        }

        new1->link = temp->link;
        temp->link = new1;
    }
}

void deleteAtBeginning() {
    if (start == NULL) {
        printf("The list is empty, nothing to delete.\n");
        return;
    }

    node *temp = start;
    start = start->link;
    free(temp);

    printf("Node deleted from the beginning.\n");
}

void deleteAtEnd() {
    if (start == NULL) {
        printf("The list is empty, nothing to delete.\n");
        return;
    }

    if (start->link == NULL) {
        free(start);
        start = NULL;
        printf("Node deleted from the end.\n");
        return;
    }

    node *temp = start;
    while (temp->link != NULL && temp->link->link != NULL) { // Traverse to the second last node
        temp = temp->link;
    }

    free(temp->link);
    temp->link = NULL;
    printf("Node deleted from the end.\n");
}

void deleteAtPosition() {
    int pos;
    printf("Enter the position to delete: ");
    scanf("%d", &pos);



    if (pos < 1 || start == NULL) {
        printf("Invalid position or list is empty.\n");
        return;
    }

    if (pos == 1) {
        node *temp = start;
        start = start->link;
        free(temp);
        printf("Node deleted from position 1.\n");
        return;
    }

    node *temp = start;
    for (int i = 1; i < pos - 1; i++) {
        if (temp == NULL || temp->link == NULL) {
            printf("Position out of range.\n");
            return;
        }
        temp = temp->link;
    }
    node *delNode = temp->link;
    temp->link = temp->link->link;
    free(delNode);

    printf("Node deleted from position %d.\n", pos);
}

void display() {
    node *temp;
    if (start == NULL) {
        printf("Linked list is empty.\n");
        return;
    }

    printf("Elements in the linked list: ");
    temp = start;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->link;
    }
    printf("\n");
}

int main() {
    int ch;

    while (1) {
        printf("1 :Insert at Beginning  \n2 :Insert at End  \n3 :Insert at Position  \n4 :Delete at Beginning
\n5 :Delete at End  \n6 :Delete at Position  \n7 :Display  \n8 :Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                insertAtBeginning();
                break;
            case 2:
                insertAtEnd();
                break;
            case 3:
                insertAtPosition();
                break;
            case 4:
                deleteAtBeginning();
                break;
            case 5:
                deleteAtEnd();
                break;
            case 6:
                deleteAtPosition();
                break;
            case 7:
                display();
                break;
            case 8:
                exit(0);  // Exit the program
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

OUTPUT:



```
1 :Insert at Beginning
2 :Insert at End
3 :Insert at Position
4 :Delete at Beginning
5 :Delete at End
6 :Delete at Position
7 :Display
8 :Exit
Enter your choice: 1
Enter value to insert at beginning: 10
1 :Insert at Beginning
2 :Insert at End
3 :Insert at Position
4 :Delete at Beginning
5 :Delete at End
6 :Delete at Position
7 :Display
8 :Exit
Enter your choice: 2
Enter value to insert at end: 20
1 :Insert at Beginning
2 :Insert at End
3 :Insert at Position
4 :Delete at Beginning
5 :Delete at End
6 :Delete at Position
7 :Display
8 :Exit
Enter your choice: 3
```



```
6 :Delete at Position
7 :Display
8 :Exit
Enter your choice: 3
Enter the position to insert: 1
Enter value to insert at position 1: 5
1 :Insert at Beginning
2 :Insert at End
3 :Insert at Position
4 :Delete at Beginning
5 :Delete at End
6 :Delete at Position
7 :Display
8 :Exit
Enter your choice: 7
Elements in the linked list: 5 10 20
1 :Insert at Beginning
2 :Insert at End
3 :Insert at Position
4 :Delete at Beginning
5 :Delete at End
6 :Delete at Position
7 :Display
8 :Exit
Enter your choice: 8
```

6.WAP to Implement Singly Linked List with following operations

(10 Marks)

a) Create a linked list.

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

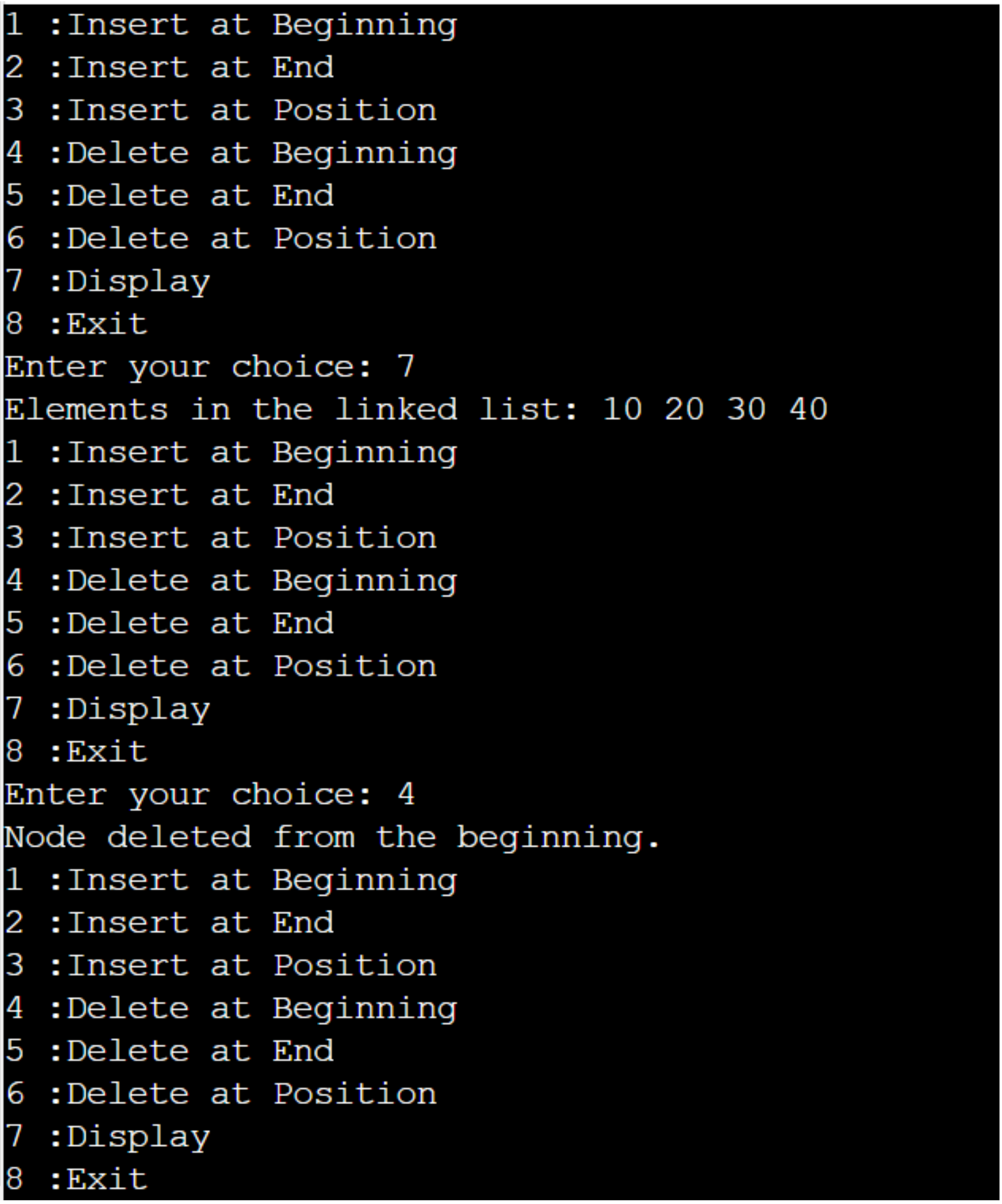
c) Display the contents of the linked list.

Code:

