VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

ABHINAV SANJAY (1BM23CS009)

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 **ABHINAV SANJAY (1BM23CS009)**, 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. Namratha M	**Dr. Kavitha Sooda**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index

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.

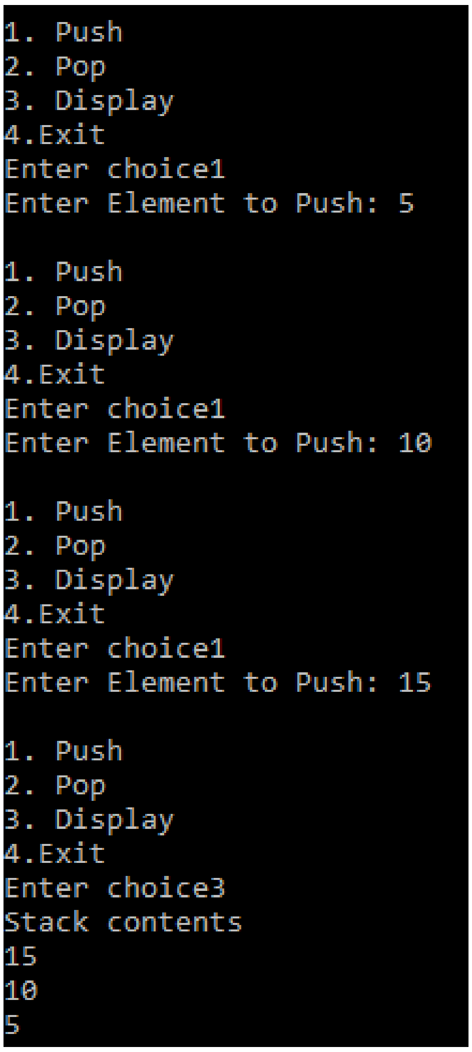
Lab program 1:

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.

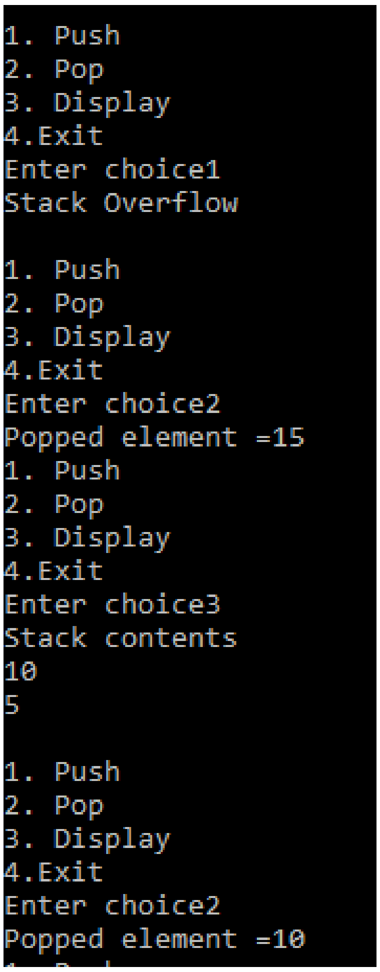
```
#include<stdio.h>
#define max 3

int s[10],top=-1,i,item,ch;
void main()
{
    while(1)
    {
        printf("\n1. Push\n2. Pop\n3. Display\n4.Exit");
        printf("\nEnter choice");
        scanf("%d",&ch);

            switch(ch)
            {
                case 1: push();
                        break;
                case 2: item=pop();
                        if(item!=-1)
                        printf("Popped element =%d",item);
                        break;
                case 3: display();
                        break;
                case 4: exit(0);
            }
    }
}
    void push(){
    if(top==max-1){
        printf("Stack Overflow\n");
    }
    else{
        top++;
        printf("Enter Element to Push: ");
        scanf("%d",&item);
        s[top]=item;
    }
}
```

```
int pop()
{
   if(top==-1){
   printf("Stack underflow");
   return -1;}
   else{
   item=s[top];
   top--;
   return(item);
}
}
void display()
{
   if(top==-1)
   {
      printf("Stack is empty");
      return;
   }
   printf("Stack contents\n");
   for(i=top;i>=0;i--)
      printf("%d\n",s[i]);
}
```

