VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Aparna Sankar (1BM23CS047)

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
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 Aparna Sankar **(1BM23CS047)**, 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.

Dr. Karanam Sunil Kumar	**Dr. Kavitha Sooda**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Course outcomes:

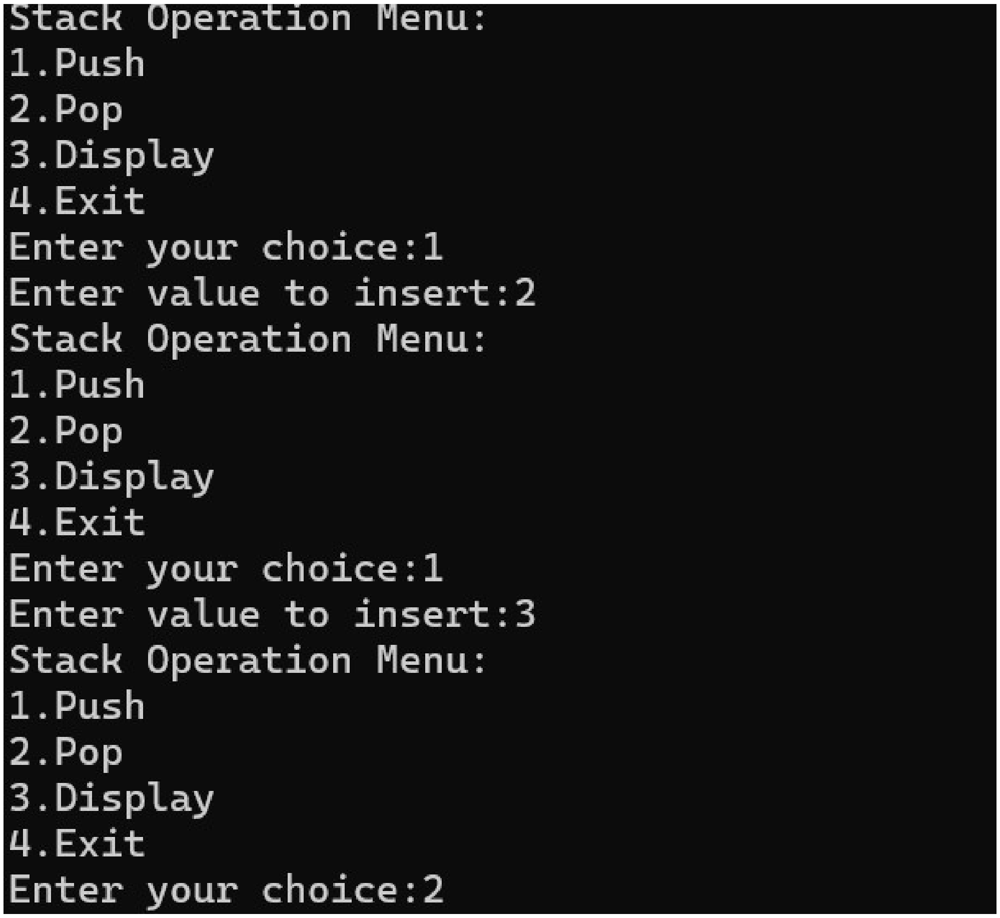
CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations 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>
#define MAX 3

int top=-1,stack[MAX],ele,i;

void push(int item)
{
  if(top==MAX-1)
  {
    printf("Stack is full,couldn't insert %d\n",item);
    return;
    }
    stack[++top]=item;
}

int pop()
{
    if(top==-1)
    {
        printf("Stack is empty\n");
        return -1;
    }
    ele=stack[top];
    top--;
    return ele;
}

void display()
{
    if(top==-1)
    {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack Contents:");
    for(i=top;i>=0;i--)
     printf("%d\t",stack[i]);
}

void main()
{
    int ch;
    do
    {
        printf("Stack Operation Menu:\n");
        printf("1.Push\n2.Pop\n3.Display4.Exit\nEnter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:printf("Enter value to insert:");
                    scanf("%d",&ele);
                    push(ele);
                    break;
            case 2:ele=pop();
                  if(ele!=-1)
                  {
                    printf("Deleted %d",ele);
                  }break;
            case 3:display();break;
            case 4:printf("Exiting...");break;
            default:printf("Invalid choice");
        }
    } while (ch!=4);

}
```

Output:

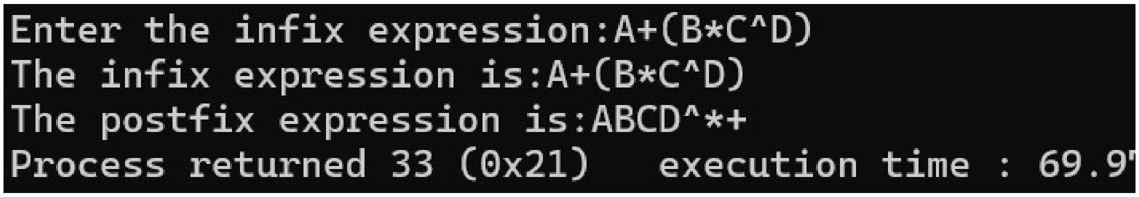


```
Stack Operation Menu:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:1
Enter value to insert:2
Stack Operation Menu:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:1
Enter value to insert:3
Stack Operation Menu:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:2
```

2.) 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) and / (divide)

```
 #include <stdio.h>

#include <conio.h>

#include <string.h>

int index=0, pos=0, top = -1, length=0;
```

```
char symb, temp, infix[20], postfix[20], stack[20];

void infixtopostfix();
void push(char symb);
char pop();
int pred(char symb);
void main()
{
    //clrscr();
    printf("Enter the infix expression:");
    scanf("%s", &infix);
    infixtopostfix();
    printf("The infix expression is:%s\n", infix);
    printf("The postfix expression is:%s", postfix);
    }
void infixtopostfix()
{
    length = strlen(infix);
    push('#');
    while (index<length)
    {
        symb = infix[index];
        switch (symb)
        {
        case '(':
            push('(');
            break;

        case ')':
            temp = pop();
            while (temp != '(')
            {
                postfix[pos] = temp;
                pos++;
                temp = pop();
            }
            break;
        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
            while (pred(stack[top]) >= pred(symb))
            {
                temp = pop();
                postfix[pos++] = temp;
            }
            push(symb);
            break;
        default:
            postfix[pos++] = symb;
        }
        index++;
    }
    while (top > 0)
    {
        temp = pop();
        postfix[pos++] = temp;
    }
}
void push(char symb)
{
    top += 1;
    stack[top] = symb;
}
char pop()
{
    char symb;
    symb = stack[top];
    top -= 1;
    return symb;
}
int pred(char symb)
{
    int p;
    switch (symb)
    {
    case '^':
        p = 3;
        break;
    case '*':
    case '/':
        p = 2;
        break;
    case '+':
    case '-':
        p = 1;
        break;
    case '(':
        p = 0;
        break;
    case '#':
        p = -1;
        break;
    }
    return p;
}
```

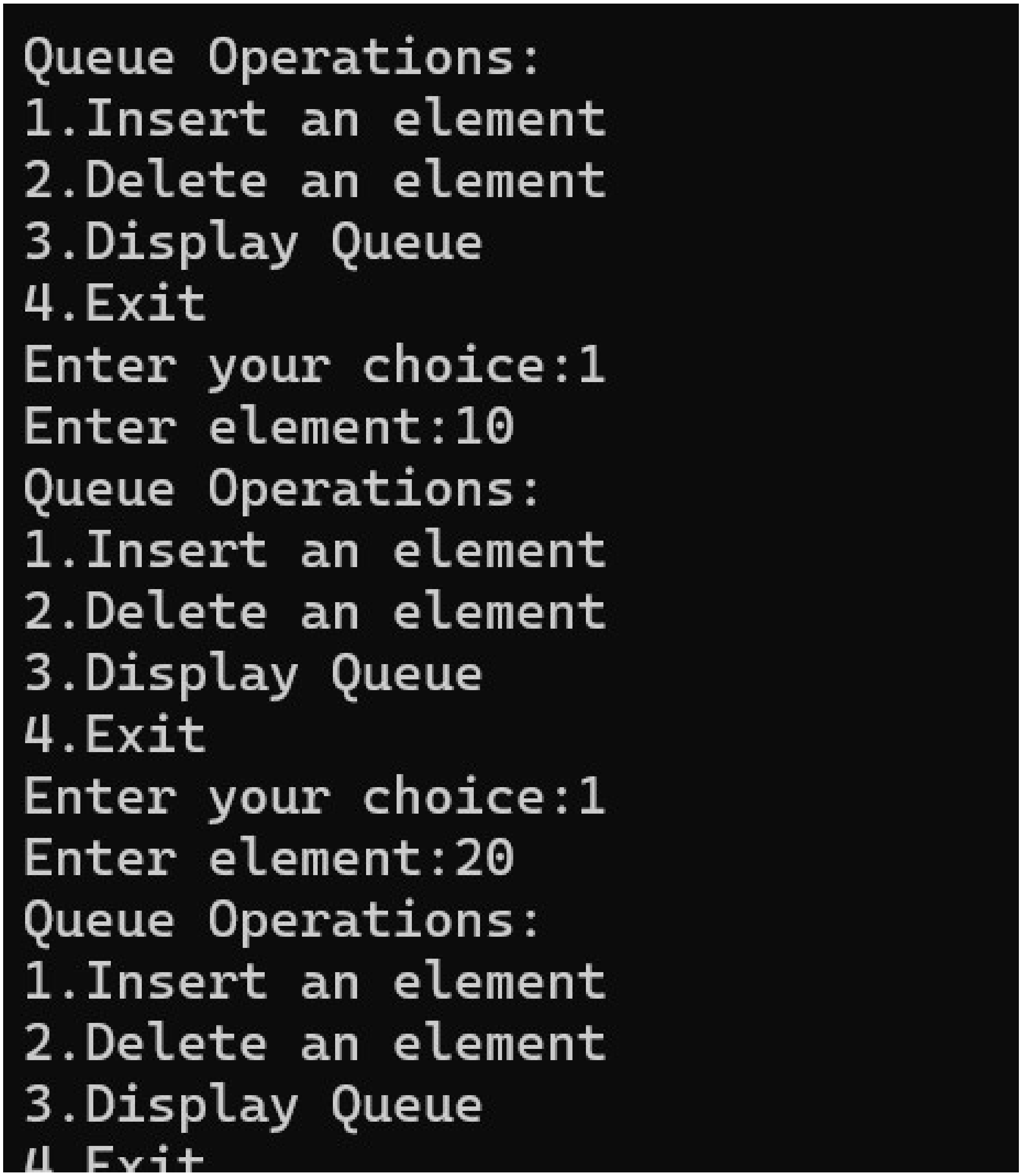


```
Enter the infix expression:A+(B*C^D)
The infix expression is:A+(B*C^D)
The postfix expression is:ABCD^*+
Process returned 33 (0x21)   execution time : 69.9
```

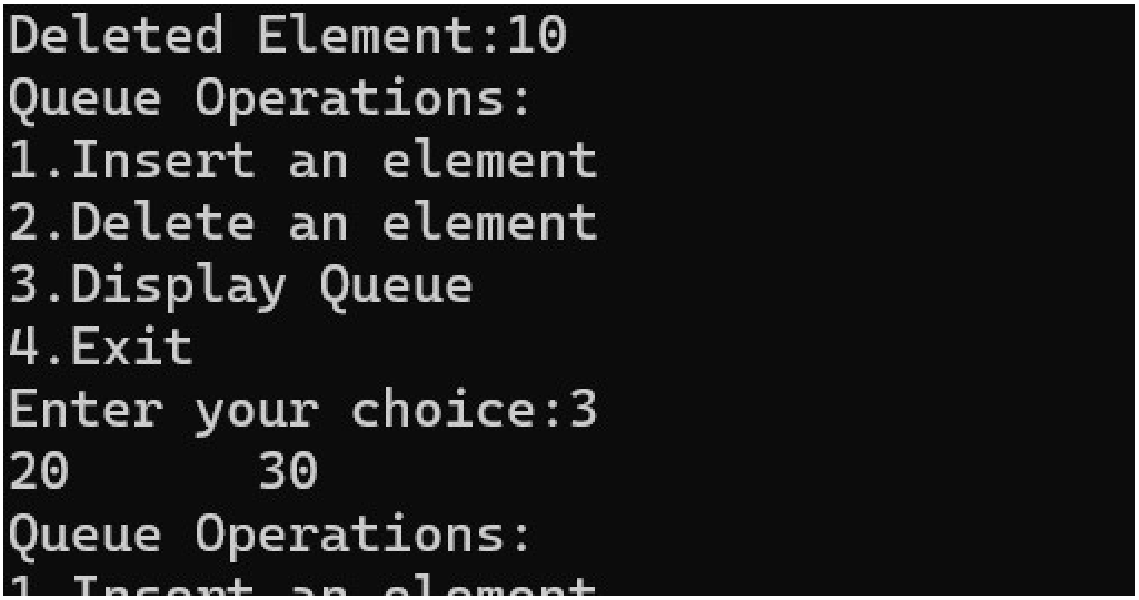
3.) Linear Queue Implementation