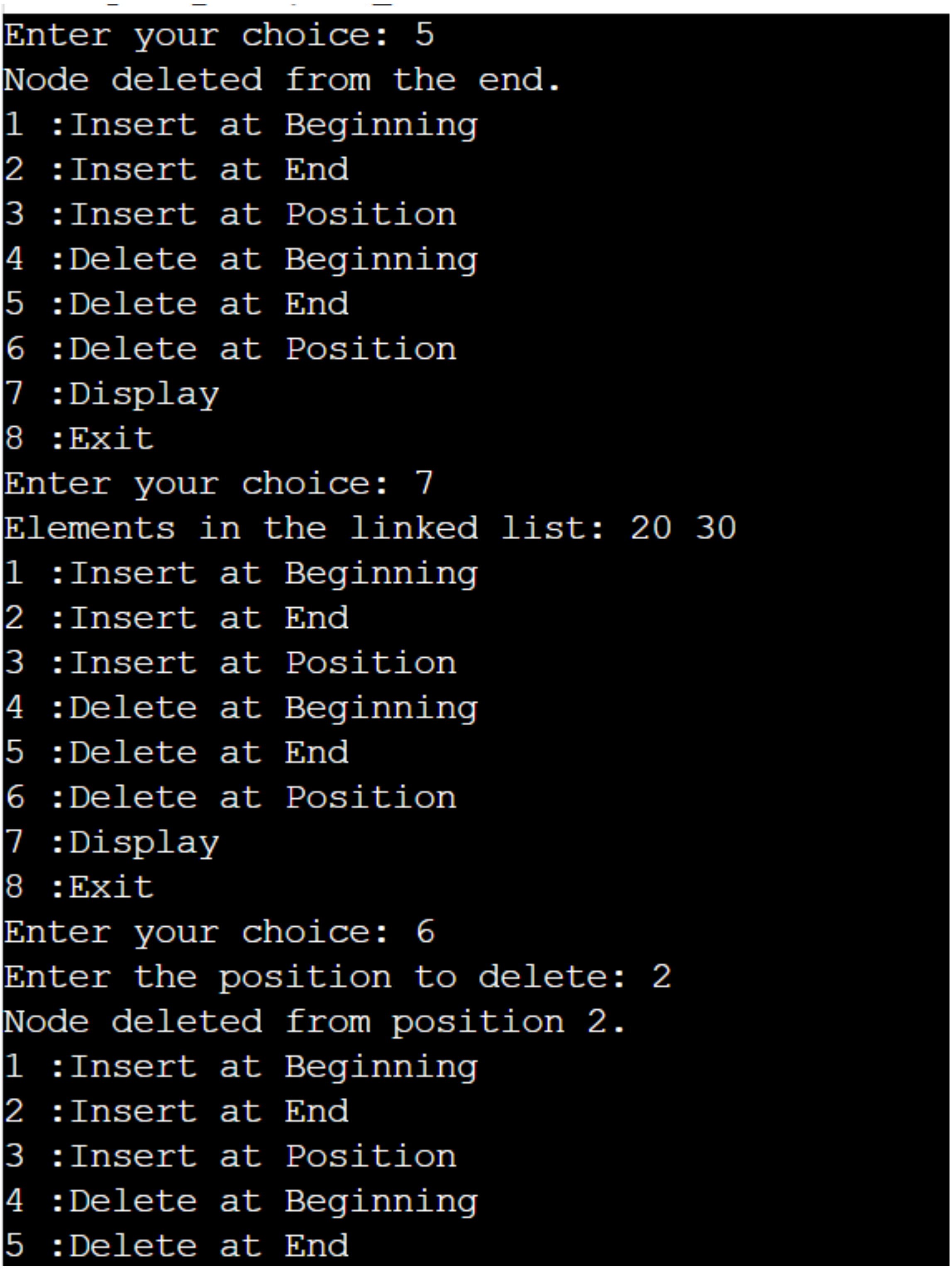
```
#include <stdio.h>

#include <stdlib.h>
```

```
struct Node {
    int data;
    struct Node *link;
};

typedef struct Node node;
node *start = NULL;

void insertAtBeginning() {
    node *new1 = (node*)malloc(sizeof(node));
    if (new1 == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }

    printf("Enter value to insert at beginning: ");
    scanf("%d", &new1->data);

    new1->link = start;
    start = new1;
}

void insertAtEnd() {
    node *new1 = (node*)malloc(sizeof(node));
    if (new1 == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }

    printf("Enter value to insert at end: ");
    scanf("%d", &new1->data);
    new1->link = NULL;

    if (start == NULL) {
        start = new1;
    } else {
        node *temp = start;
        while (temp->link != NULL) {
            temp = temp->link;
        }
        temp->link = new1;
    }
}

void insertAtPosition() {
    int pos;
    printf("Enter the position to insert: ");
    scanf("%d", &pos);

    if (pos < 1) {
        printf("Invalid position. Position should be >= 1.\n");
        return;
    }

    node *new1 = (node*)malloc(sizeof(node));
    if (new1 == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }

    printf("Enter value to insert at position %d: ", pos);
    scanf("%d", &new1->data);
    if (pos == 1) {
        new1->link = start;
        start = new1;
    } else {
        node *temp = start;
        for (int i = 1; i < pos - 1; i++) {
            if (temp == NULL) {
                printf("Position out of range.\n");
                free(new1);
                return;
            }
            temp = temp->link;
        }

        new1->link = temp->link;
        temp->link = new1;
    }
}

void deleteAtBeginning() {
    if (start == NULL) {
        printf("The list is empty, nothing to delete.\n");
        return;
    }

    node *temp = start;
    start = start->link;
    free(temp);

    printf("Node deleted from the beginning.\n");
}
```

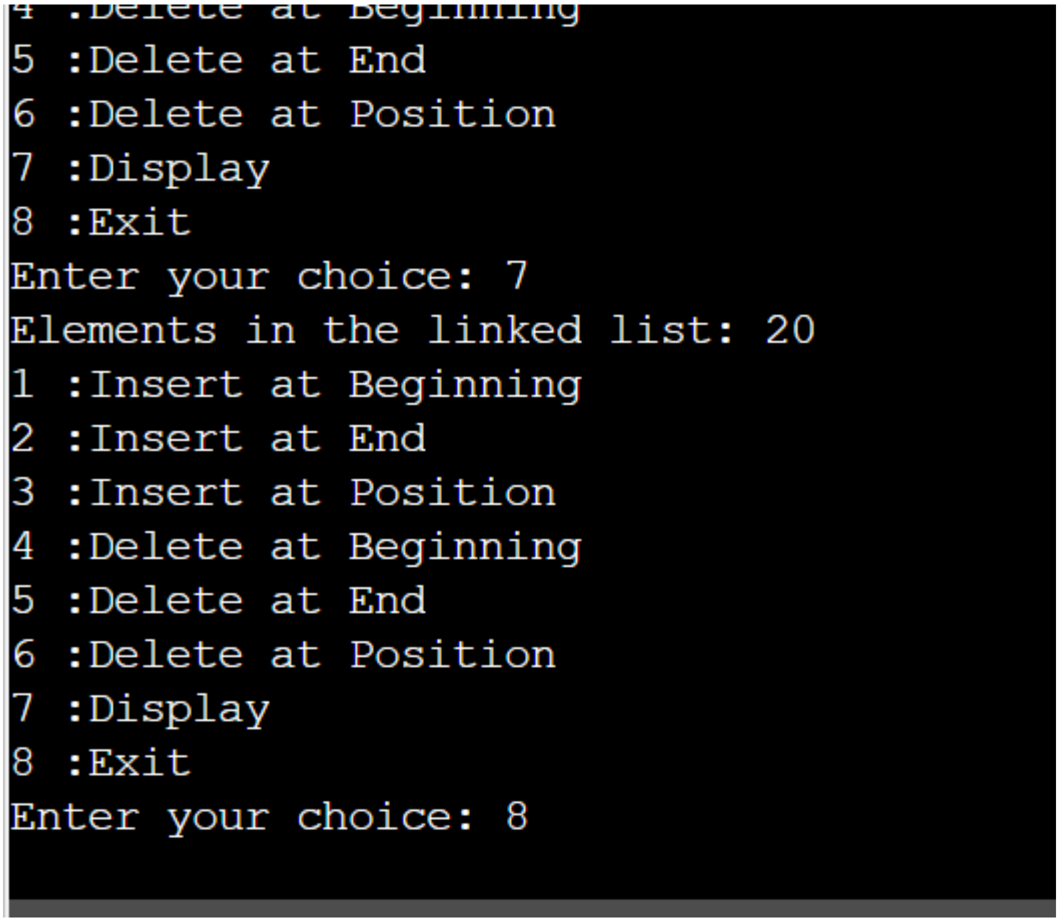
```
void deleteAtEnd() {
    if (start == NULL) {
        printf("The list is empty, nothing to delete.\n");
        return;
    }

    if (start->link == NULL) {
        free(start);
        start = NULL;
        printf("Node deleted from the end.\n");
        return;
    }

    node *temp = start;
    while (temp->link != NULL && temp->link->link != NULL) { // Traverse to the second last node
        temp = temp->link;
    }

    free(temp->link);
    temp->link = NULL;

    printf("Node deleted from the end.\n");
}

void deleteAtPosition() {
    int pos;
    printf("Enter the position to delete: ");
    scanf("%d", &pos);



    if (pos < 1 || start == NULL) {
        printf("Invalid position or list is empty.\n");
        return;
    }

    if (pos == 1) {
        node *temp = start;
        start = start->link;
        free(temp);
        printf("Node deleted from position 1.\n");
        return;
    }

    node *temp = start;
    for (int i = 1; i < pos - 1; i++) {
        if (temp == NULL || temp->link == NULL) {
            printf("Position out of range.\n");
            return;
        }
        temp = temp->link;
    }

    node *delNode = temp->link;
    temp->link = temp->link->link;
    free(delNode);

    printf("Node deleted from position %d.\n", pos);
}

void display() {
    node *temp;
    if (start == NULL) {
        printf("Linked list is empty.\n");
        return;
    }

    printf("Elements in the linked list: ");
    temp = start;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->link;
    }
    printf("\n");
}

int main() {
    int ch;

    while (1) {
        printf("1 :Insert at Beginning  \n2 :Insert at End  \n3 :Insert at Position  \n4 :Delete at Beginning  \n5 :Delete at End  \n6 :Delete at Position  \n7 :Display  \n8 :Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                insertAtBeginning();
                break;
            case 2:
                insertAtEnd();
                break;
            case 3:
                insertAtPosition();
                break;
            case 4:
                deleteAtBeginning();
                break;
            case 5:
                deleteAtEnd();
                break;
            case 6:
                deleteAtPosition();
                break;
            case 7:
                display();
                break;
            case 8:
                exit(0);  // Exit the program
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```



```
1 :Insert at Beginning
2 :Insert at End
3 :Insert at Position
4 :Delete at Beginning
5 :Delete at End
6 :Delete at Position
7 :Display
8 :Exit
Enter your choice: 7
Elements in the linked list: 10 20 30 40
1 :Insert at Beginning
2 :Insert at End
3 :Insert at Position
4 :Delete at Beginning
5 :Delete at End
6 :Delete at Position
7 :Display
8 :Exit
Enter your choice: 4
Node deleted from the beginning.
1 :Insert at Beginning
2 :Insert at End
3 :Insert at Position
4 :Delete at Beginning
5 :Delete at End
6 :Delete at Position
7 :Display
8 :Exit
```



```
Enter your choice: 5
Node deleted from the end.
1 :Insert at Beginning
2 :Insert at End
3 :Insert at Position
4 :Delete at Beginning
5 :Delete at End
6 :Delete at Position
7 :Display
8 :Exit
Enter your choice: 7
Elements in the linked list: 20 30
1 :Insert at Beginning
2 :Insert at End
3 :Insert at Position
4 :Delete at Beginning
5 :Delete at End
6 :Delete at Position
7 :Display
8 :Exit
Enter your choice: 6
Enter the position to delete: 2
Node deleted from position 2.
1 :Insert at Beginning
2 :Insert at End
3 :Insert at Position
4 :Delete at Beginning
5 :Delete at End
```