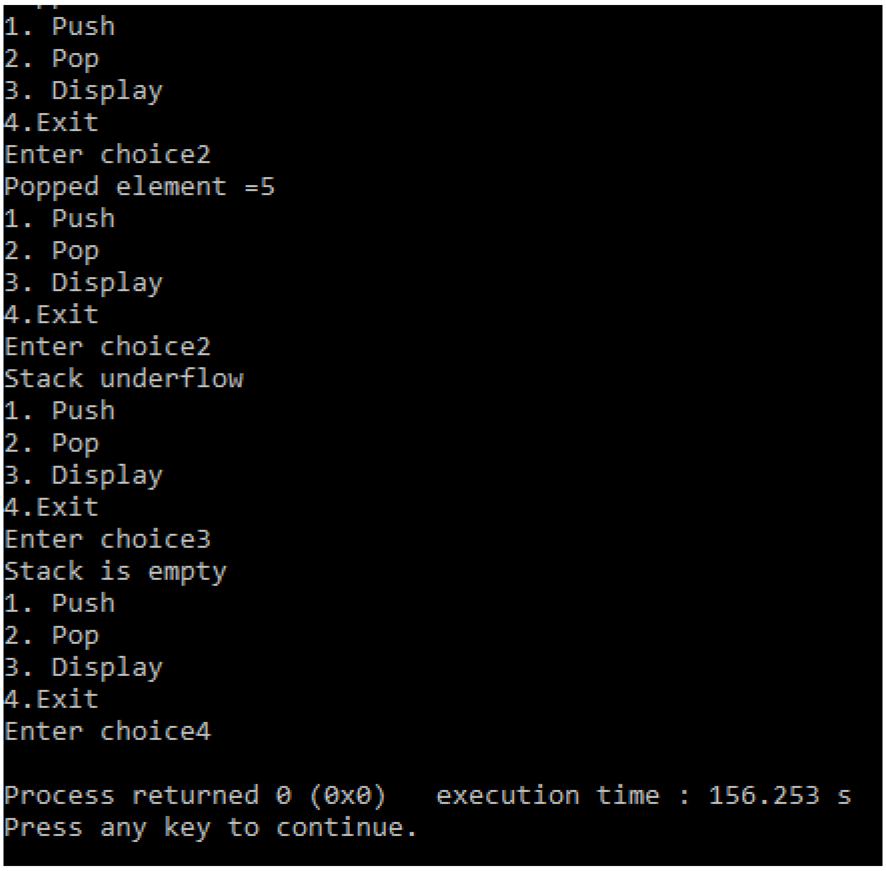
Output:



```
1. Push
2. Pop
3. Display
4.Exit
Enter choice1
Enter Element to Push: 5

1. Push
2. Pop
3. Display
4.Exit
Enter choice1
Enter Element to Push: 10

1. Push
2. Pop
3. Display
4.Exit
Enter choice1
Enter Element to Push: 15

1. Push
2. Pop
3. Display
4.Exit
Enter choice3
Stack contents
15
10
5
```



```
1. Push
2. Pop
3. Display
4.Exit
Enter choice1
Stack Overflow

1. Push
2. Pop
3. Display
4.Exit
Enter choice2
Popped element =15
1. Push
2. Pop
3. Display
4.Exit
Enter choice3
Stack contents
10
5

1. Push
2. Pop
3. Display
4.Exit
Enter choice2
Popped element =10
```



```
1. Push
2. Pop
3. Display
4.Exit
Enter choice2
Popped element =5
1. Push
2. Pop
3. Display
4.Exit
Enter choice2
Stack underflow
1. Push
2. Pop
3. Display
4.Exit
Enter choice3
Stack is empty
1. Push
2. Pop
3. Display
4.Exit
Enter choice4

Process returned 0 (0x0)   execution time : 156.253 s
Press any key to continue.
```