```
 #include <stdio.h>
#include <stdlib.h>
#define MAX 3

int front = -1, rear = -1, q[MAX], ele;

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

void main()
{
    int ch;
    do
    {
        printf("Queue Operations:\n1.Insert an element\n2.Delete an element\n3.Display
Queue\n4.Exit\nEnter your choice:");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            enqueue();
            break;
        case 2:
            ele = dequeue();
            if (ele != -1)
            {
                printf("Deleted Element:%d\n", ele);
            }
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting..");
            break;
        default:
            printf("Invalid choice");
        }

    } while (ch != 4);
}

void enqueue()
```

```
{
   if(rear==MAX-1)
   {
      printf("Queue is full\n");
      return;
   }
   printf("Enter element:");
   scanf("%d",&ele);
   if(front==-1 && rear==-1)
   {
      front=0;
   }
   q[++rear]=ele;
}

int dequeue()
{
   if(front==-1 && rear==-1)
   {
      printf("Queue is empty\n");
      return -1;
   }
   ele=q[front];
   if(front==rear)
   {
      front=-1;
      rear=-1;
   }
   else{
        front++;
    }
    return ele;
}

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



```
Queue Operations:
1.Insert an element
2.Delete an element
3.Display Queue
4.Exit
Enter your choice:1
Enter element:10
Queue Operations:
1.Insert an element
2.Delete an element
3.Display Queue
4.Exit
Enter your choice:1
Enter element:20
Queue Operations:
1.Insert an element
2.Delete an element
3.Display Queue
4.Exit
```



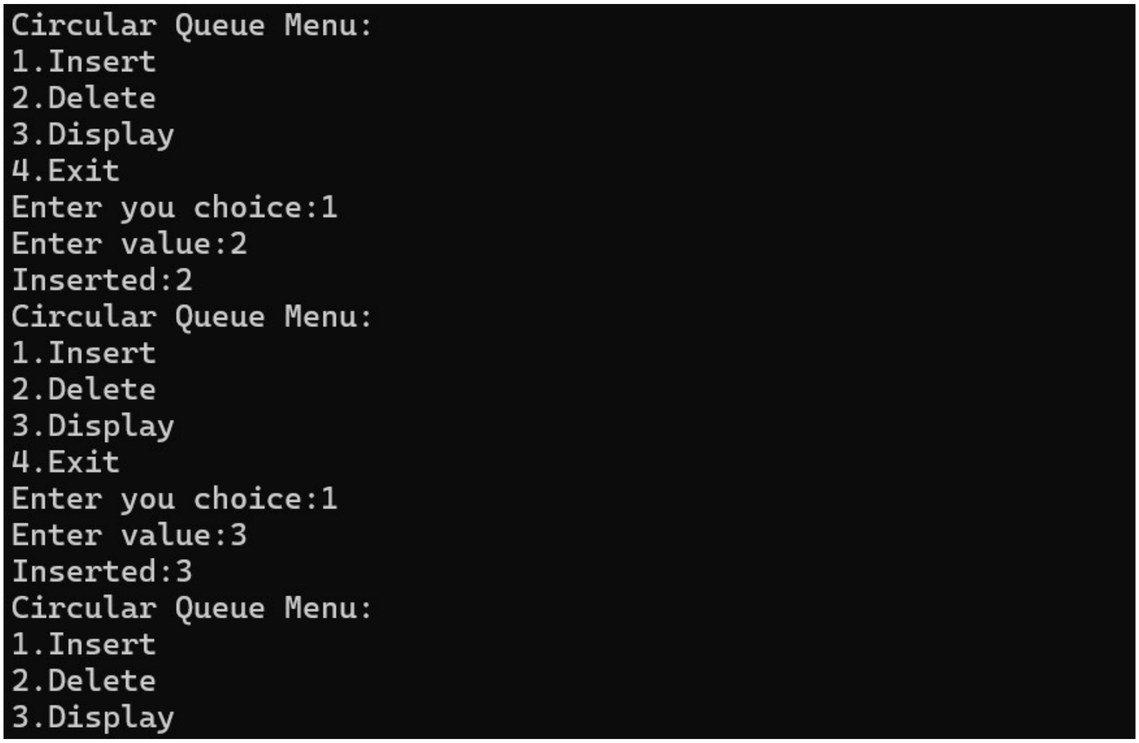
```
Deleted Element:10
Queue Operations:
1.Insert an element
2.Delete an element
3.Display Queue
4.Exit
Enter your choice:3
20      30
Queue Operations:
1.Insert an element
```

4) Circular Queue implementation

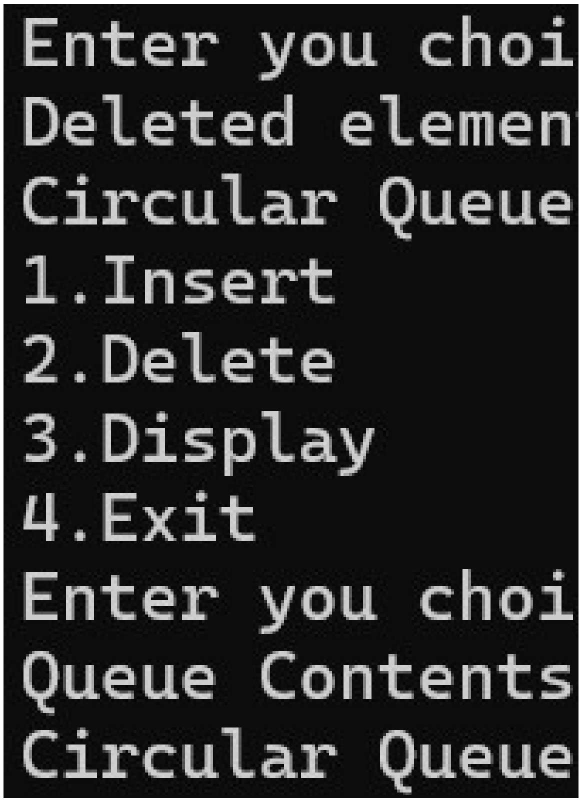
```
#include<stdio.h>
#define size 3


int cq[size],front=-1,rear=-1,ele;


void insert(int item)
{
   if(front==(rear+1)% size)
   {
      printf("Queue is full");
      return;
   }
   if(front==-1 && rear==-1){
      front=0;rear=0;
   }
```

```
    else
     rear=(rear+1)%size;

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

int delete()
{
    if(front==-1 && rear==-1){
        printf("Queue is empty");
        return -1;
    }
    ele=cq[front];
    if(front==rear){
        front=-1;rear=-1;
    }
    else
     front=(front+1)%size;
    return ele;
}

void display()
{
    if(front==-1 && rear==-1){
        printf("Queue is empty");
        return;
    }
    printf("Queue Contents:");
    if(front<=rear)
    {
        for(int i=front;i<=rear;i++)
            printf("%d\t",cq[i]);
        printf("\n");
    }
    else{
        for(int i=front;i<size;i++)
            printf("%d\t",cq[i]);
        for(int i=0;i<=rear;i++)
            printf("%d\t",cq[i]);
        printf("\n");
    }
}

void main()
{
    int ch;
    do
    {
        printf("Circular Queue Menu:\n");
        printf("1.Insert\n2.Delete\n3.Display\n4.Exit\nEnter you choice:");
        scanf("%d",&ch);
        switch(ch){
            case 1:printf("Enter value:");
```

```
                scanf("%d",&ele);
                insert(ele);
                break;
            case 2:ele=delete();
                if(ele!=-1)
                    printf("Deleted element:%d\n",ele);
                break;
            case 3:display();break;
            case 4:printf("Exiting....");break;
            default:printf("Invalid Choice\n");
        }
    }while(ch!=4);
}
```



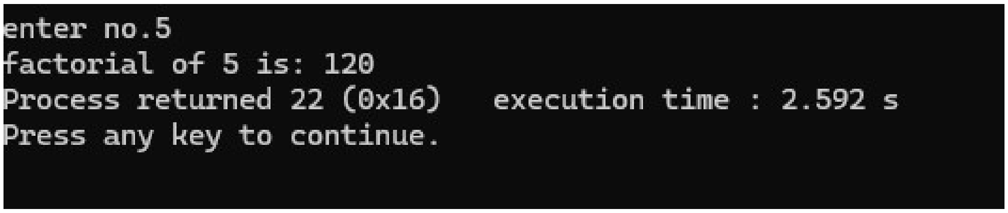
```
Circular Queue Menu:
1.Insert
2.Delete
3.Display
4.Exit
Enter you choice:1
Enter value:2
Inserted:2
Circular Queue Menu:
1.Insert
2.Delete
3.Display
4.Exit
Enter you choice:1
Enter value:3
Inserted:3
Circular Queue Menu:
1.Insert
2.Delete
3.Display
```



```
Enter you choi
Deleted elemen
Circular Queue
1.Insert
2.Delete
3.Display
4.Exit
Enter you choi
Queue Contents
Circular Queue
```

Q5) Recursion program for factorial of a number

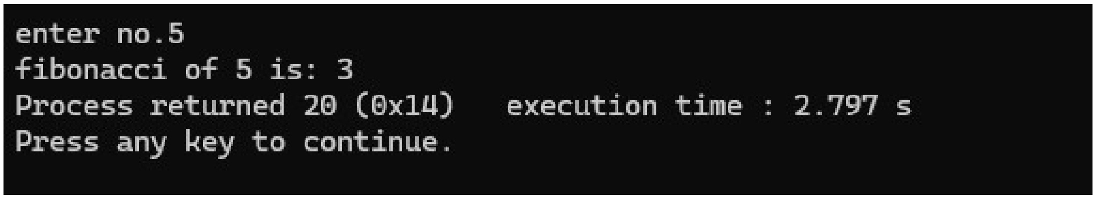
```
#include<stdio.h>
int fact(int n)
{
  if(n==1)
   return 1;
  else
   return(n*fact(n-1));
}

void main() {
  int n;
  printf("enter no.");
  scanf("%d",&n);
  printf("factorial of %d is: %d",n,fact(n));
}
```



```
enter no.5
factorial of 5 is: 120
Process returned 22 (0x16)   execution time : 2.592 s
Press any key to continue.
```

3B) Recursion program for fibonacci of a number

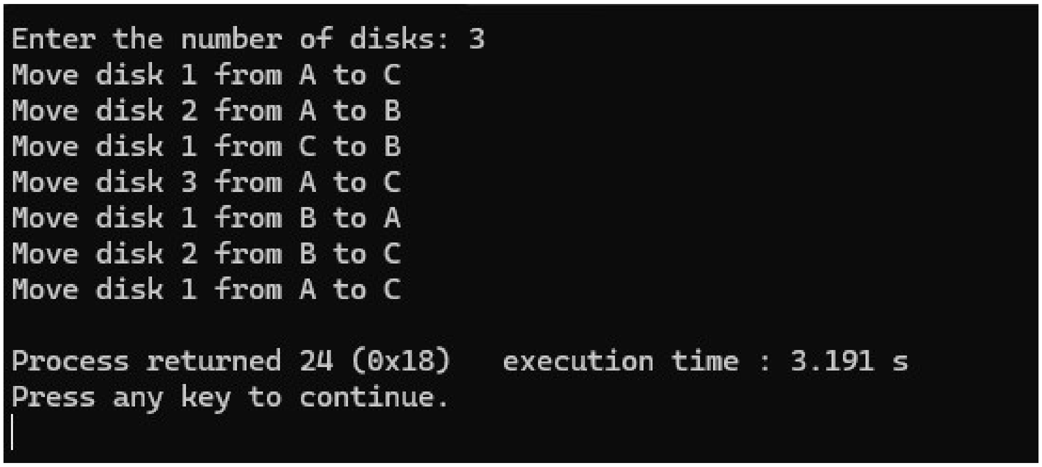
```
#include<stdio.h>
int fib(int n)
{
   if(n==1)
    return 0;
   else if(n==2)
    return 1;
   else
    return(fib(n-1)+fib(n-2));
}

void main() {
   int n;
   printf("enter no.");
   scanf("%d",&n);
   printf("fibonacci of %d is: %d",n,fib(n));
}
```



```
enter no.5
fibonacci of 5 is: 3
Process returned 20 (0x14)   execution time : 2.797 s
Press any key to continue.
```

3C) Tower of Hanoi using recursion

```
#include<stdio.h>

void towerOfHanoi(int n, char source, char temp, char destination) {
  if (n == 1) {
     printf("Move disk 1 from %c to %c\n", source, destination);
     return;
  }
  towerOfHanoi(n - 1, source, destination, temp);
  printf("Move disk %d from %c to %c\n", n, source, destination);
  towerOfHanoi(n - 1, temp,source,destination);
}

void main() {
  int n;
  printf("Enter the number of disks: ");
  scanf("%d", &n);
  towerOfHanoi(n, 'A', 'B', 'C');
}
```



```
Enter the number of disks: 3
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C

Process returned 24 (0x18)   execution time : 3.191 s
Press any key to continue.
```

6) WAP to Implement Singly Linked List
with following operations

a) Create a linked list.