```
4 :Delete at Beginning
5 :Delete at End
6 :Delete at Position
7 :Display
8 :Exit
Enter your choice: 7
Elements in the linked list: 20
1 :Insert at Beginning
2 :Insert at End
3 :Insert at Position
4 :Delete at Beginning
5 :Delete at End
6 :Delete at Position
7 :Display
8 :Exit
Enter your choice: 8
```

7.)i) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists. (5 Marks)

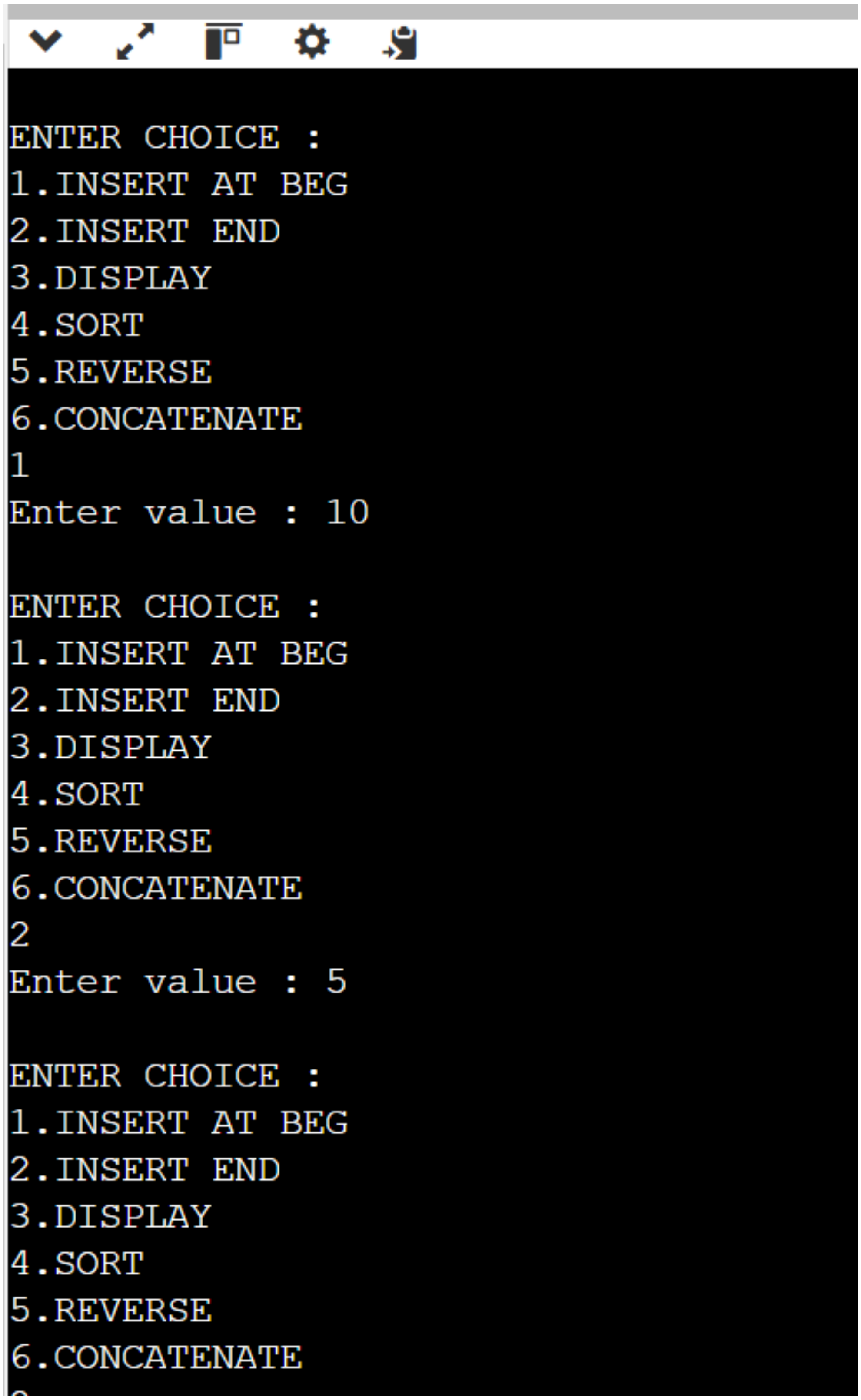
Code:

```
#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

   int data;

   struct node *link;

}NODE,NODE1;

NODE *head=NULL;

NODE1 *head1=NULL;

NODE* createnode()

{

   NODE *ptr = (NODE*)malloc(sizeof(NODE));
```

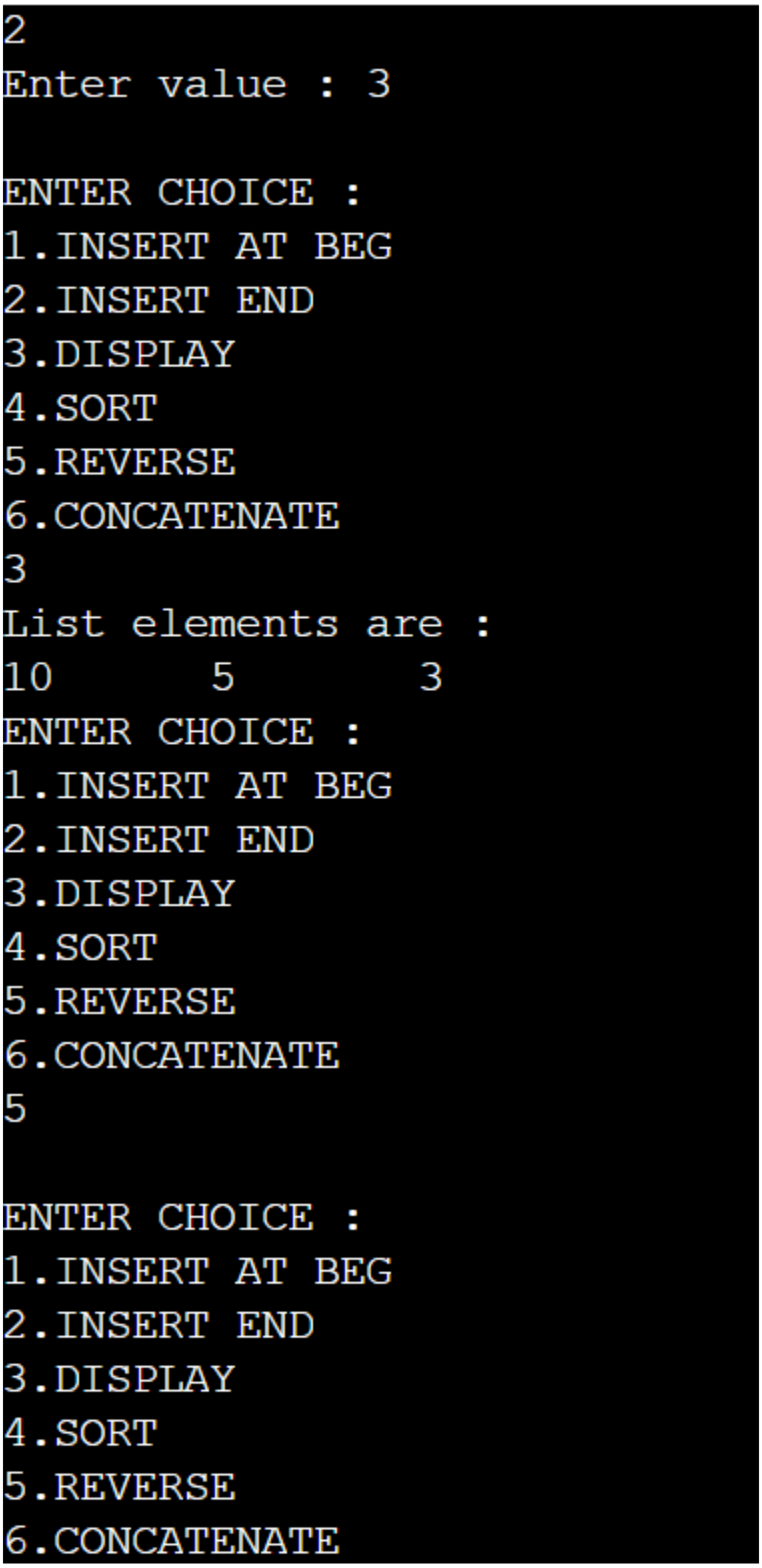
```
    int item;
    printf("Enter value : ");
    scanf("%d",&item);
    if(ptr==NULL)
    {
        printf("Memory not allocated");
    }
    else
    {
        ptr->data = item;
        ptr->link=NULL;
    }
    return ptr;

}

void insert_beg()
{
    NODE *ptr=createnode();
    if(head==NULL)
    {
        head=ptr;
    }
    else if(head!=NULL)
    {
        ptr->link=head;
        head=ptr;
    }
}
void insert_end()
{
    NODE *ptr=createnode();
    NODE *temp;
    temp = head;
    while(temp->link!=NULL)
    {
        temp=temp->link;
    }
    temp->link=ptr;
}
void display()
{
    NODE *temp;
    temp=head;
    if(head==NULL)
    {
        printf("List is empty");
        return;
    }
    else
    {
        printf("List elements are : \n");
    }
    while(temp!=NULL)
    {
        printf("%d\t",temp->data);
        temp = temp->link;
    }
}
void sort()
{
    NODE *start = head;
    while(start!=NULL)
    {
        NODE *temp=start->link;
        while(temp!=NULL)
        {
            if(start->data > temp->data)
            {
                int x=start->data;
                start->data=temp->data;
                temp->data=x;
            }
            temp=temp->link;
        }
        start=start->link;
    }

}
void reverse()
{
    NODE* prev=NULL;
    NODE* next = NULL;
    NODE* curr = head;
    while(curr!=NULL)
    {
        next=curr->link;
        curr->link=prev;
        prev = curr;
        curr=  next;
    }
    head = prev;
}
NODE1* createnode1()
{
    NODE1 *ptr = (NODE1*)malloc(sizeof(NODE1));
    int item;
    printf("Enter value FOR NODE 2 : ");
    scanf("%d",&item);
    if(ptr==NULL)
    {
        printf("Memory not allocated");
    }
    else
    {
        ptr->data = item;
        ptr->link=NULL;
    }
    return ptr;

}
void insert_beg1()
{
    NODE1 *ptr=createnode1();
    if(head1==NULL)
    {
        head1=ptr;
    }
    else if(head1!=NULL)
    {
        ptr->link=head1;
        head1=ptr;
    }
}
void insert_end1()
{
    NODE1 *ptr=createnode1();
    NODE1 *temp;
    temp = head1;
    while(temp->link!=NULL)
    {
        temp=temp->link;
    }
    temp->link=ptr;
}
void display1()
{
    NODE1 *temp;
    temp=head1;
    if(head1==NULL)
    {
        printf("List is empty");
        return;
    }
    else
    {
        printf("List elements are : \n");
    }
    while(temp!=NULL)
    {
        printf("%d\t",temp->data);
        temp = temp->link;
    }
}
void extra()
{
    int choice=0;
    while(choice<=5)
    {
        printf("\nENTER CHOICE : \n1.INSERT AT BEG\n2.INSERT END\n3.DISPLAY\n4.CONCATENATE\n");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:insert_beg1();

                    break;

            case 2:insert_end1();

                    break;

            case 3:display1();

                    break;

            case 4:concatenate();

                    break;