Lab Program 2

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

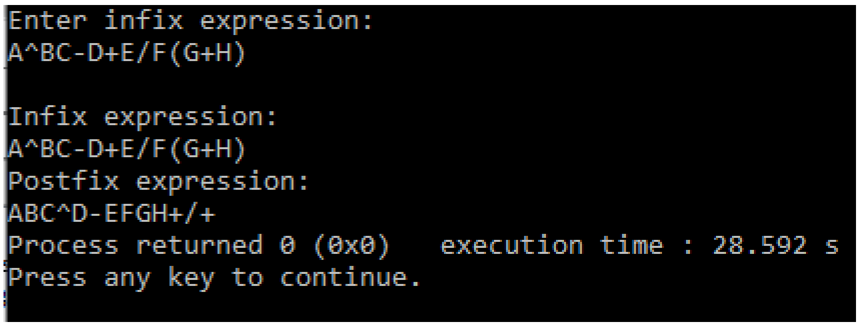
```
#include <stdio.h>

#include <conio.h>

#include <string.h>


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

char symbol, temp, infix[30], postfix[30], stack[30];


void infixToPostfix();

void push(char symbol);

char pop();

int precedence(char symb);


int main() {

    printf("Enter infix expression:\n");

    scanf("%s", infix);


    infixToPostfix();

    printf("\nInfix expression:\n%s", infix);

    printf("\nPostfix expression:\n%s", postfix);

    return 0;

}


void infixToPostfix() {

    length = strlen(infix);

    push('#');
```

```
while (index < length) {
    symbol = infix[index];
    switch (symbol) {
        case '(':
            push(symbol);
            break;

        case ')':
            temp = pop();
            while (temp != '(') {
                postfix[pos++] = temp;
                temp = pop();
            }
            break;

        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
            while (precedence(stack[top]) >= precedence(symbol)) {
                temp = pop();
                postfix[pos++] = temp;
            }
            push(symbol);
            break;
        default:
            postfix[pos++] = symbol;
    }
```

```
        index++;
    }

    while (top > 0) {
        temp = pop();
        postfix[pos++] = temp;
    }
    postfix[pos] = '\0';
}

void push(char symbol) {
    top = top + 1;
    stack[top] = symbol;
}

char pop() {
    char symb = stack[top];
    top--;
    return symb;
}

int precedence(char symbol) {
    int p;
    switch (symbol) {
        case '^':
            p = 3;
            break;

        case '*':
```

```
        case '/':
            p = 2;
            break;

        case '+':
        case '-':
            p = 1;
            break;

        case '(':
            p = 0;
            break;

        case '#':
            p = -1;
            break;
    }
    return p;
}
```

Output:



```
Enter infix expression:
A^BC-D+E/F(G+H)

Infix expression:
A^BC-D+E/F(G+H)
Postfix expression:
ABC^D-EFGH+/+
Process returned 0 (0x0)   execution time : 28.592 s
Press any key to continue.
```

Lab program 3:

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

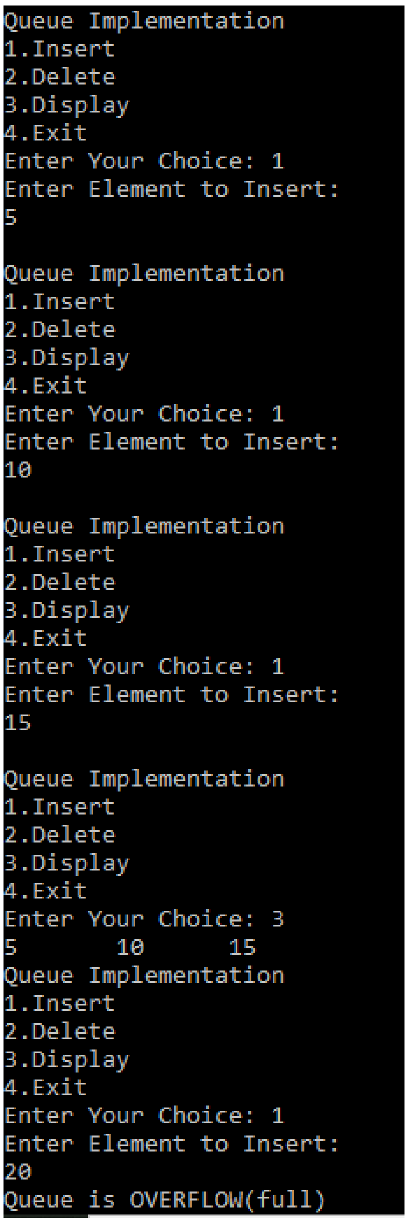
WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions.