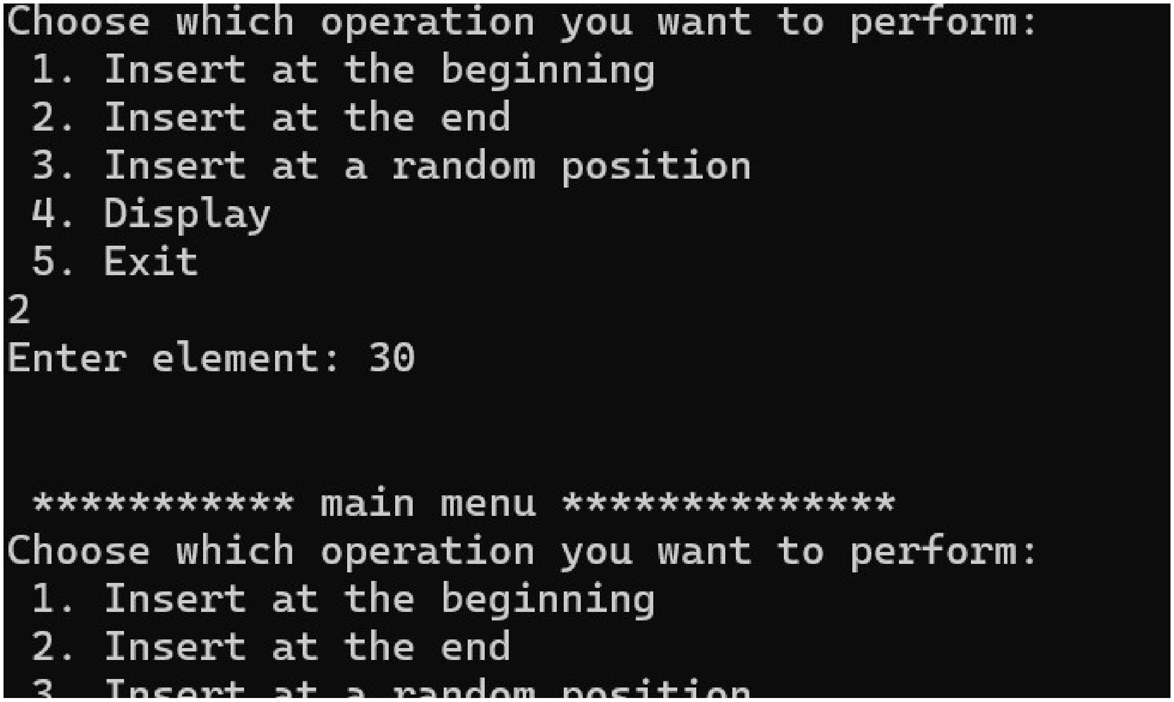
b) Insertion of a node at first position, at any position and at end of list.

c) Display the contents of the linked list.

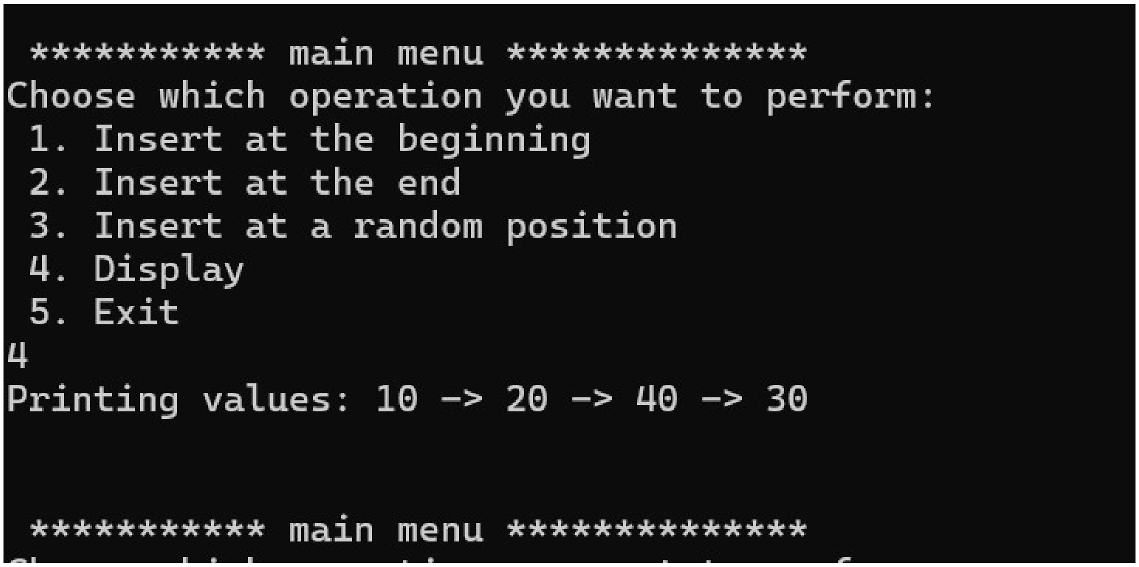
```
#include<stdio.h>
#include<stdlib.h>
```

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

struct node *head = NULL;

void begininsert();
void endinsert();
void randominsert();
void display();

void main () {
    int choice = 0;  // Initialize choice
    while(choice != 9) {
        printf("\n \n *********** main menu ************** \n");
        printf("Choose which operation you want to perform: \n");
        printf(" 1. Insert at the beginning \n 2. Insert at the end \n 3. Insert at a random position \n 4. Display \n 5. Exit \n");
        scanf("%d", &choice);

        switch(choice) {
            case 1: begininsert(); break;
            case 2: endinsert(); break;
            case 3: randominsert(); break;
            case 4: display(); break;
            case 5: exit(0); break;  // Correct exit
            default: printf("Invalid choice, please try again.\n");
        }
    }
```

```
}

void begininsert() {
    struct node* ptr;
    int item;
    ptr = (struct node*)malloc(sizeof(struct node));

    if(ptr == NULL) {
        printf("Overflow \n");
    } else {
        printf("Enter element: ");
        scanf("%d", &item);
        ptr->data = item;
        ptr->next = head;
        head = ptr;
        printf("Node inserted \n");
    }
}

void endinsert() {
    struct node* ptr, *temp;
    int item;
    ptr = (struct node*)malloc(sizeof(struct node));

    if(ptr == NULL) {
        printf("Overflow \n");
    } else {
        printf("Enter element: ");
        scanf("%d", &item);
        ptr->data = item;
    }

    if(head == NULL) {
        ptr->next = NULL;
        head = ptr;
        printf("Node inserted \n");
    } else {
        temp = head;
        while(temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = ptr;
        ptr->next = NULL;
    }
}

void randominsert() {
    int i, item, pos;
    struct node* ptr, *temp;
    ptr = (struct node*)malloc(sizeof(struct node));

    if(ptr == NULL) {
        printf("Overflow \n");
    } else {
        printf("Enter element: ");
        scanf("%d", &item);
        ptr->data = item;
        printf("Enter position where you want to insert: ");
        scanf("%d", &pos);
        temp = head;
        if (pos == 1) {
            ptr->next = head;
            head = ptr;
            printf("Node inserted\n");
            return;
        }

        for(i = 0; i < pos - 1; i++) {
            if(temp == NULL) {
                printf("Position is out of bounds.\n");
                return;
            }
            temp = temp->next;
        }

        ptr->next = temp->next;
        temp->next = ptr;
        printf("Node inserted\n");
    }
}

void display() {
    struct node* ptr;
    ptr = head;
    if(ptr == NULL) {
        printf("Nothing to print\n");
    } else {
        printf("Printing values: ");
        while(ptr != NULL) {
            printf("%d", ptr->data);
            if(ptr->next != NULL) {
                printf(" -> ");
            }
            ptr = ptr->next;
        }
        printf("\n");
    }
}
```



```
Choose which operation you want to perform:
 1. Insert at the beginning
 2. Insert at the end
 3. Insert at a random position
 4. Display
 5. Exit
2
Enter element: 30


 *********** main menu **************
Choose which operation you want to perform:
 1. Insert at the beginning
 2. Insert at the end
 3. Insert at a random position
```



```
 *********** main menu **************
Choose which operation you want to perform:
 1. Insert at the beginning
 2. Insert at the end
 3. Insert at a random position
 4. Display
 5. Exit
4
Printing values: 10 -> 20 -> 40 -> 30


 *********** main menu **************
```

a) Create a linked list.

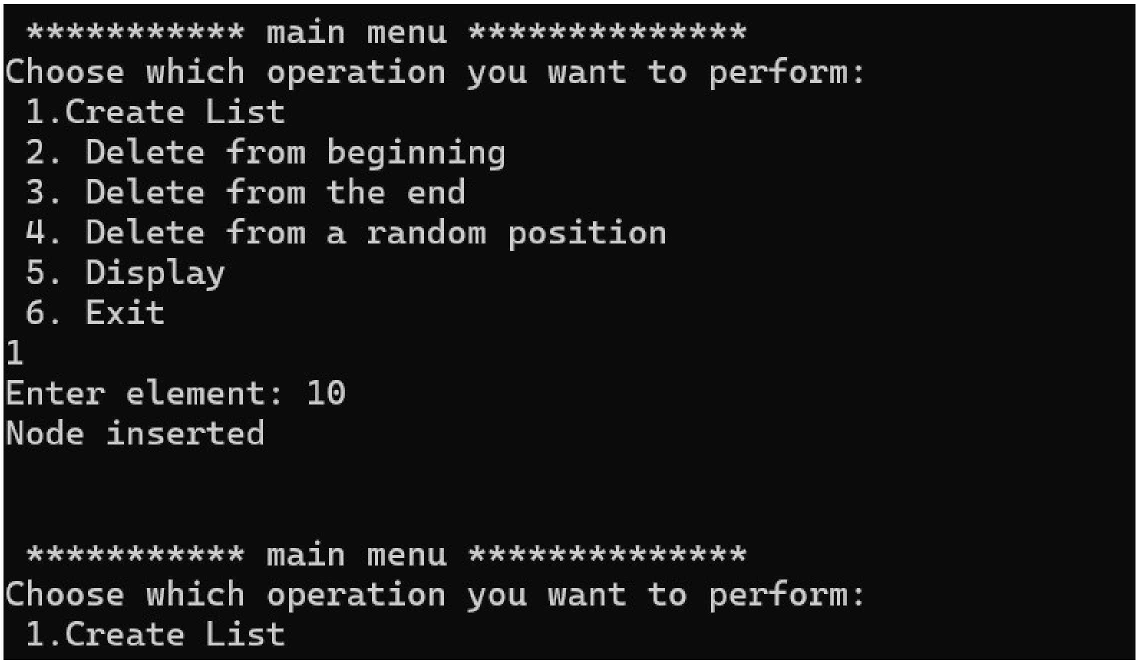
b) Deletion
of first element, specified element and last element in the list.
c) Display
the contents of the linked list.