        }


    }

}

void concatenate()

{

   NODE *temp = head;

   NODE *temp1=head1;

    while(temp->link!=NULL)

    {

        temp=temp->link;

    }

    temp->link=head1;

    temp=head;

    while(temp!=NULL)

    {

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

        temp=temp->link;

    }
```

```
}

void main()

{

    int choice=0;



    while(choice<=5)

    {

        printf("\nENTER CHOICE : \n1.INSERT AT BEG\n2.INSERT
END\n3.DISPLAY\n4.SORT\n5.REVERSE\n6.CONCATENATE\n");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:insert_beg();

                    break;

            case 2:insert_end();

                    break;

            case 3:display();

                    break;

            case 4:sort();

                    break;

            case 5:reverse();

                    break;

            case 6:printf("Enter node 2 \n");

                    extra();

                    break;

        }

    }

}
```

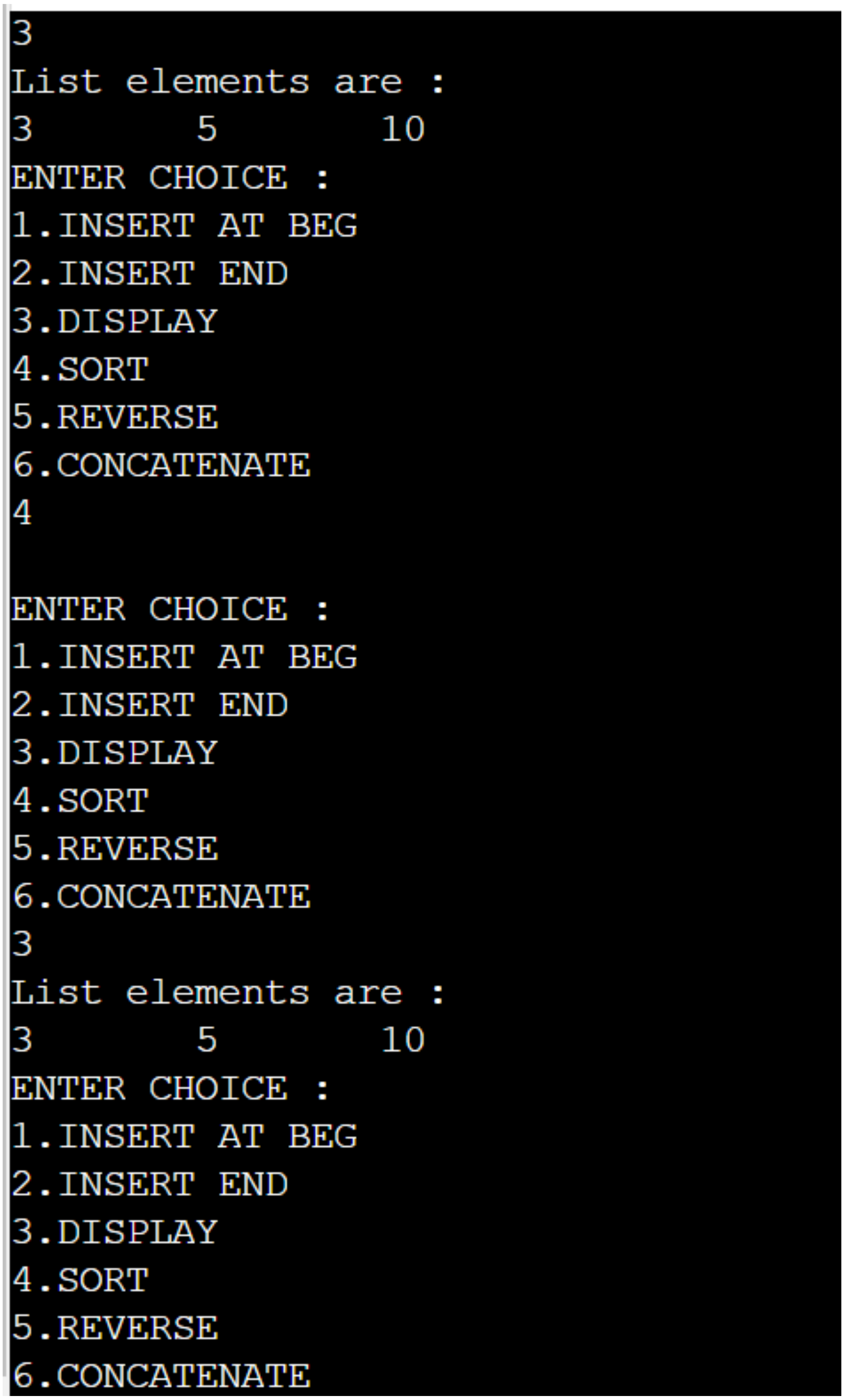
OUTPUT:



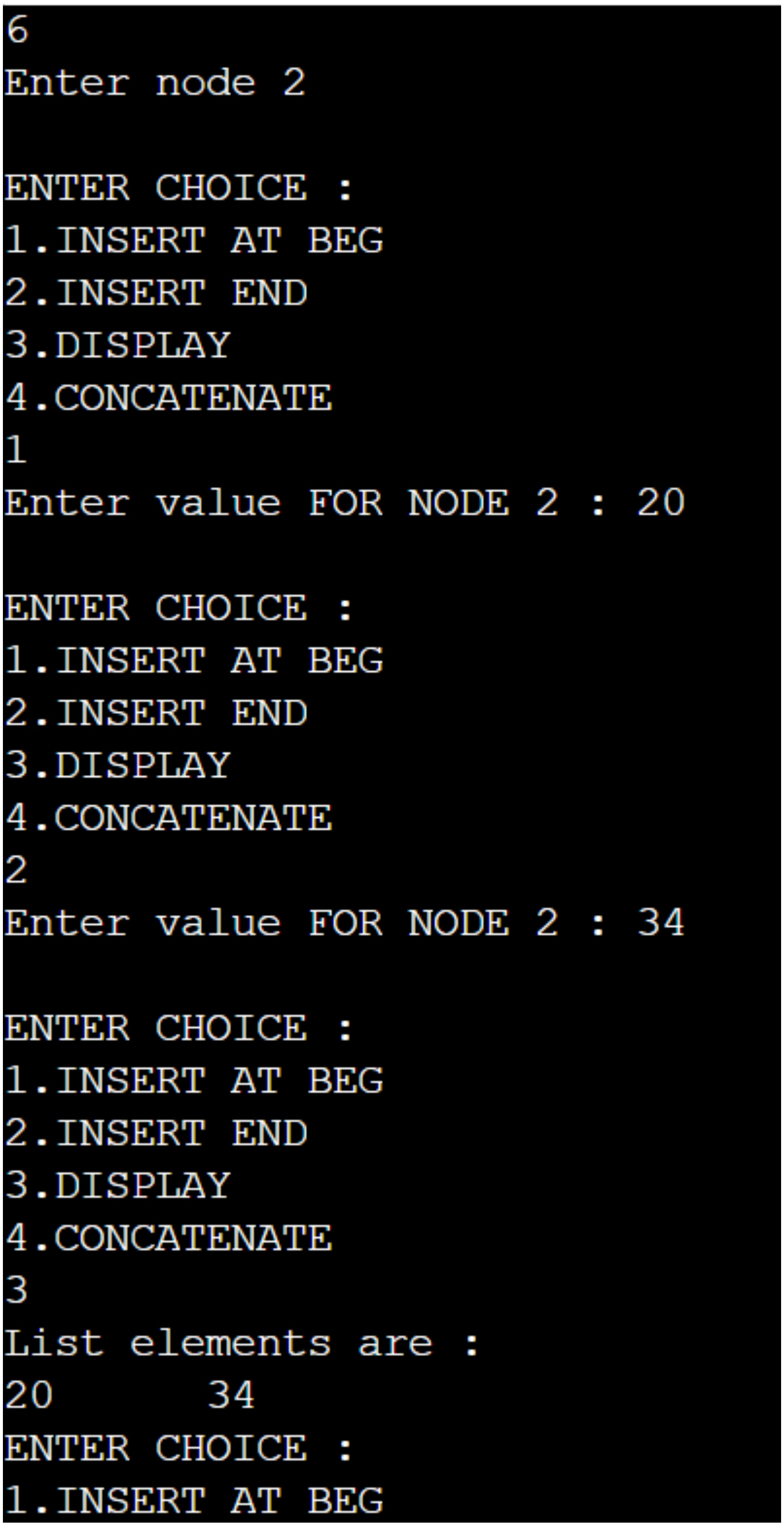
```

ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.SORT
5.REVERSE
6.CONCATENATE
1
Enter value : 10

ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.SORT
5.REVERSE
6.CONCATENATE
2
Enter value : 5

ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.SORT
5.REVERSE
6.CONCATENATE
```



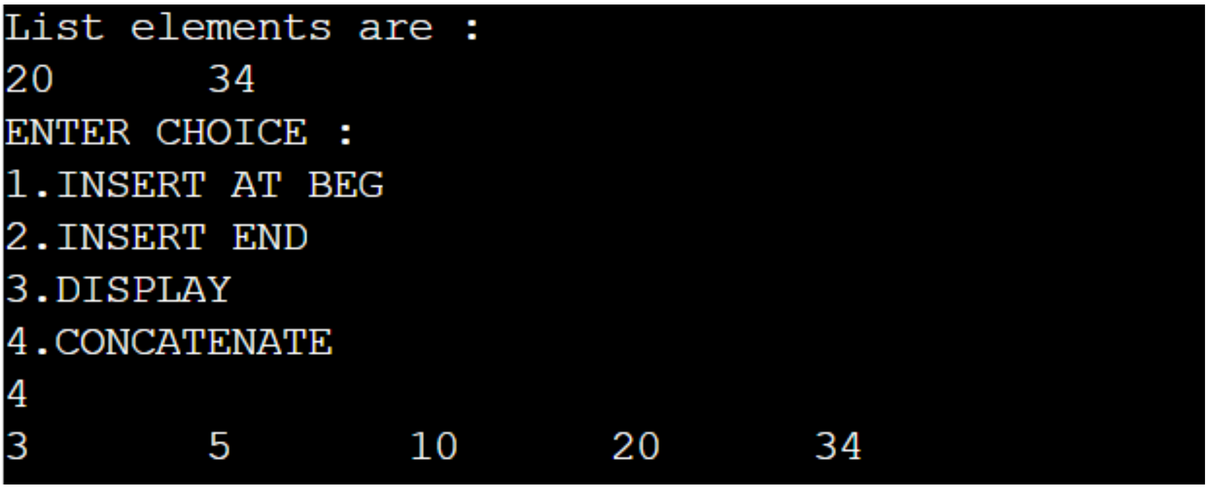
```
2
Enter value : 3

ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.SORT
5.REVERSE
6.CONCATENATE
3
List elements are :
10      5       3
ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.SORT
5.REVERSE
6.CONCATENATE
5

ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.SORT
5.REVERSE
6.CONCATENATE
```



```
3
List elements are :
3        5        10
ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.SORT
5.REVERSE
6.CONCATENATE
4

ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.SORT
5.REVERSE
6.CONCATENATE
3
List elements are :
3        5        10
ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.SORT
5.REVERSE
6.CONCATENATE
```



```
6
Enter node 2

ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.CONCATENATE
1
Enter value FOR NODE 2 : 20

ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.CONCATENATE
2
Enter value FOR NODE 2 : 34

ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.CONCATENATE
3
List elements are :
20      34
ENTER CHOICE :
1.INSERT AT BEG
```



```
List elements are :
20        34
ENTER CHOICE :
1.INSERT AT BEG
2.INSERT END
3.DISPLAY
4.CONCATENATE
4
3         5         10        20        34
```

ii)WAP to Implement Single Link List to simulate Stack & Queue Operations.   (5 Marks)

Code:

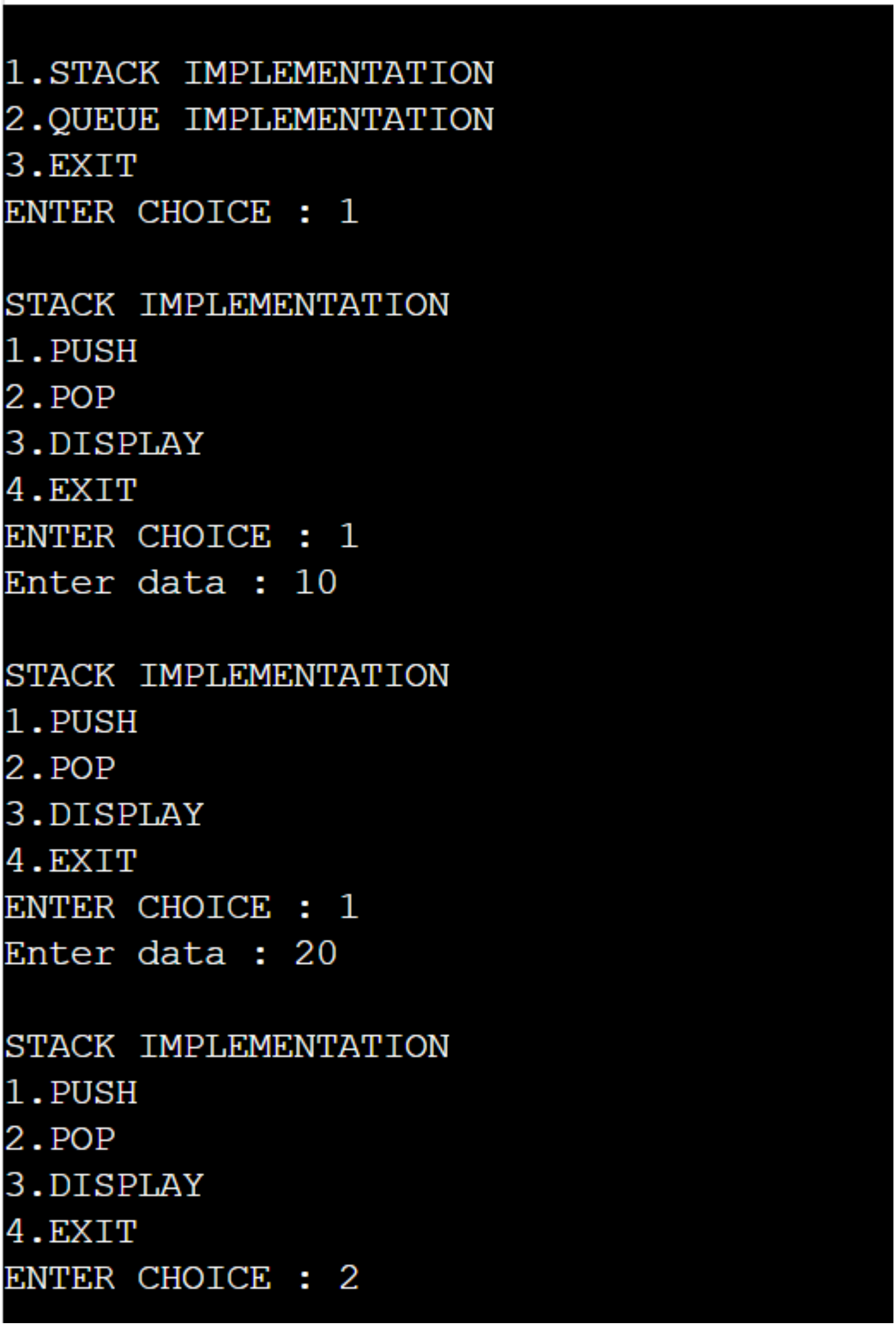
```
#include<stdio.h>

#include<stdlib.h>


int count1=0,count2=0,s=3;

typedef struct node

{

   int data;

   struct node *link;

}NODE;

NODE *head=NULL,*head1=NULL;


NODE* createnode()

{

   int a;

   NODE *ptr=  (NODE*)malloc(sizeof(NODE));

   printf("Enter data : ");

   scanf("%d",&a);

   if(ptr==NULL)

   {

      printf("Memory not allocated");

   }
```

```
    else
    {
        ptr->data=a;
        ptr->link=NULL;
    }
    return ptr;
}
void push()
{
     if(count1==s)
    {
        printf("\nStack Overflow");
        return;
    }
    NODE *ptr=createnode();
    if(head==NULL)
    {
        head=ptr;
        count1++;
        return;
    }
    NODE *temp=head;
    while(temp->link!=NULL)
    {
        temp=temp->link;
    }
    temp->link=ptr;
    count1++;
}
void pop()
{
    if(count1==0)
    {
        printf("\nStack Underflow");
        return;
    }
    NODE *temp=head;
    NODE *prev;
    if(head->link==NULL)
    {
        free(head);
        head=NULL;
        count1--;
        return;
    }
    while(temp->link!=NULL)
    {
        prev=temp;
        temp=temp->link;
    }
    free(temp);
    prev->link=NULL;
    count1--;
}
void display()
{
    NODE *temp;
    temp=head;
    if(head==NULL)
    {
        printf("List is empty\n");
        return;
    }
    else
    {
        printf("List elements are : ");
    }
    while(temp!=NULL)
    {
        printf("%d\t",temp->data);
        temp = temp->link;
    }
}
void stack()
{
    int choice=0;
    while(choice<=4)
    {
        printf("\nSTACK IMPLEMENTATION\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\nENTER CHOICE : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: push();
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: return;
        }
    }

}
void enqueue()
{
    if(count2==s)
    {
        printf("\nQueue Overflow");
        return;
    }
    NODE *ptr=createnode();
    if(head1==NULL)
    {
        head1=ptr;
        count2++;
        return;
    }
    NODE *temp=head1;
    while(temp->link!=NULL)
    {
        temp=temp->link;
    }
    temp->link=ptr;
    count2++;
}
void dequeue()
{
    if(head1==NULL)
    {
        printf("\nQueue Underflow");
        return;
    }
    NODE *prev=head1;
    head1=head1->link;
    free(prev);
    prev=head1;
    count2--;
```

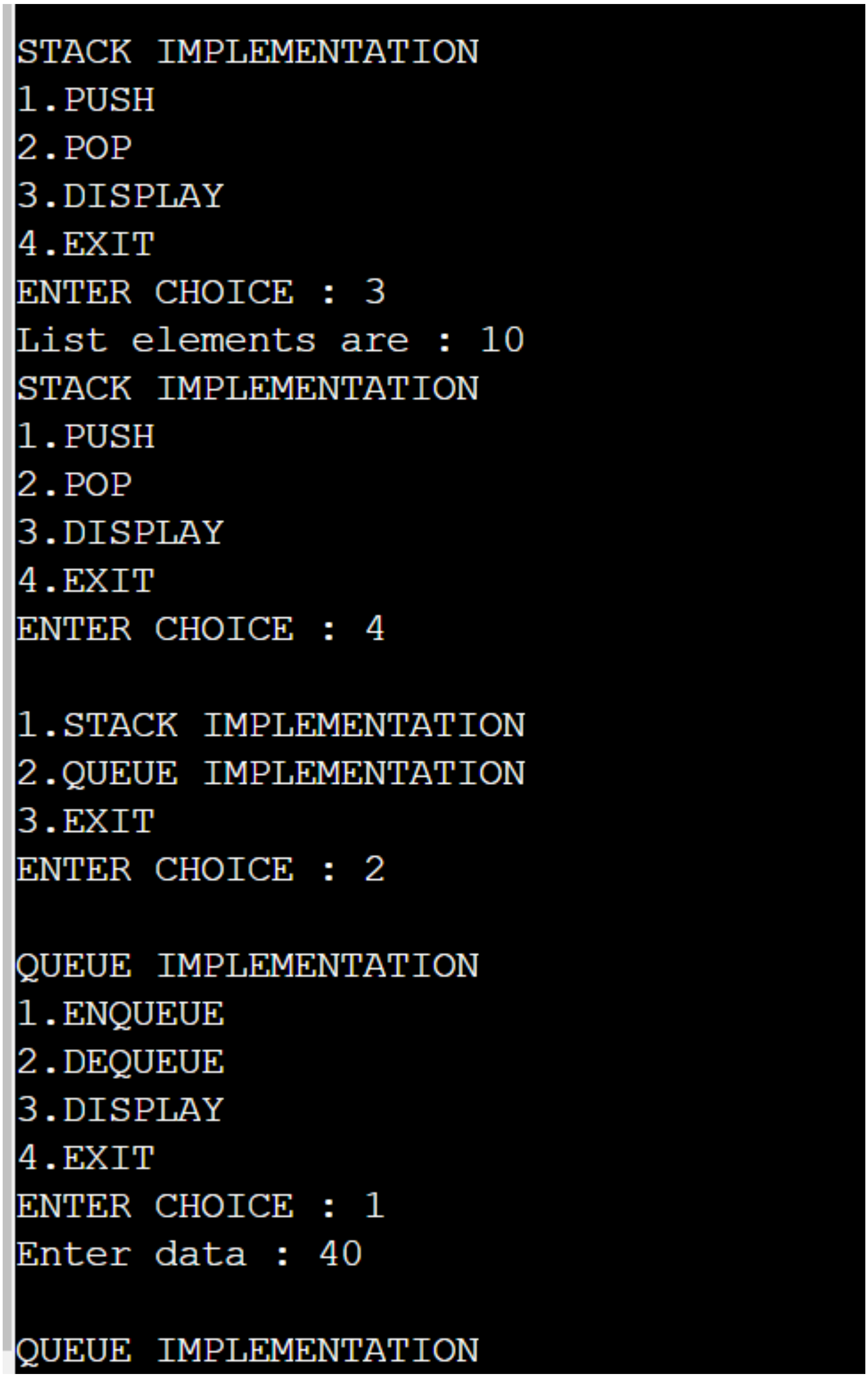
```
}
void display1()
{
   NODE *temp;
   temp=head1;
   if(head1==NULL)
   {
      printf("Queue is empty\n");
      return;
   }
   else
   {
      printf("Queue elements are : ");
   }
   while(temp!=NULL)
   {
      printf("%d\t",temp->data);
      temp = temp->link;
   }
}
void queue()
{
   int choice=0;
   while(choice<=4)
   {
      printf("\nQUEUE
IMPLEMENTATION\n1.ENQUEUE\n2.DEQUEUE\n3.DISPLAY\n4.EXIT\nENTER CHOICE : ");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1: enqueue();
               break;
```