Code for Linear Queue:

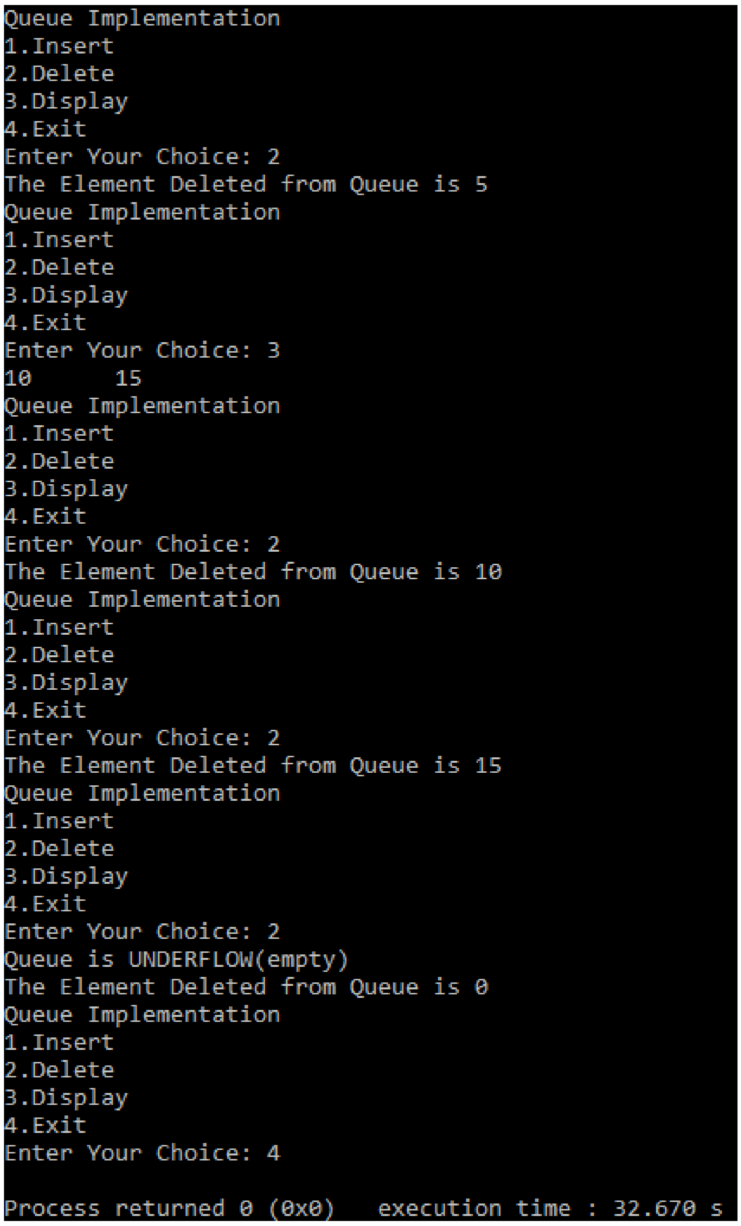
```
#include<stdio.h>
#include<conio.h>
#define max 3
int q[max],front=-1,rear=-1,ch,item,x;
void insert(int);
int del();
void display();
void main()
{
while(1){
printf("\nQueue Implementation");
printf("\n1.Insert");
printf("\n2.Delete");
printf("\n3.Display");
printf("\n4.Exit");

printf("\nEnter Your Choice: ");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("Enter Element to Insert:\n");
       scanf("%d",&item);
       insert(item);
       /*if(x==1)
       printf("Queue is Full");*/
       break;

case 2:x=del();
       printf("The Element Deleted from Queue is %d",x);
       /*if(x==1)
       printf("Queue is Empty");*/
       break;
case 3:display();
       break;

case 4:exit(0);
       break;
default:printf("INVALID Choice\n");
}
}
}

void insert(x)
{
if(rear==max-1)
printf("Queue is OVERFLOW(full) \n");
else if(rear==-1)
  {
   front=0;rear=0;
   q[rear]=x;
  }
else
  {
rear++;
q[rear]=x;
  }
}

int del()
{
if(front==-1)
printf("Queue is UNDERFLOW(empty) \n");
else if(front==rear)
 {
 x=q[front];
 front=-1;
 rear=-1;
 return(x);
 }
 else
 {
 x=q[front];
 front++;
 return(x);
 }
 }
```