```
#include<stdio.h>

#include<stdlib.h>


struct node {

   int data;

   struct node *next;

};


struct node *head = NULL;



void create_list();

void begin_delete();

void end_delete();
void random_delete();
void display();

void main () {
   int choice = 0;  // Initialize choice
   while(choice != 9) {
      printf("\n \n *********** main menu ************** \n");
      printf("Choose which operation you want to perform: \n");
      printf(" 1.Create List \n 2. Delete from beginning\n 3. Delete from the end \n 4. Delete from a random position \n 5. Display \n 6. Exit \n");
      scanf("%d", &choice);

      switch(choice) {
         case 1: create_list(); break;
         case 2: begin_delete(); break;
         case 3: end_delete(); break;
         case 4: random_delete(); break;
         case 5: display(); break;
         case 6: exit(0); break;  // Correct exit
         default: printf("Invalid choice, please try again.\n");
      }
   }
}

void create_list() {
   struct node* ptr, *temp;
   int item;
   ptr = (struct node*)malloc(sizeof(struct node));
```

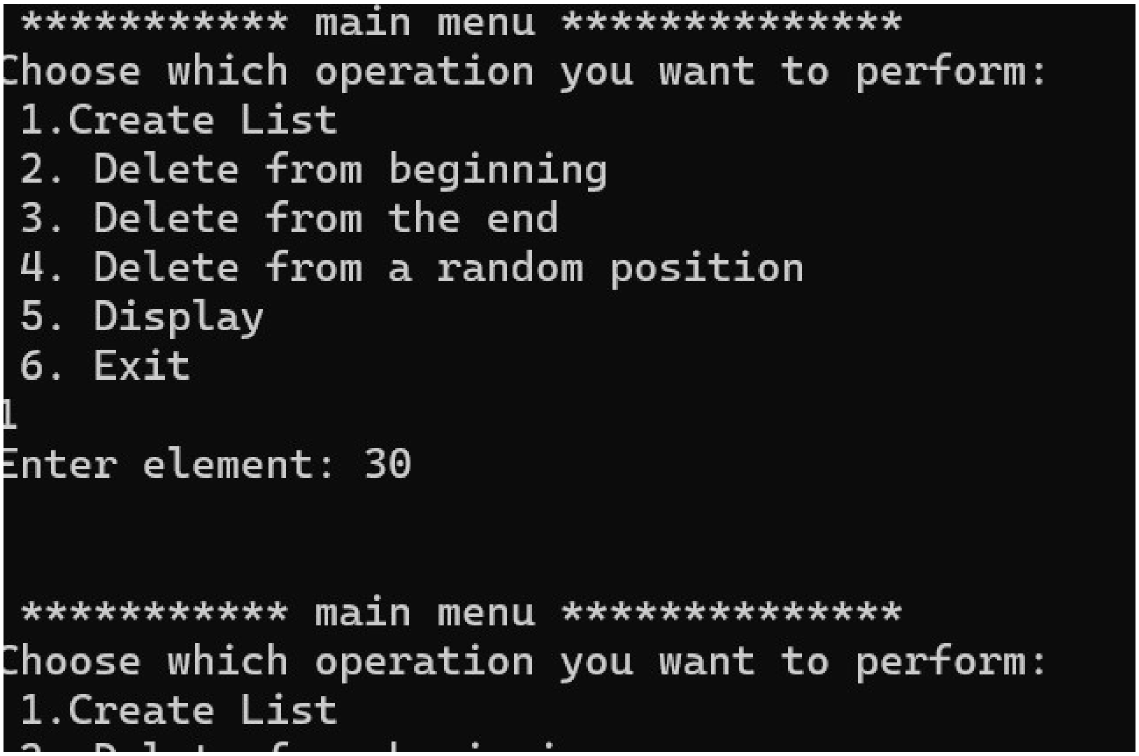
```
    if(ptr == NULL) {
        printf("Overflow \n");
    } else {
        printf("Enter element: ");
        scanf("%d", &item);
        ptr->data = item;
    }

    if(head == NULL) {
        ptr->next = NULL;
        head = ptr;
        printf("Node inserted \n");
    } else {
        temp = head;
        while(temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = ptr;
        ptr->next = NULL;
    }
}


void begin_delete() {
    struct node* ptr;
    if(head == NULL) {
        printf("List is empty \n");
    } else {
        ptr = head;
        head = ptr->next;
        free(ptr);
        printf("Node deleted from the beginning \n");
    }
}

void end_delete() {
    struct node* ptr, *ptr1;
    if(head == NULL) {
        printf("List is empty \n");
    } else if(head->next == NULL) {
        head = NULL;
        free(head);
        printf("The only node is deleted\n");
    } else {
        ptr = head;
        while(ptr->next != NULL) {
            ptr1 = ptr;
            ptr = ptr->next;
        }
        ptr1->next = NULL;
        free(ptr);
        printf("Node deleted from the end\n");
    }
```

```
}

void random_delete() {
    struct node* ptr, *ptr1;
    int i, loc;
    printf("Enter position you want to delete from: ");
    scanf("%d", &loc);
    ptr = head;
    if (loc == 1) {
        head = ptr->next;
        free(ptr);
        printf("Node deleted from position 1\n");
        return;
    }

    for(i = 0; i < loc - 1; i++) {
        ptr1 = ptr;
        ptr = ptr->next;
        if(ptr == NULL) {
            printf("Can't delete \n");
            return;
        }
    }
    ptr1->next = ptr->next;
    free(ptr);
    printf("Node deleted from position: %d \n", loc);
}
```

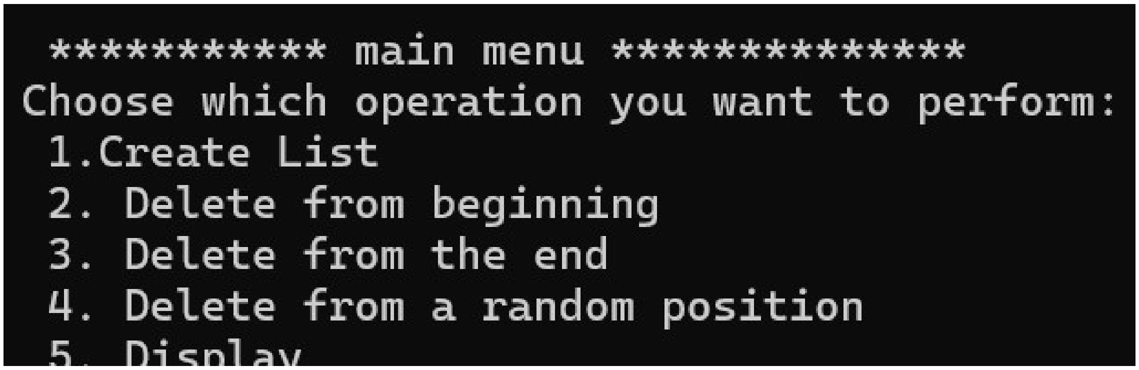
```
void display() {
    struct node* ptr;
    ptr = head;
    if(ptr == NULL) {
        printf("Nothing to print\n");
    } else {
        printf("Printing values: ");
        while(ptr != NULL) {
            printf("%d", ptr->data);
            if(ptr->next != NULL) {
                printf(" -> ");
            }
            ptr = ptr->next;
        }
        printf("\n");
    }
}
```



```
 *********** main menu **************
Choose which operation you want to perform:
 1.Create List
 2. Delete from beginning
 3. Delete from the end
 4. Delete from a random position
 5. Display
 6. Exit
1
Enter element: 10
Node inserted


 *********** main menu **************
Choose which operation you want to perform:
 1.Create List
```



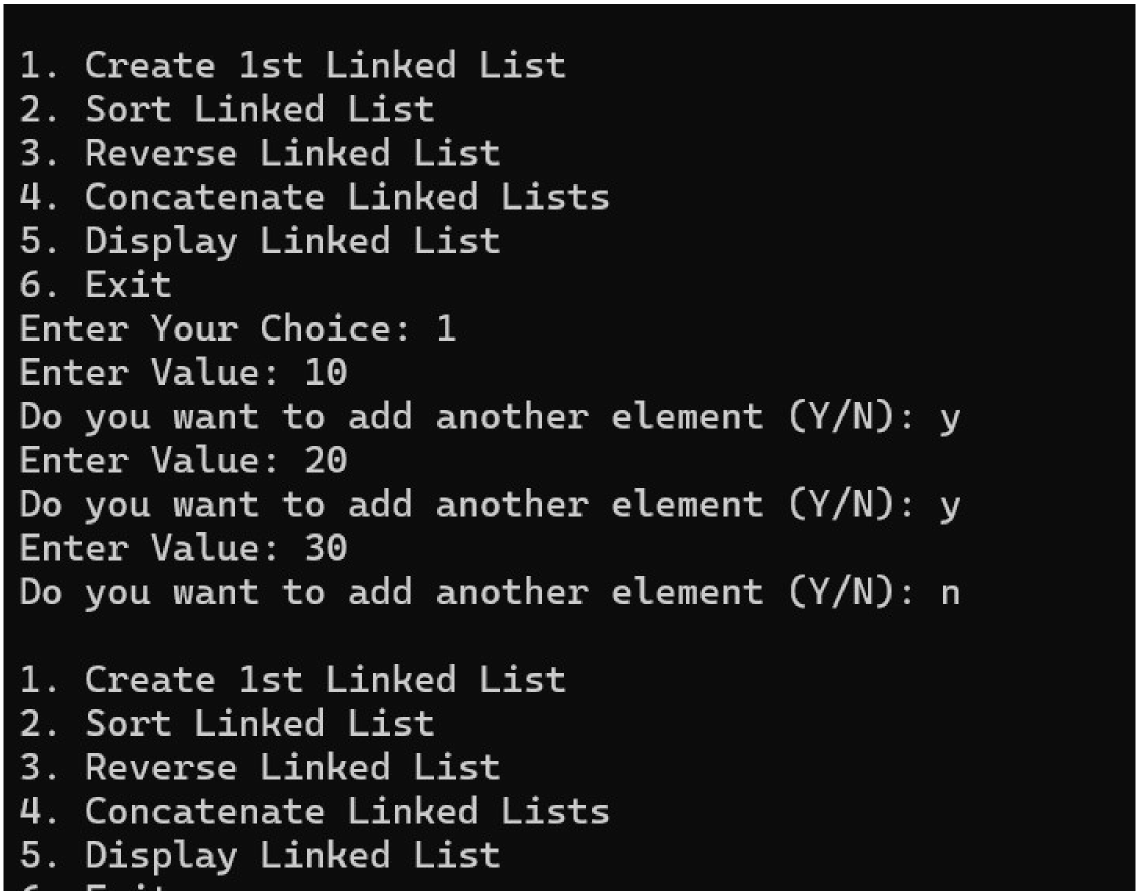
```
 *********** main menu **************
Choose which operation you want to perform:
 1.Create List
 2. Delete from beginning
 3. Delete from the end
 4. Delete from a random position
 5. Display
 6. Exit
1
Enter element: 30


 *********** main menu **************
Choose which operation you want to perform:
 1.Create List
 2. Delete from beginning
```



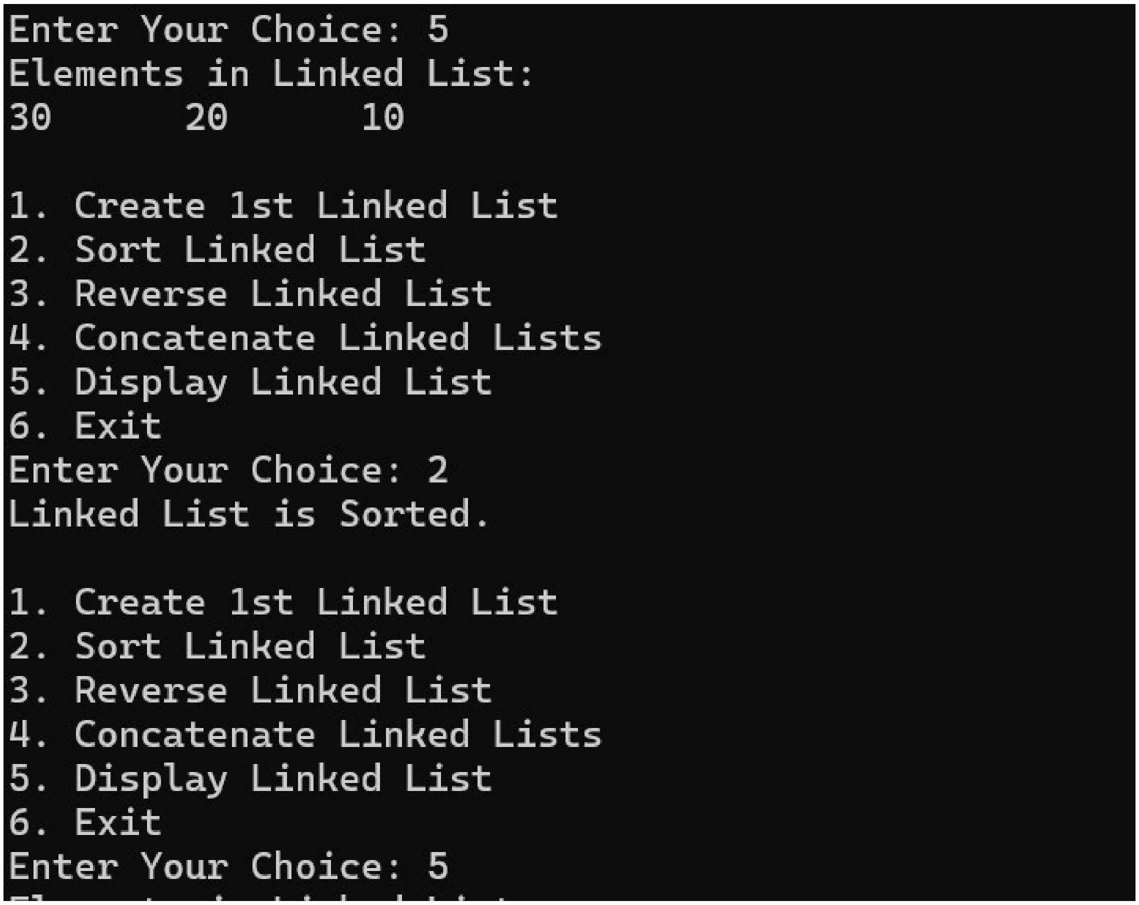
```
 *********** main menu ***************
Choose which operation you want to perform:
 1.Create List
 2. Delete from beginning
 3. Delete from the end
 4. Delete from a random position
 5. Display
```

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

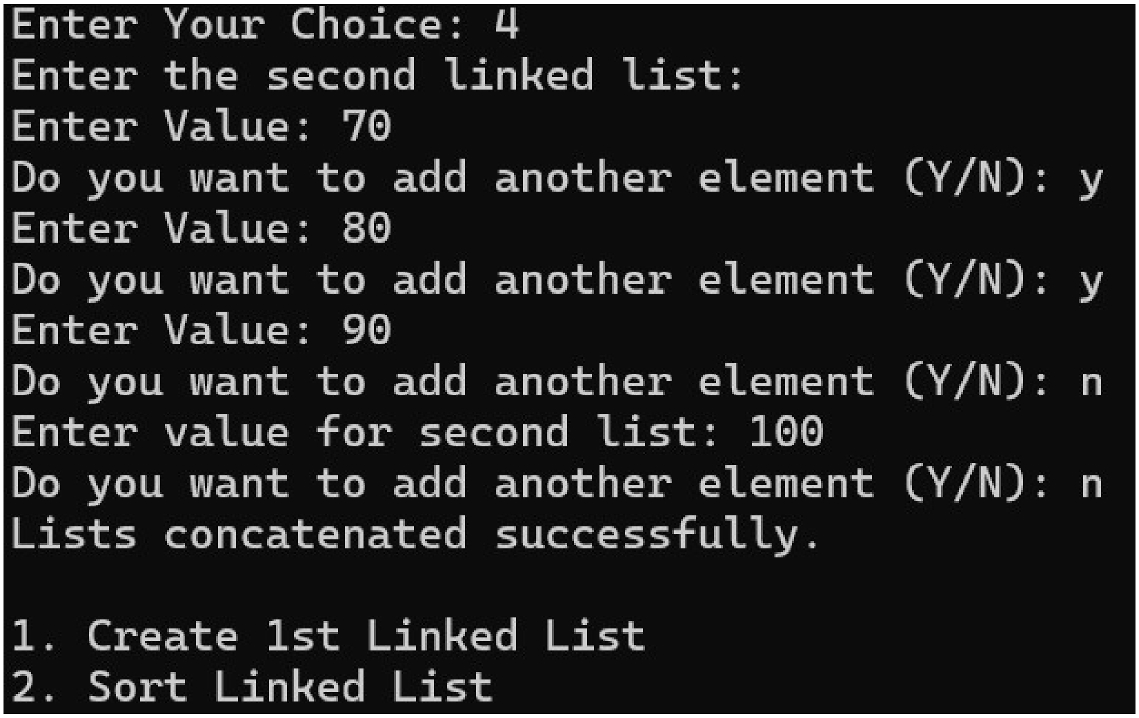
```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node *link;