```
            case 2: dequeue();
                    break;
            case 3: display1();
                    break;
            case 4: return;
        }
    }

}
void main()
{
    int c=0;
    while(c<=3)
    {
        printf("\n1.STACK IMPLEMENTATION\n2.QUEUE IMPLEMENTATION\n3.EXIT\n");
        printf("ENTER CHOICE : ");
        scanf("%d",&c);
        switch(c)
        {
            case 1: stack();
                    break;
            case 2: queue();
                    break;
            case 3: printf("\nExiting from program");
                    exit(0);
        }
    }
}
```

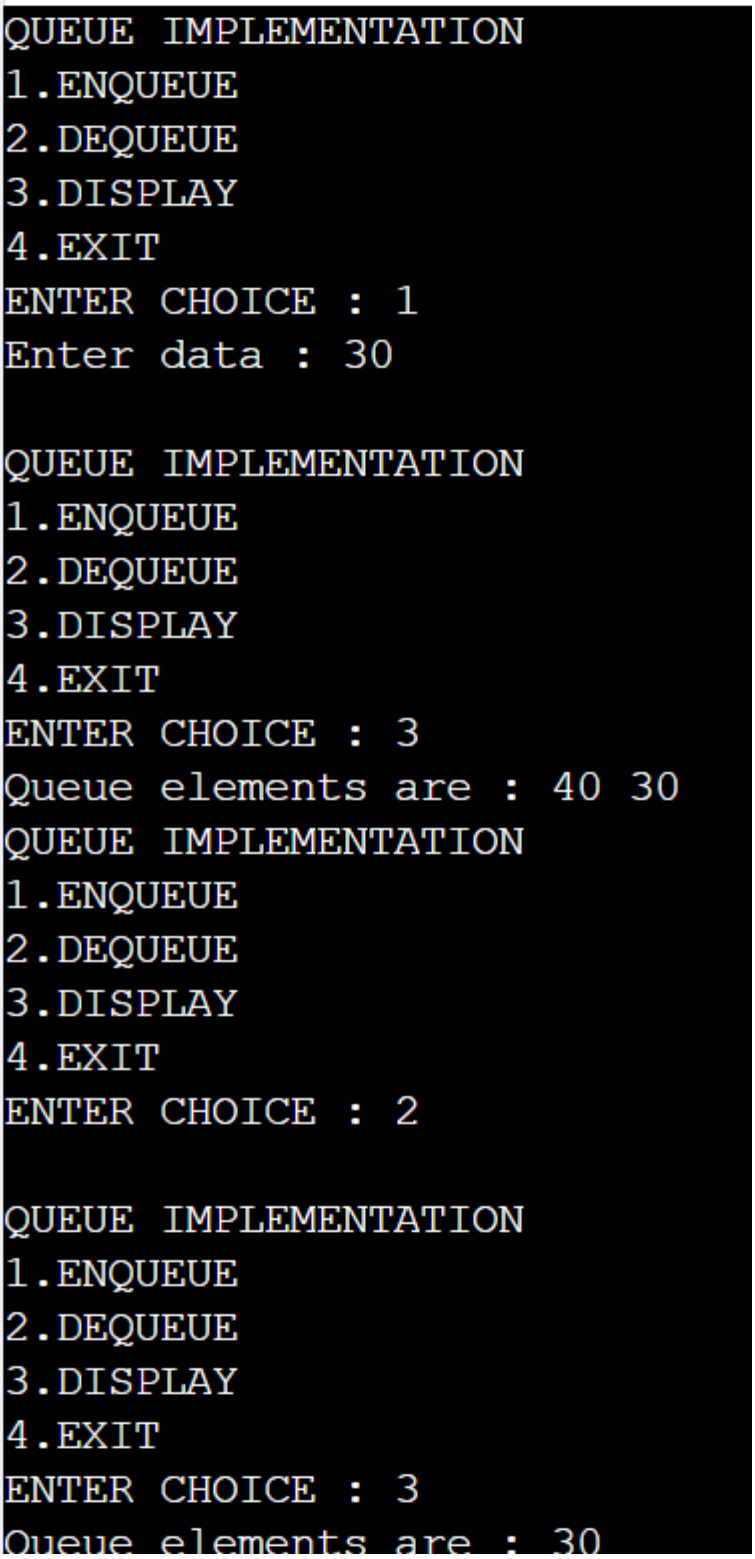
Output:



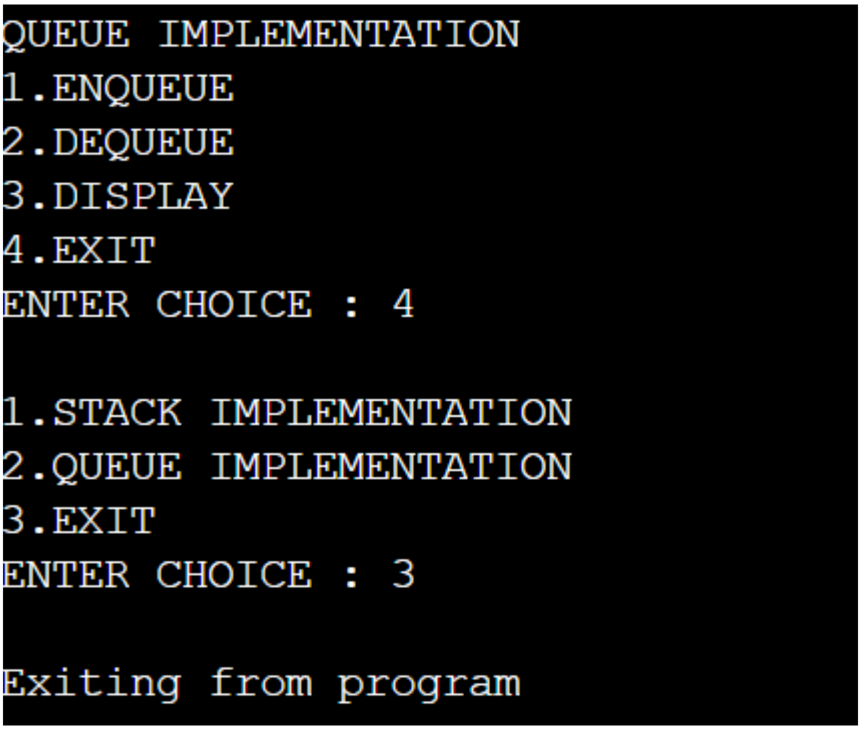
```
1.STACK IMPLEMENTATION
2.QUEUE IMPLEMENTATION
3.EXIT
ENTER CHOICE : 1

STACK IMPLEMENTATION
1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER CHOICE : 1
Enter data : 10

STACK IMPLEMENTATION
1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER CHOICE : 1
Enter data : 20

STACK IMPLEMENTATION
1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER CHOICE : 2
```



```
STACK IMPLEMENTATION
1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER CHOICE : 3
List elements are : 10
STACK IMPLEMENTATION
1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER CHOICE : 4

1.STACK IMPLEMENTATION
2.QUEUE IMPLEMENTATION
3.EXIT
ENTER CHOICE : 2

QUEUE IMPLEMENTATION
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT
ENTER CHOICE : 1
Enter data : 40

QUEUE IMPLEMENTATION
```



```
QUEUE IMPLEMENTATION
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT
ENTER CHOICE : 1
Enter data : 30

QUEUE IMPLEMENTATION
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT
ENTER CHOICE : 3
Queue elements are : 40 30
QUEUE IMPLEMENTATION
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT
ENTER CHOICE : 2

QUEUE IMPLEMENTATION
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT
ENTER CHOICE : 3
Queue elements are : 30
```



```
QUEUE IMPLEMENTATION
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT
ENTER CHOICE : 4