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

Output:



```
Queue Implementation
1.Insert
2.Delete
3.Display
4.Exit
Enter Your Choice: 1
Enter Element to Insert:
5

Queue Implementation
1.Insert
2.Delete
3.Display
4.Exit
Enter Your Choice: 1
Enter Element to Insert:
10

Queue Implementation
1.Insert
2.Delete
3.Display
4.Exit
Enter Your Choice: 1
Enter Element to Insert:
15

Queue Implementation
1.Insert
2.Delete
3.Display
4.Exit
Enter Your Choice: 3
5       10      15
Queue Implementation
1.Insert
2.Delete
3.Display
4.Exit
Enter Your Choice: 1
Enter Element to Insert:
20
Queue is OVERFLOW(full)
```



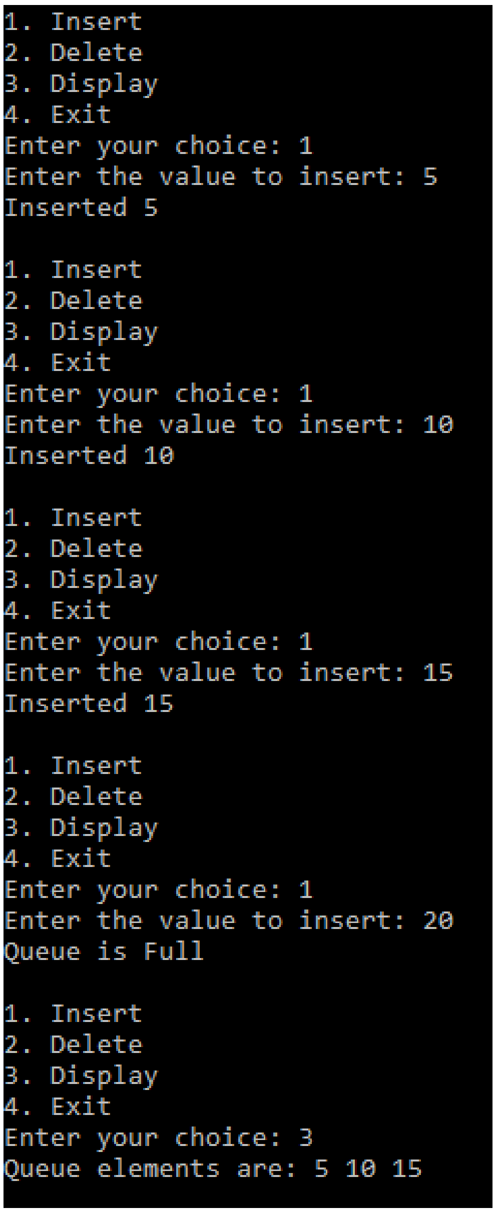
```
Queue Implementation
1.Insert
2.Delete
3.Display
4.Exit
Enter Your Choice: 2
The Element Deleted from Queue is 5
Queue Implementation
1.Insert
2.Delete
3.Display
4.Exit
Enter Your Choice: 3
10      15
Queue Implementation
1.Insert
2.Delete
3.Display
4.Exit
Enter Your Choice: 2
The Element Deleted from Queue is 10
Queue Implementation
1.Insert
2.Delete
3.Display
4.Exit
Enter Your Choice: 2
The Element Deleted from Queue is 15
Queue Implementation
1.Insert
2.Delete
3.Display
4.Exit
Enter Your Choice: 2
Queue is UNDERFLOW(empty)
The Element Deleted from Queue is 0
Queue Implementation
1.Insert
2.Delete
3.Display
4.Exit
Enter Your Choice: 4

Process returned 0 (0x0)   execution time : 32.670 s
```