};

typedef struct Node node;


node *start = NULL, *temp, *new1, *curr;

int ch;

char c;


void createList();

void sort();

void reverse();

void display();

void concatenate();


void createList() {
    do {
        new1 = (node*)malloc(sizeof(node));
        printf("Enter Value: ");
        scanf("%d", &new1->data);
        new1->link = NULL;

        if (start == NULL) {
            start = new1;
            curr = new1;
        } else {
            curr->link = new1;
            curr = new1;
        }
        printf("Do you want to add another element (Y/N): ");
        scanf(" %c", &c);
    } while (c == 'y' || c == 'Y');
}

void sort() {
    if (start == NULL) {
        printf("The Linked List is Empty.\n");
        return;
    }

    node *i, *j;
    int tempData;
    for (i = start; i != NULL; i = i->link) {
        for (j = i->link; j != NULL; j = j->link) {
            if (i->data > j->data) {
                tempData = i->data;
                i->data = j->data;
                j->data = tempData;
            }
        }
    }
    printf("Linked List is Sorted.\n");
}

void reverse() {
    node *a = start, *b = NULL;
    while (a != NULL) {
        temp = a->link;
        a->link = b;
        b = a;
        a = temp;
    }
    start = b;
    printf("Linked List is Reversed.\n");
}

void display() {
    if (start == NULL) {
        printf("Linked list is Empty\n");
        return;
    }

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

void concatenate() {
    node *start2 = NULL, *curr2 = NULL;

    printf("Enter the second linked list:\n");
    createList();

    do {
        new1 = (node*)malloc(sizeof(node));
        printf("Enter value for second list: ");
        scanf("%d", &new1->data);
        new1->link = NULL;

        if (start2 == NULL) {
            start2 = new1;
            curr2 = new1;
        } else {
            curr2->link = new1;
            curr2 = new1;
        }
        printf("Do you want to add another element (Y/N): ");
        scanf(" %c", &c);
    } while (c == 'y' || c == 'Y');
    if (start == NULL) {
        start = start2;
    } else {
        temp = start;
        while (temp->link != NULL) {
            temp = temp->link;
        }
        temp->link = start2;
    }
    start2 = NULL;
    printf("Lists concatenated successfully.\n");
}

int main() {
    while (1) {
        printf("\n1. Create 1st Linked List\n2. Sort Linked List\n3. Reverse Linked List\n4. Concatenate Linked Lists\n5. Display Linked List\n6. Exit\n");
        printf("Enter Your Choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                createList();
                break;
            case 2:
                sort();
                break;
            case 3:
                reverse();
                break;
            case 4:
                concatenate();
                break;
            case 5:
                display();
                break;
            case 6:
                exit(0);
                break;
            default:
                printf("Invalid choice. Please try again.\n");
                break;
        }
    }
}
```



```
1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 1
Enter Value: 10
Do you want to add another element (Y/N): y
Enter Value: 20
Do you want to add another element (Y/N): y
Enter Value: 30
Do you want to add another element (Y/N): n

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
```



```
Enter Your Choice: 5
Elements in Linked List:
30      20      10

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 2
Linked List is Sorted.

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
```

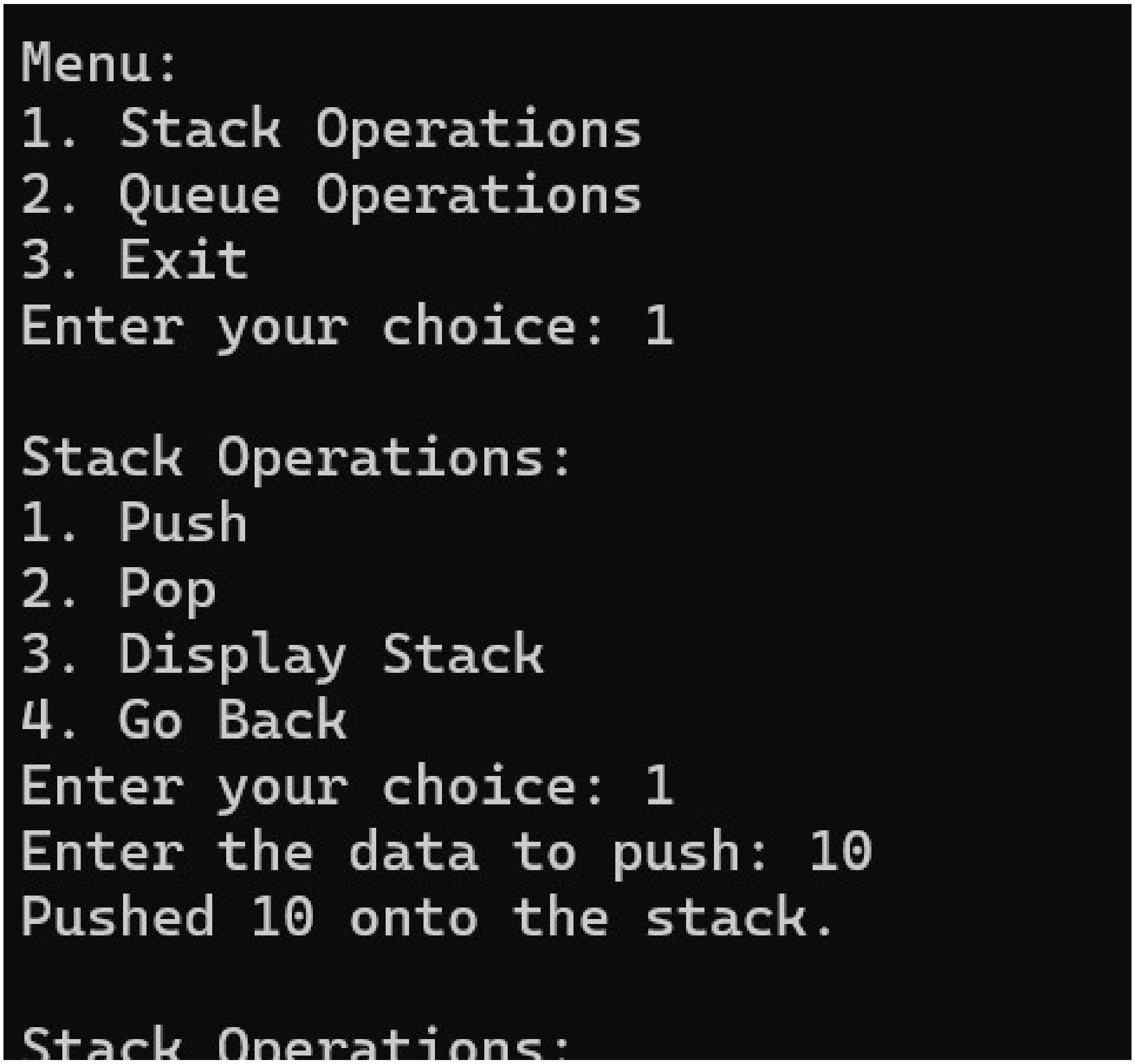


```
Enter Your Choice: 4
Enter the second linked list:
Enter Value: 70
Do you want to add another element (Y/N): y
Enter Value: 80
Do you want to add another element (Y/N): y
Enter Value: 90
Do you want to add another element (Y/N): n
Enter value for second list: 100
Do you want to add another element (Y/N): n
Lists concatenated successfully.

1. Create 1st Linked List
2. Sort Linked List
```

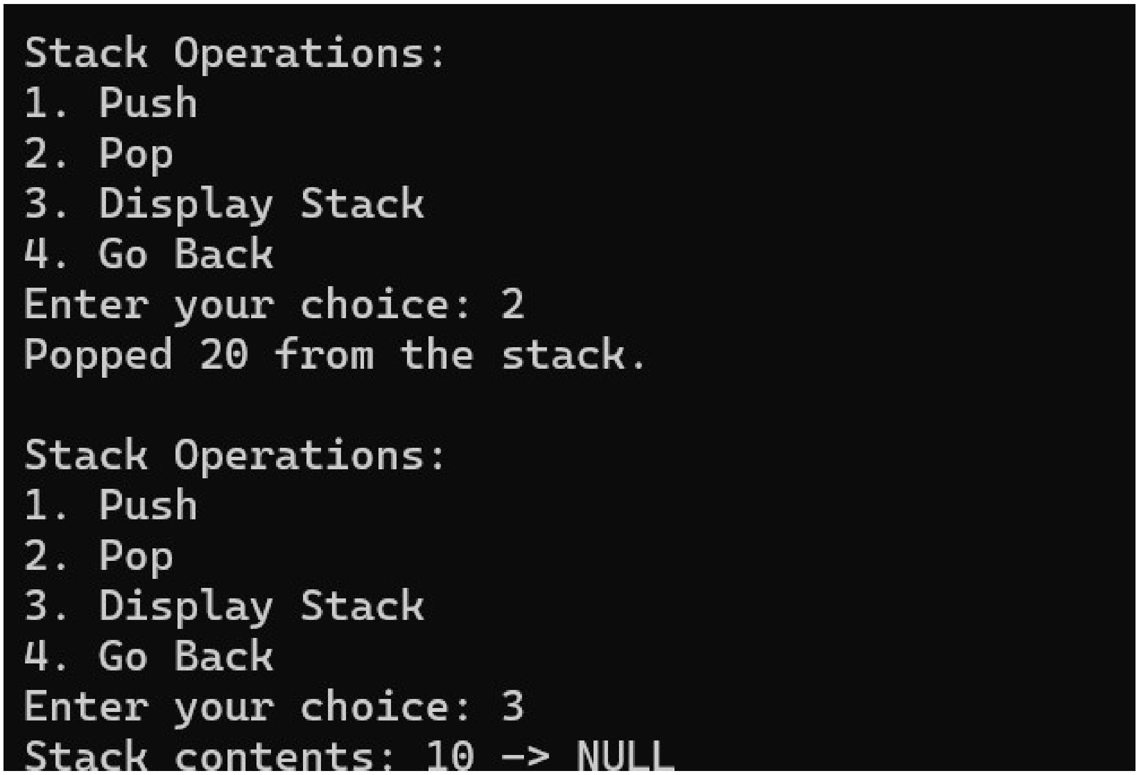
7B) WAP to Implement Single Link List to simulate Stack & Queue Operations.

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


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


struct Stack {
    struct Node* top;
};


struct Queue {
    struct Node* front;
    struct Node* rear;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

void initQueue(struct Queue* queue) {
    queue->front = queue->rear = NULL;
}


void push(struct Stack* stack, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = stack->top;
    stack->top = newNode;
    printf("Pushed %d onto the stack.\n", data);
}


void pop(struct Stack* stack) {
    if (stack->top == NULL) {
        printf("Stack underflow! The stack is empty.\n");
        return;
    }
    struct Node* temp = stack->top;
    stack->top = stack->top->next;
    printf("Popped %d from the stack.\n", temp->data);
    free(temp);
}
```