1.STACK IMPLEMENTATION
2.QUEUE IMPLEMENTATION
3.EXIT
ENTER CHOICE : 3

Exiting from program
```

8.Doubly Linked List Implementation

Code:

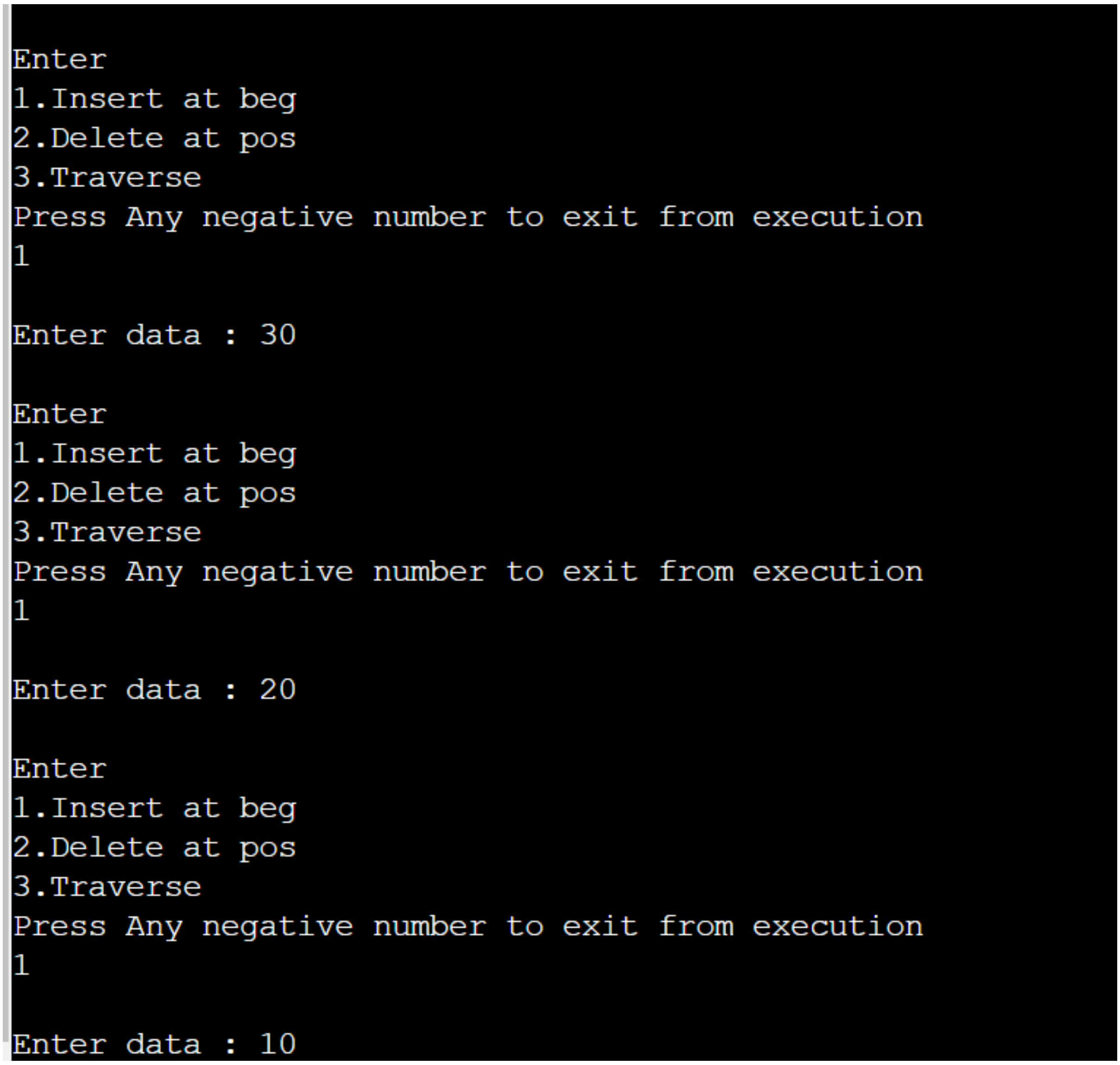
```
#include <stdio.h>
#include <stdlib.h>
typedef struct doublyList
{
   struct doublyList* llink;
   int data;
   struct doublyList* rlink;
}DNode;
DNode *first=NULL;
DNode* createNode()
{
   int item;
   DNode *ptr=(DNode*)malloc(sizeof(DNode));
   printf("\nEnter data : ");
   scanf("%d",&item);
   if( ptr!=NULL)
```

```
    {
        ptr->data=item;
        ptr->llink=NULL;
        ptr->rlink=NULL;
    }
    return ptr;
}

void insertBeg()
{
    DNode *temp=createNode();
    if(temp!=NULL && first==NULL)
        first=temp;
    else
        {
            temp->rlink=first;
            first->llink=temp;
            first=temp;
        }
}

void del_val()
{
    int item;
    printf("\nEnter the data to be deleted\n");
    scanf("%d",&item);
    DNode *temp=first;
    if(first==NULL)
    {
        printf("\nList is empty.Deletion not possible");
        return;
    }
```

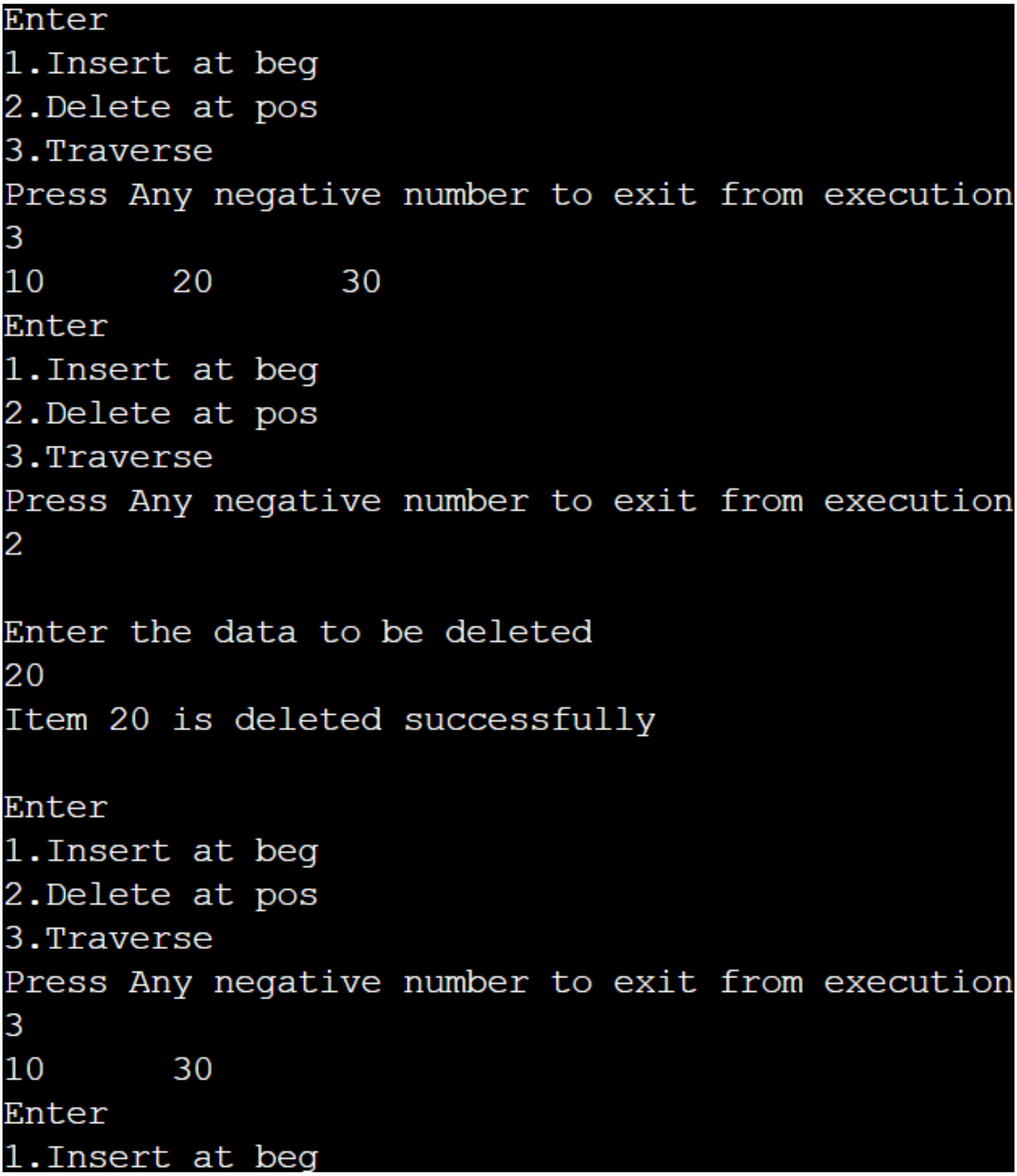
```
if(first->data==item && first->rlink==NULL)
{
    printf("Item %d is deleted successfully\n",first->data);
    first=NULL;
    free(first);
    return;
}
else if(first->data==item && first->rlink!=NULL)
{
     printf("Item %d is deleted successfully\n",first->data);
     first=first->rlink;
     free(first->llink);
     first->llink=NULL;
     return;
}
else
{
    while(temp!=NULL && temp->data!=item)
    {
          temp=temp->rlink;
    }
    if(temp==NULL)
    {
        printf("\nItem %d is not present in the list\n",item);
        return;
    }
    else if(temp->data==item && temp->rlink!=NULL)
    {
        printf("Item %d is deleted successfully\n",temp->data);
        temp->rlink->llink=temp->llink;
        temp->llink->rlink=temp->rlink;
        free(temp);
```

```
            return;
        }
        else if(temp->data==item && temp->rlink==NULL)
        {
             printf("Item %d is deleted successfully\n",temp->data);
             temp->llink->rlink=NULL;
             free(temp);
             return;

        }

    }
}

void traverse()
{
    DNode *temp=first;
    if(temp==NULL)
        printf("List is empty\n");
    else
    {
        while(temp!=NULL)
        {
            printf("%d\t",temp->data);
            temp=temp->rlink;
        }
    }
}
int main()
{

    int choice;
    while(1)

    {

        printf("\nEnter\n1.Insert at beg\n2.Delete at pos\n3.Traverse\nPress Any negative number to exit
from execution\n");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:insertBeg();

                   break;


            case 2: del_val();

                    break;

            case 3: traverse();

                    break;

            default:exit(0);

        }

    }

    return 0;

}
```

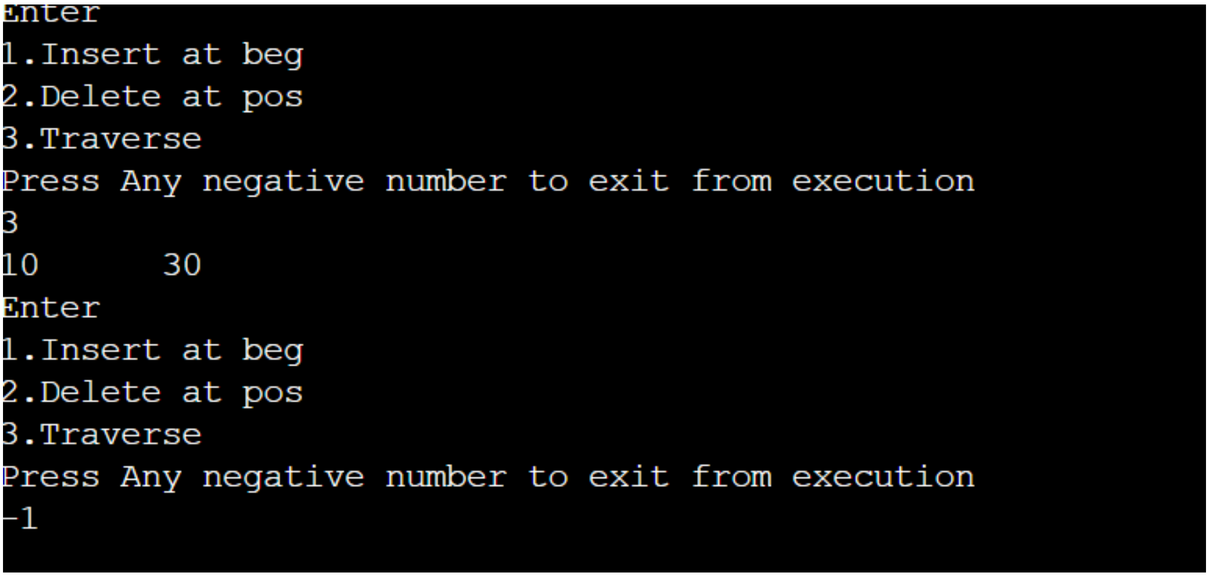
OUTPUT:



```
Enter
1.Insert at beg
2.Delete at pos
3.Traverse
Press Any negative number to exit from execution
1

Enter data : 30

Enter
1.Insert at beg
2.Delete at pos
3.Traverse
Press Any negative number to exit from execution
1

Enter data : 20

Enter
1.Insert at beg
2.Delete at pos
3.Traverse
Press Any negative number to exit from execution
1

Enter data : 10
```



```
Enter
1.Insert at beg
2.Delete at pos
3.Traverse
Press Any negative number to exit from execution
3
10      20      30
Enter
1.Insert at beg
2.Delete at pos
3.Traverse
Press Any negative number to exit from execution
2

Enter the data to be deleted
20
Item 20 is deleted successfully

Enter
1.Insert at beg
2.Delete at pos
3.Traverse
Press Any negative number to exit from execution
3
10      30
Enter
1.Insert at beg
```



```
Enter
1.Insert at beg
2.Delete at pos
3.Traverse
Press Any negative number to exit from execution
3
10      30
Enter
1.Insert at beg
2.Delete at pos
3.Traverse
Press Any negative number to exit from execution
-1
```

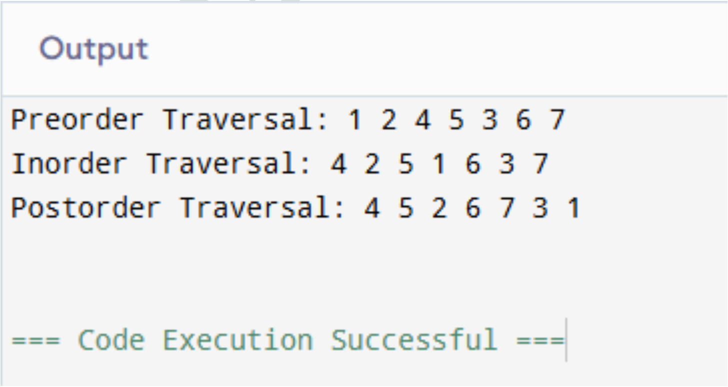
9.Binary Search

Code:

```
#include <stdio.h>

#include <stdlib.h>


// Definition of a node in the binary tree

struct Node {

   int value;

   struct Node* left;

   struct Node* right;

};


// Function to create a new node

struct Node* createNode(int value) {

   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

   newNode->value = value;

   newNode->left = newNode->right = NULL;

   return newNode;

}
```

```
// Preorder Traversal: Root -> Left -> Right
void preorder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->value);   // Visit the root
        preorder(root->left);         // Traverse the left subtree
        preorder(root->right);        // Traverse the right subtree
    }
}

// Inorder Traversal: Left -> Root -> Right
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);          // Traverse the left subtree
        printf("%d ", root->value);   // Visit the root
        inorder(root->right);         // Traverse the right subtree
    }
}

// Postorder Traversal: Left -> Right -> Root
void postorder(struct Node* root) {
    if (root != NULL) {
        postorder(root->left);        // Traverse the left subtree
        postorder(root->right);       // Traverse the right subtree
        printf("%d ", root->value);   // Visit the root
    }
}

// Main function
int main() {
    // Create the root node and other nodes
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->left = createNode(6);
    root->right->right = createNode(7);

    // Print traversals
    printf("Preorder Traversal: ");
    preorder(root);
    printf("\n");

    printf("Inorder Traversal: ");
    inorder(root);
    printf("\n");

    printf("Postorder Traversal: ");
    postorder(root);
    printf("\n");

    return 0;
}
```

OUTPUT:



```
Preorder Traversal: 1 2 4 5 3 6 7
Inorder Traversal: 4 2 5 1 6 3 7
Postorder Traversal: 4 5 2 6 7 3 1


=== Code Execution Successful ===
```

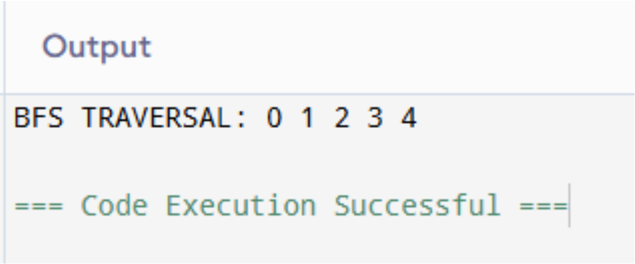
10.a)BFS

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


#define MAX_NODES 100


typedef struct Queue{

    int items[MAX_NODES];

    int front ,rear;

}Queue;


void initQueue(Queue *q){

    q->front =-1;

    q->rear=-1;

}


bool isEmpty(Queue *q){

    return q->front==-1;

}


void enqueue(Queue *q,int value){

    if(q->rear==MAX_NODES-1){

        printf("Queue overflown\n");

        return ;

    }

    if(q->front==-1){


        q->front=0;

    }

    q->items[++q->rear]=value;

}
```

```
int dequeue(Queue *q){
    if(isEmpty(q)){
        printf("Queue Underflow\n");
        return -1;
    }
    int value=q->items[q->front];
    if(q->front == q->rear){
        q->front=q->rear=-1;
    }
    else{
        q->front++;
    }
    return value;
}

typedef struct Graph {
    int adjMatrix[MAX_NODES][MAX_NODES];
    int numNodes;
}Graph;

void initGraph(Graph *g,int numNodes){
    g->numNodes=numNodes;
    for(int i=0;i<numNodes;i++){
        for(int j=0;j<numNodes;j++){
            g->adjMatrix[i][j]=0;
        }
    }
}

void addEdge(Graph *g,int src,int dest){
    g->adjMatrix[src][dest]=1;
    g->adjMatrix[dest][src]=1;
}

void bfs(Graph *g,int startNode){
    bool visited[MAX_NODES]={false};
    Queue q;
    initQueue(&q);
    visited[startNode]=true;
    enqueue(&q,startNode);
    printf("BFS TRAVERSAL: ");
    while(!isEmpty(&q)){
        int currentNode=dequeue(&q);
        printf("%d ",currentNode);
        for(int i=0;i<g->numNodes;i++){
            if(g->adjMatrix[currentNode][i]==1 && !visited[i]){
                visited[i]=true;
                enqueue(&q,i);
            }
        }
    }

}

int main(){
    Graph  g;
    initGraph(&g,5);
    addEdge(&g,0,1);
    addEdge(&g,0,2);
    addEdge(&g,1,3);
    addEdge(&g,1,4);
    addEdge(&g,2,4);
    bfs(&g,0);
}
```

OUTPUT:



```
BFS TRAVERSAL: 0 1 2 3 4

=== Code Execution Successful ===
```

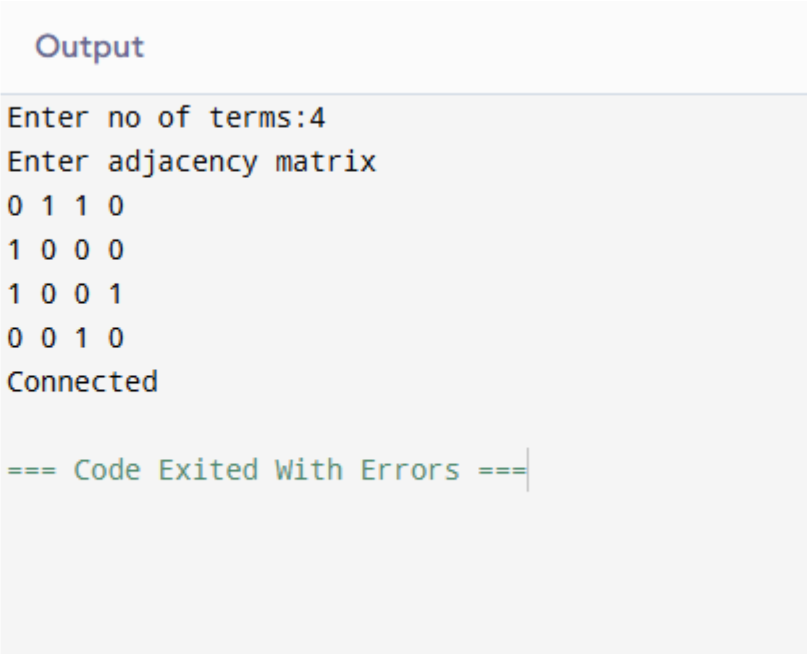
b)DFS

Code:

```
#include<stdio.h>
int vis[10],a[10][10],n;
void dfs(int i);

void main(){

   printf("Enter no of terms:");
   scanf("%d",&n);
   printf("Enter adjacency matrix\n");
   for(int i=1;i<=n;i++){
      for(int j=1;j<=n;j++){
         scanf("%d",&a[i][j]);
      }
   }
   for(int i=1;i<=n;i++){
      vis[i]=0;
   }
   dfs(1);
   for(int i=1;i<=n;i++){
      if(vis[i]==0){
         printf("Not connected");
         return;
      }
   }
```

```
    printf("Connected");
}
void dfs(int x){
    vis[x]=1;
    for(int i=1;i<=n;i++){
        if(a[i][x]==1 && vis[i]==0){
            dfs(i);
        }
    }
}
```

OUTPUT:



```
Enter no of terms:4
Enter adjacency matrix
0 1 1 0
1 0 0 0
1 0 0 1
0 0 1 0
Connected

=== Code Exited With Errors ===
```