Code for Circular Queue :

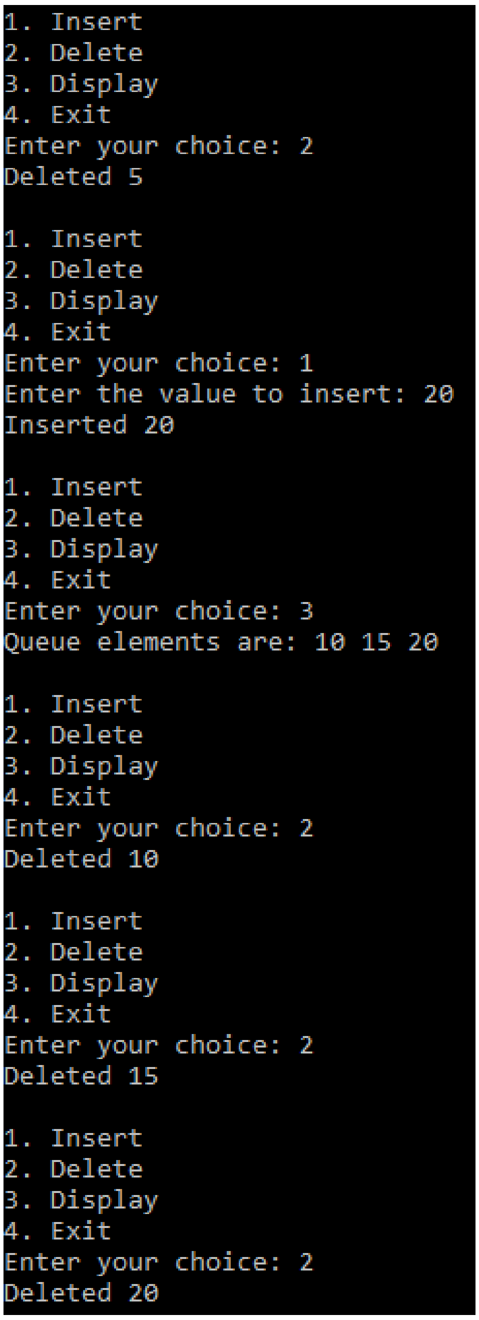
```
#include <stdio.h>

#include <stdlib.h>

#define MAX 3 // Define the maximum size of the circular queue


// Global variables for circular queue

int queue[MAX], front = -1,rear = -1,ch,value;


void main() {


    while(1){

        printf("\nMenu:\n");

        printf("1. Enqueue\n");

        printf("2. Dequeue\n");

        printf("3. Display\n");

        printf("4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &ch);


        switch (ch) {

            case 1:

                printf("Enter value to enqueue: ");

                scanf("%d", &value);

                enqueue(value);

                break;

            case 2:

                value = dequeue();

                if (value != -1) {

                    printf("Dequeued: %d\n", value);
                }
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                exit(0);
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

}

// Function to check if the queue is full
/*int isFull() {
    return (rear + 1) % MAX == front;
}*/

// Function to check if the queue is empty
/*int isEmpty() {
    return front == -1;
}*/

// Function to add an item to the queue
void enqueue(value) {
    if ((front==rear + 1) % MAX) {
        printf("Queue is full!\n");
    } else {
        if (front==-1) {
            front = 0; // Initialize front
        }
        rear = (rear + 1) % MAX; // Circular increment
        queue[rear] = value;
        printf("Inserted %d\n", value);
    }
}

// Function to remove an item from the queue
int dequeue() {
    if (front==-1,rear==-1) {
        printf("Queue is empty!\n");
        return -1;// Indicating the queue is empty
    } else {
        value = queue[front];
        if (front == rear) {
            // Queue has only one element, reset queue after dequeue
            front = -1;
            rear = -1;
        } else {
            front = (front + 1) % MAX; // Circular increment
        }
        return value;
    }
}
```

```
// Function to display the queue
void display() {
    if (front==-1) {
        printf("Queue is empty!\n");
    } else {
        printf("Queue elements: ");
        int i = front;
        while (1) {
            printf("%d ", queue[i]);
            if (i == rear) break;
            i = (i + 1) % MAX;
        }
        printf("\n");
    }
}
```