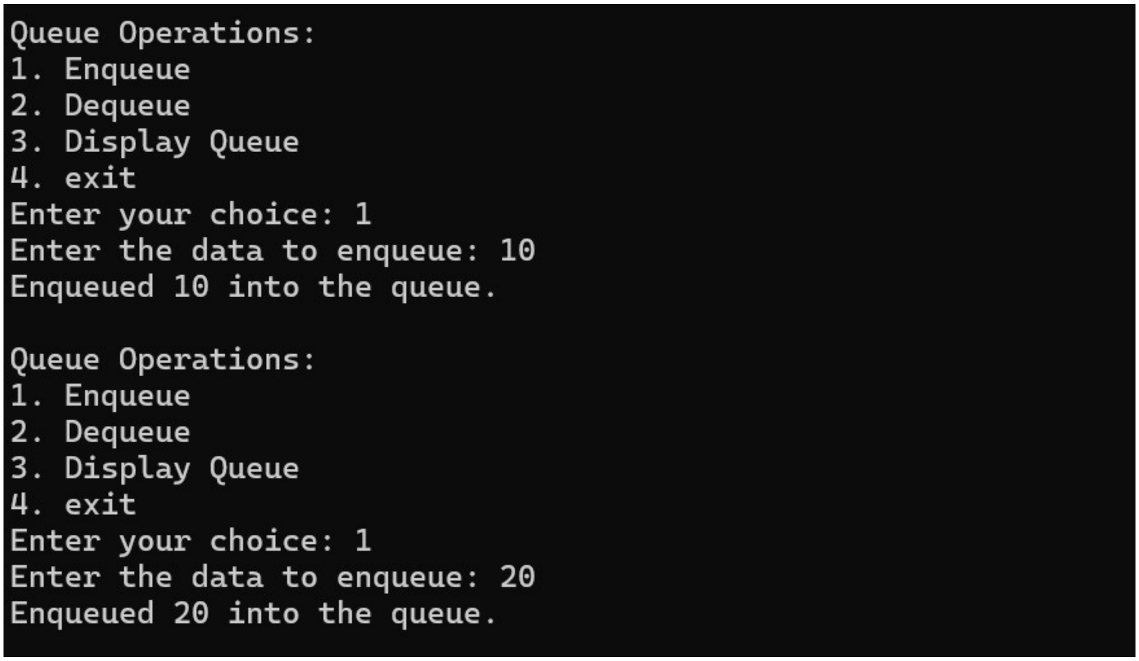
```
void displayStack(struct Stack* stack) {
    if (stack->top == NULL) {
        printf("Stack is empty.\n");
        return;
    }
    struct Node* temp = stack->top;
    printf("Stack contents: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}



void enqueue(struct Queue* queue, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
    printf("Enqueued %d into the queue.\n", data);
}

void dequeue(struct Queue* queue) {
    if (queue->front == NULL) {
        printf("Queue underflow! The queue is empty.\n");
        return;
    }
    struct Node* temp = queue->front;
    queue->front = queue->front->next;
    if (queue->front == NULL) {
        queue->rear = NULL;
    }
    printf("Dequeued %d from the queue.\n", temp->data);
    free(temp);
}

void displayQueue(struct Queue* queue) {
    if (queue->front == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    struct Node* temp = queue->front;
    printf("Queue contents: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

void menu() {
    struct Stack stack;
    struct Queue queue;
    int choice, data;

    initStack(&stack);
    initQueue(&queue);

    while (1) {
        printf("\nMenu:\n");
        printf("1. Stack Operations\n");
        printf("2. Queue Operations\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                while (1) {
                    printf("\nStack Operations:\n");
                    printf("1. Push\n");
                    printf("2. Pop\n");

                    printf("3. Display Stack\n");
                    printf("4. Go Back\n");
                    printf("Enter your choice: ");
                    scanf("%d", &choice);

                    switch (choice) {
                        case 1:
                            printf("Enter the data to push: ");
                            scanf("%d", &data);
                            push(&stack, data);
                            break;
                        case 2:
                            pop(&stack);
                            break;

                        case 3:
                            displayStack(&stack);
                            break;
                        case 4:
                            break;
                        default:
                            printf("Invalid choice. Try again.\n");
                            continue;
                    }
                    if (choice == 5) break;
                }
                break;
            case 2:
                while (1) {
                    printf("\nQueue Operations:\n");
                    printf("1. Enqueue\n");
                    printf("2. Dequeue\n");
                    printf("3. Display Queue\n");
                    printf("4. exit\n");
                    printf("Enter your choice: ");
                    scanf("%d", &choice);

                    switch (choice) {
                        case 1:
                            printf("Enter the data to enqueue: ");
                            scanf("%d", &data);
                            enqueue(&queue, data);
                            break;
                        case 2:
                            dequeue(&queue);
                            break;
                        case 3:
                            displayQueue(&queue);
                            break;
                        case 4:
                            break;
                        default:
                            printf("Invalid choice. Try again.\n");
                            continue;
                    }
                    if (choice == 4) break;
                }
                break;
            case 3:
                printf("Exiting program...\n");
                return;
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
}

// Main function
int main() {
    menu();
    return 0;
}
```



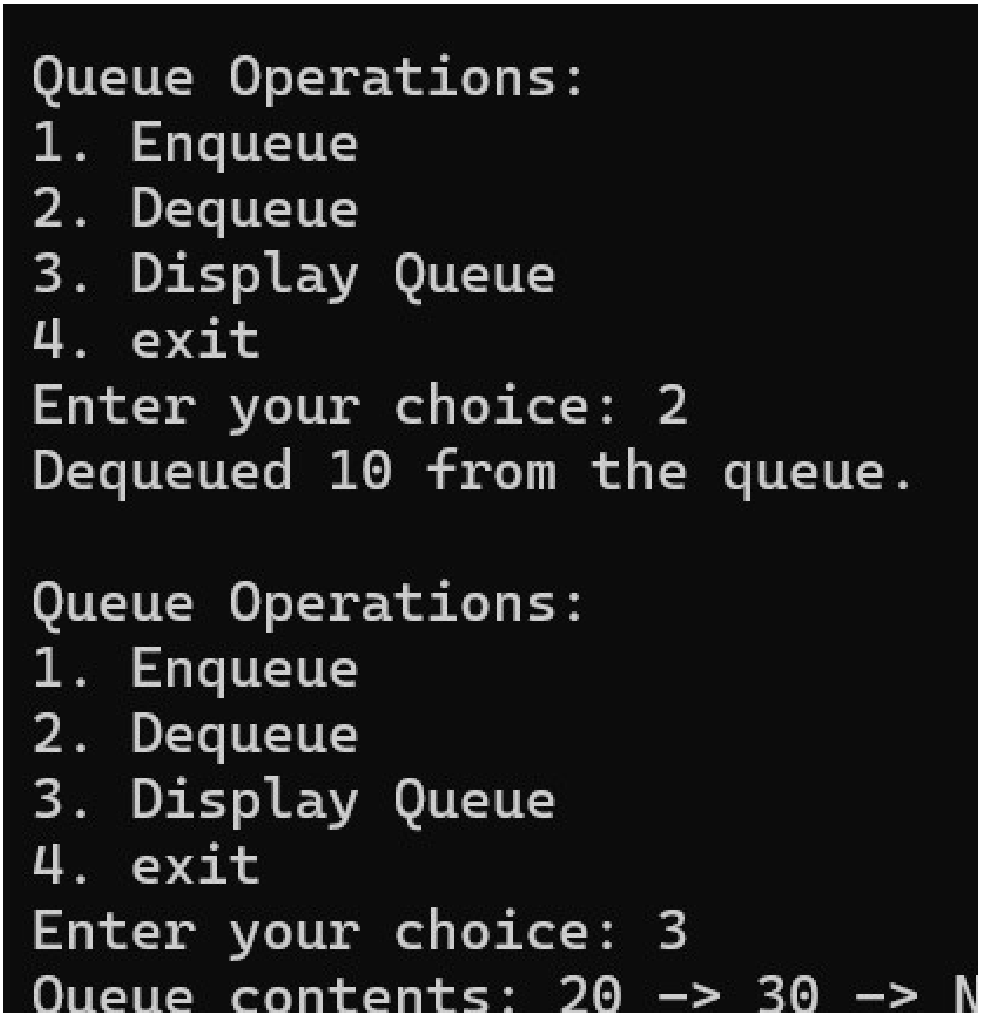
```
Menu:
1. Stack Operations
2. Queue Operations
3. Exit
Enter your choice: 1

Stack Operations:
1. Push
2. Pop
3. Display Stack
4. Go Back
Enter your choice: 1
Enter the data to push: 10
Pushed 10 onto the stack.

Stack Operations:
```



```
Stack Operations:
1. Push
2. Pop
3. Display Stack
4. Go Back
Enter your choice: 2
Popped 20 from the stack.

Stack Operations:
1. Push
2. Pop
3. Display Stack
4. Go Back
Enter your choice: 3
Stack contents: 10 -> NULL
```



```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. exit
Enter your choice: 1
Enter the data to enqueue: 10
Enqueued 10 into the queue.

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. exit
Enter your choice: 1
Enter the data to enqueue: 20
Enqueued 20 into the queue.
```

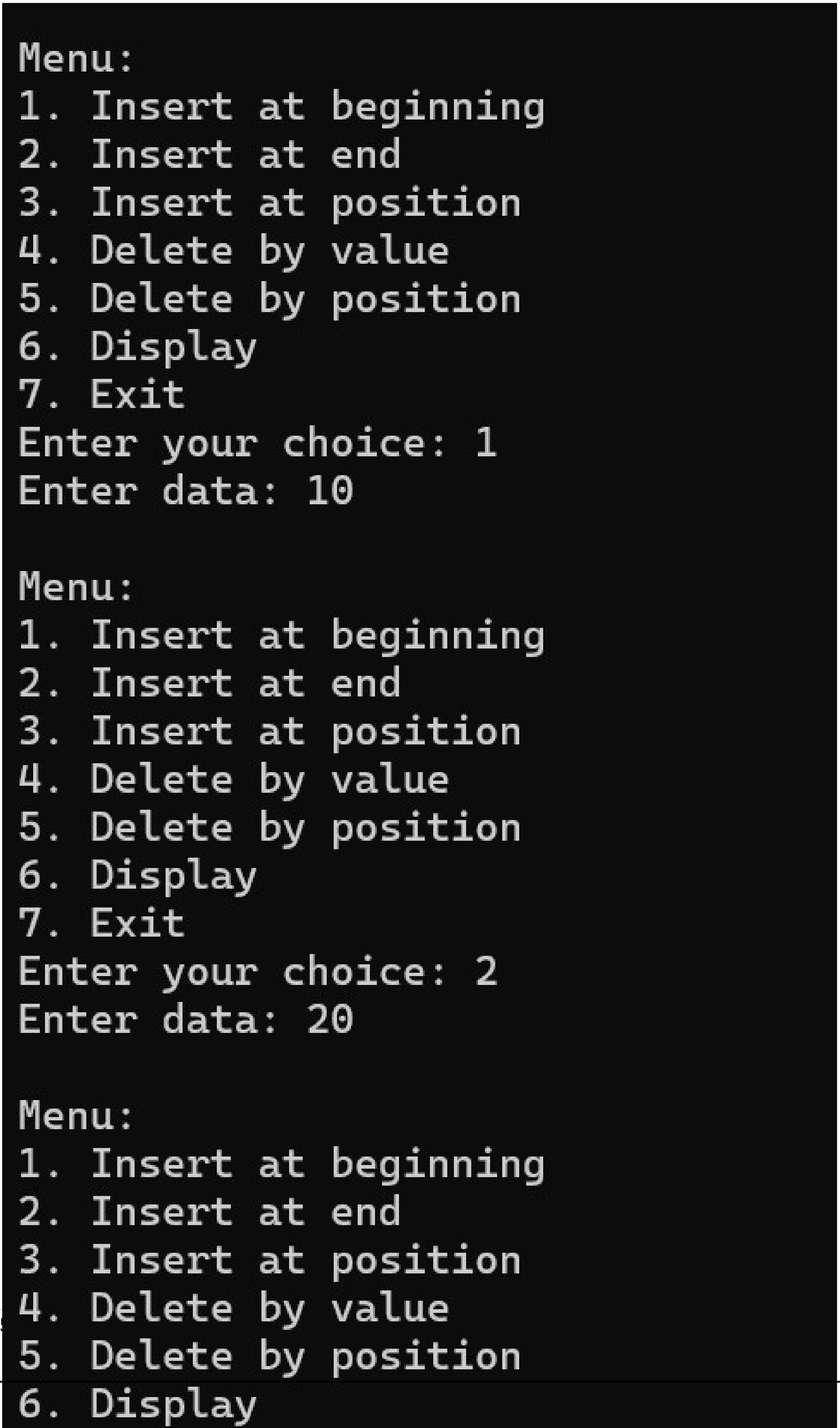


```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. exit
Enter your choice: 2
Dequeued 10 from the queue.

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. exit
Enter your choice: 3
Queue contents: 20 -> 30 -> N
```

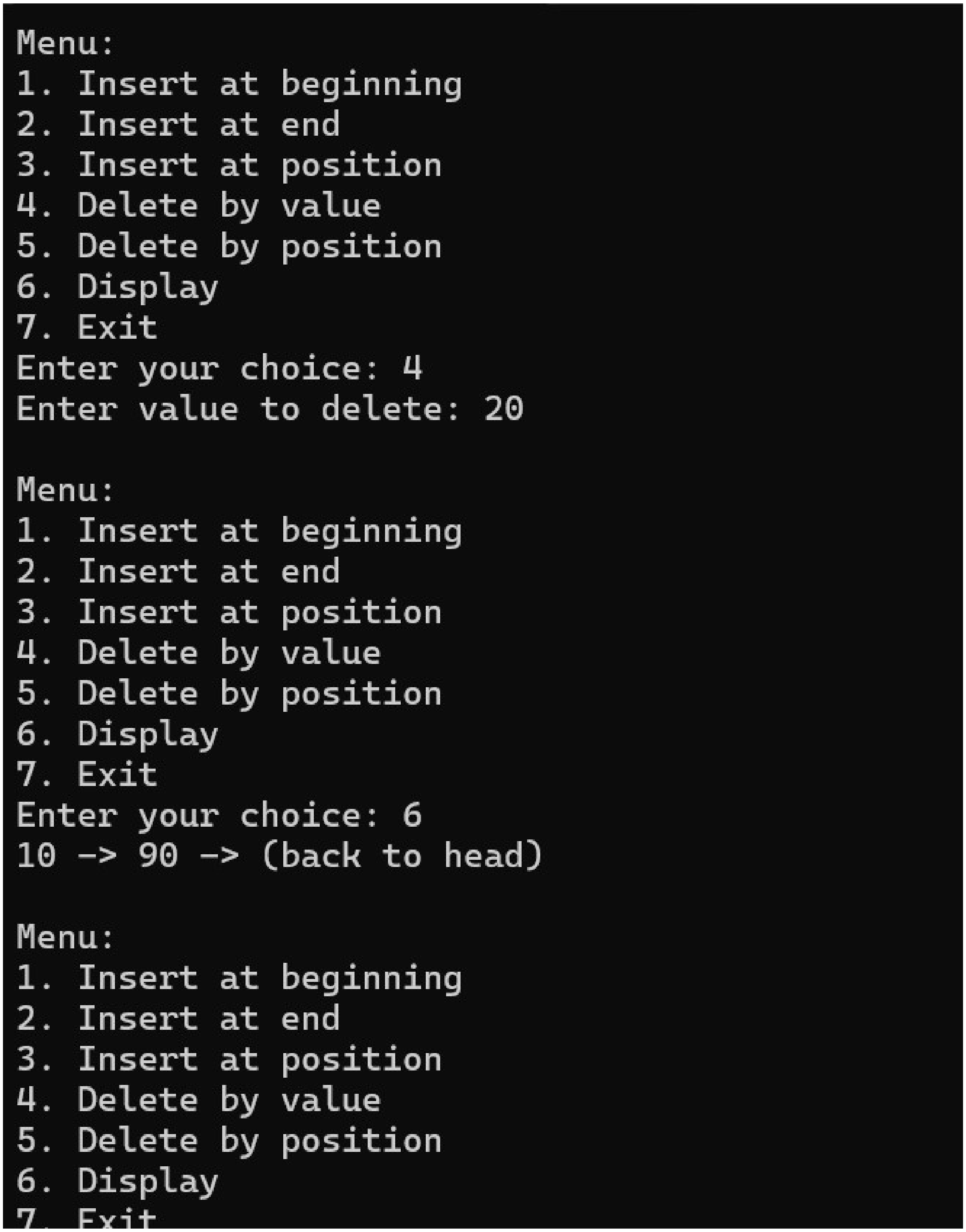
8) implementation of circular linked list

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

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

struct Node* createnew(int data) {
   struct Node* newnode = (struct Node*)malloc(sizeof(struct Node));
   newnode->data = data;
   newnode->next = newnode;
   return newnode;
}

void insert_beginning(struct Node** head, int item) {
   struct Node* newnode = createnew(item);
   if (*head == NULL) {
       *head = newnode;
   } else {
       struct Node* temp = *head;
       while (temp->next != *head) {
           temp = temp->next;
       }
       temp->next = newnode;
       newnode->next = *head;
       *head = newnode;
   }
}

void insert_end(struct Node** head, int data) {
   struct Node* newnode = createnew(data);
   if (*head == NULL) {
       *head = newnode;
   } else {
       struct Node* temp = *head;
       while (temp->next != *head) {
           temp = temp->next;
       }
       temp->next = newnode;
       newnode->next = *head;
   }
}

void insert_at_position(struct Node** head, int data, int position) {
   struct Node* newnode = createnew(data);
   if (position == 1) {
       insert_beginning(head, data);
       return;
   }
   struct Node* temp = *head;
   for (int i = 1; i < position - 1 && temp->next != *head; i++) {
       temp = temp->next;
   }
   if (temp->next == *head) return;
   newnode->next = temp->next;
   temp->next = newnode;
}

void delete_by_value(struct Node** head, int key) {
   if (*head == NULL) return;
   struct Node* temp = *head;
   struct Node* prev = NULL;
   if (temp->data == key) {
       if (temp->next == *head) {
           free(temp);
           *head = NULL;
       } else {
           while (temp->next != *head) {
               temp = temp->next;
           }
           temp->next = (*head)->next;
           free(*head);
           *head = temp->next;
       }
       return;
   }

   while (temp->next != *head && temp->data != key) {
       prev = temp;
       temp = temp->next;
   }
   if (temp == *head) return;

   prev->next = temp->next;
   free(temp);
}

void delete_by_position(struct Node** head, int position) {
   if (*head == NULL) return;
   struct Node* temp = *head;
   struct Node* prev = NULL;

   if (position == 1) {
       if (temp->next == *head) {
           free(temp);
           *head = NULL;
       } else {
           while (temp->next != *head) {
               temp = temp->next;
           }
           temp->next = (*head)->next;
           free(*head);
           *head = temp->next;
       }
       return;
   }

   for (int i = 1; i < position && temp->next != *head; i++) {
       prev = temp;
       temp = temp->next;
   }

   if (temp->next == *head) return;

   prev->next = temp->next;
   free(temp);
}

void display(struct Node* head) {
   if (head == NULL) {
       printf("List is empty\n");
       return;
   }
   struct Node* temp = head;
   do {
       printf("%d -> ", temp->data);
       temp = temp->next;
   } while (temp != head);
   printf("(back to head)\n");
}

int main() {
   struct Node* head = NULL;
   int choice, data, position;

   while (1) {
       printf("\nMenu:\n");
       printf("1. Insert at beginning\n");
       printf("2. Insert at end\n");
       printf("3. Insert at position\n");
       printf("4. Delete by value\n");
       printf("5. Delete by position\n");
printf("6. Display\n");
printf("7. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
   case 1:
       printf("Enter data: ");
       scanf("%d", &data);
       insert_beginning(&head, data);
       break;
   case 2:
       printf("Enter data: ");
       scanf("%d", &data);
       insert_end(&head, data);
       break;
   case 3:
       printf("Enter data: ");
       scanf("%d", &data);
       printf("Enter position: ");
       scanf("%d", &position);
       insert_at_position(&head, data, position);
       break;
   case 4:
       printf("Enter value to delete: ");
       scanf("%d", &data);
       delete_by_value(&head, data);
```

```
                break;
            case 5:
                printf("Enter position to delete: ");
                scanf("%d", &position);
                delete_by_position(&head, position);
                break;
            case 6:
                display(head);
                break;
            case 7:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```



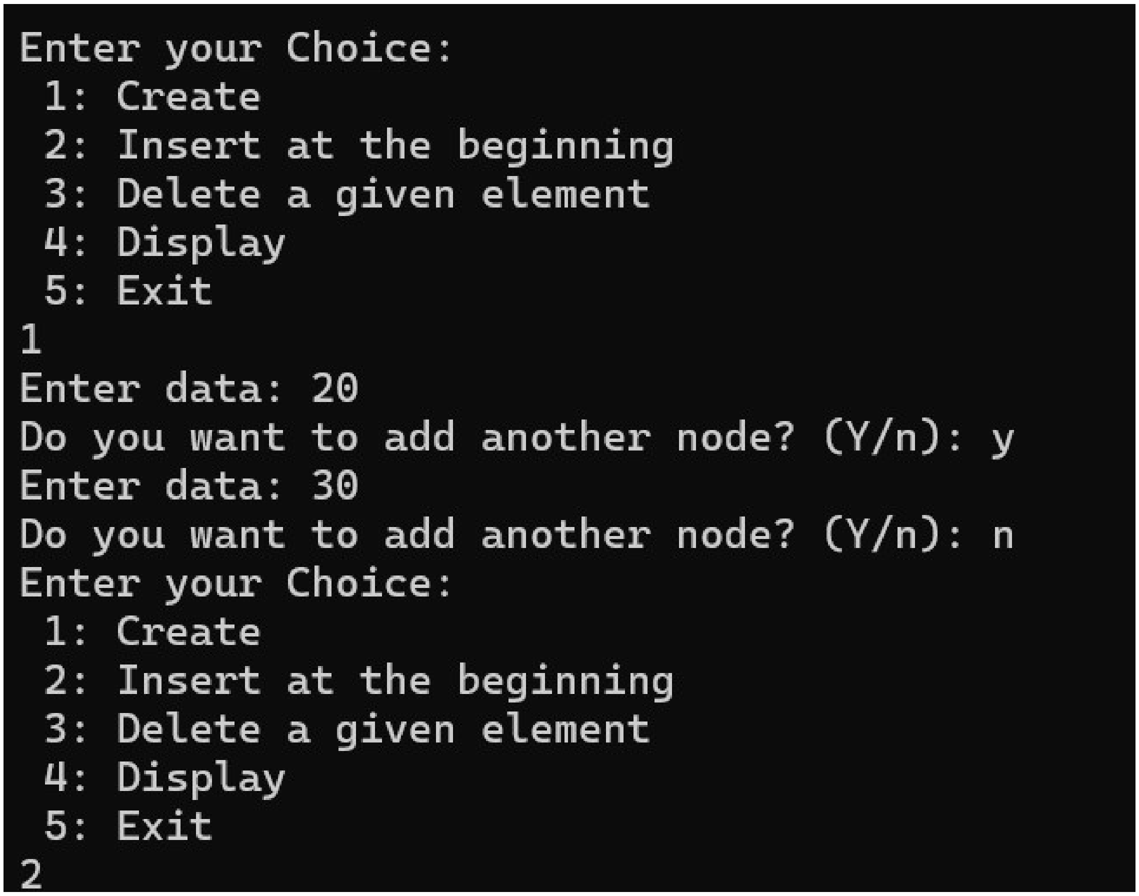
```
Menu:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Delete by value
5. Delete by position
6. Display
7. Exit
Enter your choice: 1
Enter data: 10

Menu:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Delete by value
5. Delete by position
6. Display
7. Exit
Enter your choice: 2
Enter data: 20

Menu:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Delete by value
5. Delete by position
6. Display
```



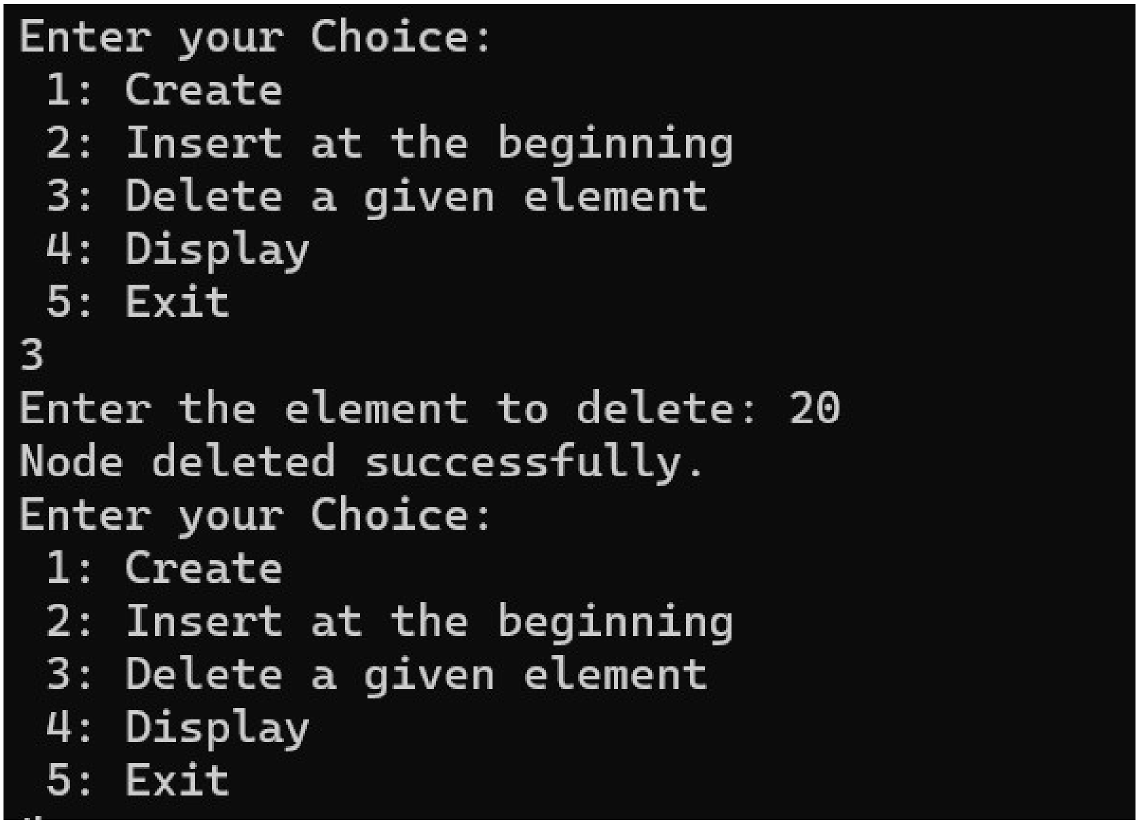
```
Menu:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Delete by value
5. Delete by position
6. Display
7. Exit
Enter your choice: 4
Enter value to delete: 20

Menu:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Delete by value
5. Delete by position
6. Display
7. Exit
Enter your choice: 6
10 -> 90 -> (back to head)

Menu:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Delete by value
5. Delete by position
6. Display
7. Exit
```

9) implementation of doubly linked list

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

void create();
void display();
void insertbeg();
void delpos();

struct Node {
    int data;
    struct Node *prev, *next;
};
typedef struct Node node;

node *new1, *start = NULL, *temp, *prev, *curr;
int ch;

void main() {
    while (1) {
        printf("Enter your Choice:\n 1: Create\n 2: Insert at the beginning\n 3: Delete a given
element\n 4: Display\n 5: Exit\n");
        scanf("%d", &ch);
        switch (ch) {
            case 1: create();
                break;
            case 2: insertbeg();
                break;
            case 3: delpos();
                break;
            case 4: display();
                break;
            case 5: exit(0);
                break;
            default: printf("Invalid choice\n");
                break;
        }
    }
}

void create() {
    char ch;
    new1 = (node*)malloc(sizeof(node));
    printf("Enter data: ");
    scanf("%d", &new1->data);
    new1->prev = NULL;
    new1->next = NULL;
    start = new1;
    curr = new1;
    while (1) {
        printf("Do you want to add another node? (Y/n): ");
        getchar();
        scanf("%c", &ch);
        if(ch == 'y' || ch == 'Y') {
            new1 = (node*)malloc(sizeof(node));
            printf("Enter data: ");
            scanf("%d", &new1->data);
            new1->prev = curr;
            curr->next = new1;
            curr = new1;
        } else {
            curr->next = NULL;
            return;
        }
    }
}

void insertbeg() {
    new1 = (node*)malloc(sizeof(node));
    printf("Enter value: ");
    scanf("%d", &new1->data);
    if (start == NULL) {
        new1->prev = NULL;
        new1->next = NULL;
        start = new1;
        return;
    }
    new1->prev = NULL;
    new1->next = start;
    start->prev = new1;
    start = new1;
}

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

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

void delpos() {
    if (start == NULL) {
        printf("Linked list is empty\n");
        return;
    }

    int ele;
    printf("Enter the element to delete: ");
    scanf("%d", &ele);


    if (start->data == ele) {
        temp = start;
        start = start->next;
        if (start != NULL) {
            start->prev = NULL;
        }
        free(temp);
        if (start == NULL) {
            printf("The list is now empty.\n");
        }
        return;
    }


    temp = start;
    while (temp != NULL && temp->data != ele) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Element not found in the list.\n");
        return;
    }


    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    }
    free(temp);
    printf("Node deleted successfully.\n");
}
```



```
Enter your Choice:
 1: Create
 2: Insert at the beginning
 3: Delete a given element
 4: Display
 5: Exit
1
Enter data: 20
Do you want to add another node? (Y/n): y
Enter data: 30
Do you want to add another node? (Y/n): n
Enter your Choice:
 1: Create
 2: Insert at the beginning
 3: Delete a given element
 4: Display
 5: Exit
2
```

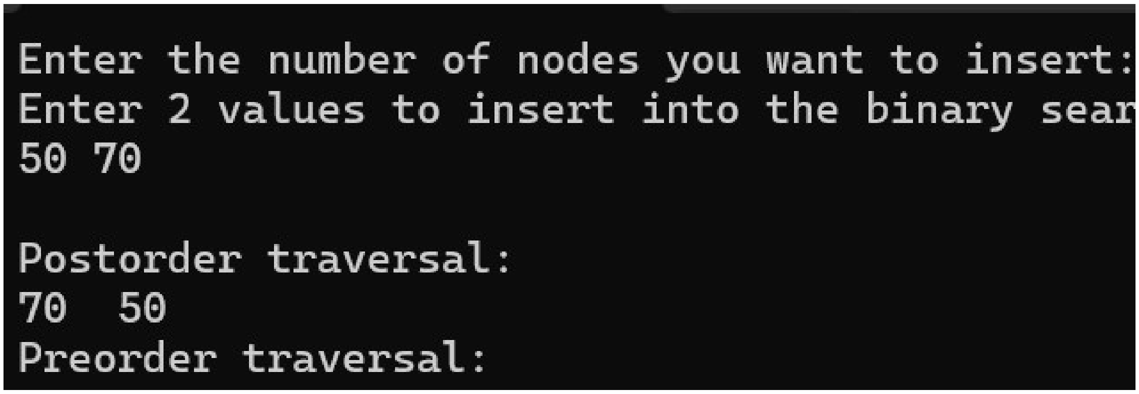


```
Enter your Choice:
 1: Create
 2: Insert at the beginning
 3: Delete a given element
 4: Display
 5: Exit
3
Enter the element to delete: 20
Node deleted successfully.
Enter your Choice:
 1: Create
 2: Insert at the beginning
 3: Delete a given element
 4: Display
 5: Exit
```

10) implementation of binary search tree

```
#include<stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node* left;

   struct Node* right;

};


struct Node* createNode(int data) {

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

   newNode->data = data;
```

```
    newNode->left=newNode->right = NULL;
    return newNode;
}
struct Node* insert(struct Node* root, int value)
{
    if(root==NULL)
    {
        return createNode(value);
    }
    if(value < root->data)
    {
        root->left=insert(root->left,value);
    }
    else
    {
        root->right=insert(root->right,value);
    }
};

struct Node*postorder(struct Node*root)
{
    if(root==NULL)
        return NULL;
    postorder(root->left);
    postorder(root->right);
    printf("%d  ",root->data);
};

struct Node*inorder(struct Node*root)
{
    if(root==NULL)
        return NULL;
    inorder(root->left);
    printf("%d  ",root->data);
    inorder(root->right);
};

struct Node*preorder(struct Node*root)
{
    if(root==NULL)
        return NULL;
    printf("%d  ",root->data);
    preorder(root->left);
    preorder(root->right);
};

int main()
{
    struct Node* root = NULL;
    int num, value;
    printf("Enter the number of nodes you want to insert: ");
    scanf("%d", &num);

    printf("Enter %d values to insert into the binary search tree:\n", num);

    for (int i = 0; i < num; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }

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

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

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

    return 0;
}
```

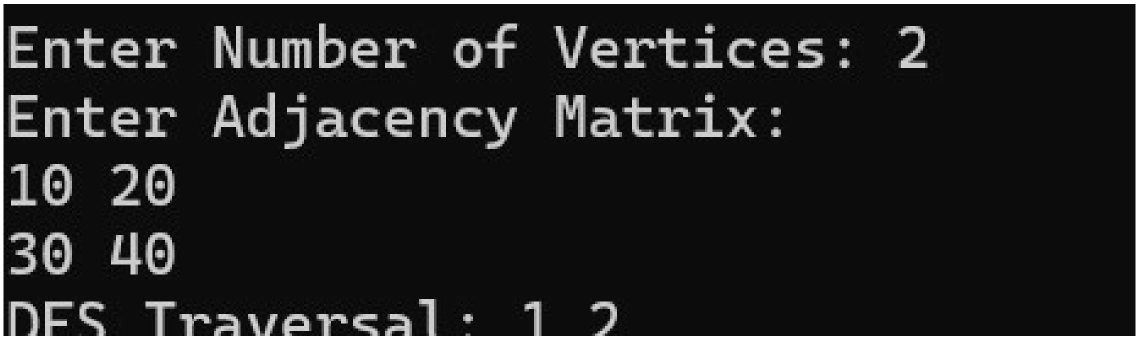


```
Enter the number of nodes you want to insert:
Enter 2 values to insert into the binary sear
50 70

Postorder traversal:
70  50
Preorder traversal:
```

11) BFS and DFS implementation

a) DFS

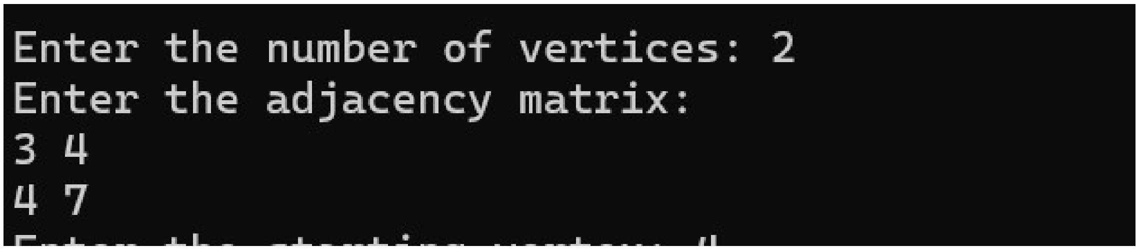
```
#include <stdio.h>

#define MAX 10

int a[MAX][MAX], vis[MAX], n;

void dfsConnected(int v) {
    vis[v] = 1;


    for (int i = 0; i < n; i++) {
        if (a[v][i] == 1 && !vis[i]) {
            dfsConnected(i);
        }
    }
}

int isConnected() {

    for (int i = 0; i < n; i++) {
        vis[i] = 0;
    }


    dfsConnected(0);
    for (int i = 0; i < n; i++) {
        if (!vis[i]) {
            return 0;
        }
    }
    return 1;
}

void dfs(int v) {
    printf("%d ", v+1);
    vis[v] = 1; // Mark the current node as visited

    for (int i = 0; i < n; i++) {
        // If there is an edge from v to i and i is not visited
        if (a[v][i] == 1 && vis[i] == 0) {
            dfs(i);
        }
    }
}



void main() {
    int i, j;

    printf("Enter Number of Vertices: ");
    scanf("%d", &n);

    printf("Enter Adjacency Matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    for (i = 0; i < n; i++) {
        vis[i] = 0; // Initialize visited array
    }

    printf("DFS Traversal: ");
    for (i = 0; i < n; i++) {
        if (vis[i] == 0) {
            dfs(i);
        }
    }

    printf("\n");
    if (isConnected()) {
        printf("The graph is connected.\n");
    }
    else {
        printf("The graph is not connected.\n");
    }
}
```



```
Enter Number of Vertices: 2
Enter Adjacency Matrix:
10 20
30 40
DFS Traversal: 1 2
```

B) BFS

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

#define MAX 100

// Queue structure for BFS
int queue[MAX], front = -1, rear = -1;

// Function to enqueue an element
void enqueue(int item) {
    if (rear == MAX - 1) {
        printf("Queue Overflow\n");
        return;
    }
    if (front == -1) front = 0;
    queue[++rear] = item;
}

// Function to dequeue an element
int dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue Underflow\n");
        return -1;
    }
    return queue[front++];
}

// BFS Function
void bfs(int graph[MAX][MAX], int visited[MAX], int start, int n) {
    int i;
    enqueue(start);
    visited[start] = 1;

    printf("BFS Traversal: ");
    while (front <= rear) {
        int current = dequeue();
        printf("%d ", current);

        for (i = 0; i < n; i++) {
            if (graph[current][i] == 1 && !visited[i]) {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}

// Main Function
void main() {
    int n, i, j, start;
    int graph[MAX][MAX], visited[MAX] = {0};

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    printf("Enter the adjacency matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

    printf("Enter the starting vertex: ");
    scanf("%d", &start);

    bfs(graph, visited, start, n);
}
```



```
Enter the number of vertices: 2
Enter the adjacency matrix:
3 4
4 7
Enter the starting vertex: 4
```