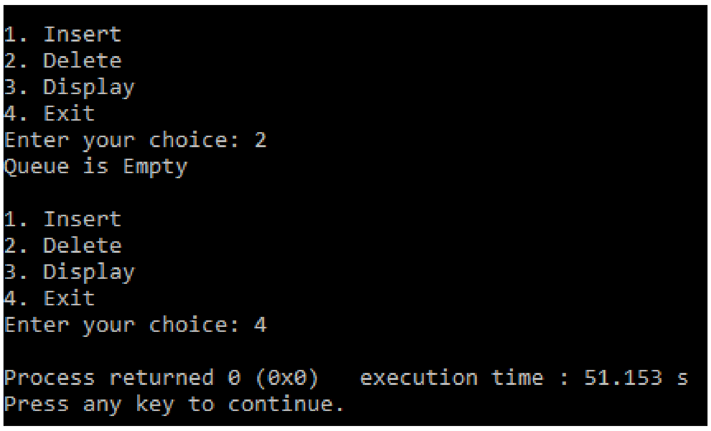
Output:



```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 5
Inserted 5

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 10
Inserted 10

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 15
Inserted 15

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 20
Queue is Full

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 5 10 15
```



```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 5

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 20
Inserted 20

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 10 15 20

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 10

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 15

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 20
```



```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Queue is Empty

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4

Process returned 0 (0x0)   execution time : 51.153 s
Press any key to continue.
```

Leetcode:

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if:

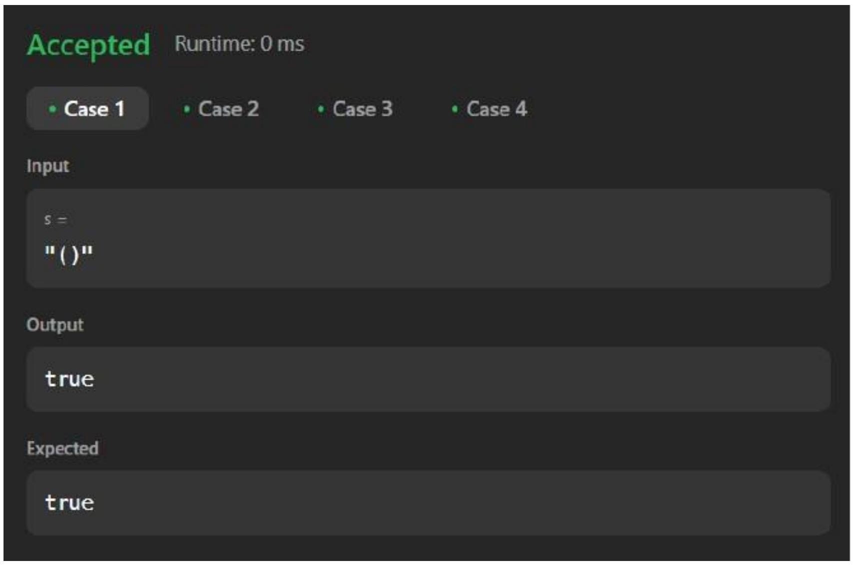
Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Every close bracket has a corresponding open bracket of the same type.

```
#include <stdbool.h>
#include <string.h>

bool isValid(char* s) {
    int n = strlen(s);
    char stack[n];
    int top = -1;
    int i = 0;
    while (i < n)
    {
        char c = s[i];
        if (c == '(' || c == '{' || c == '[')
            stack[++top] = c;
        else
        {
            if (top == -1)
                return false;
            char topChar = stack[top--];
            if ((c == ')' && topChar != '(') ||
                (c == '}' && topChar != '{') ||
                (c == ']' && topChar != '['))
                return false;
        }
        i++;
    }
    return top == -1;
}
```

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3 • Case 4

Input

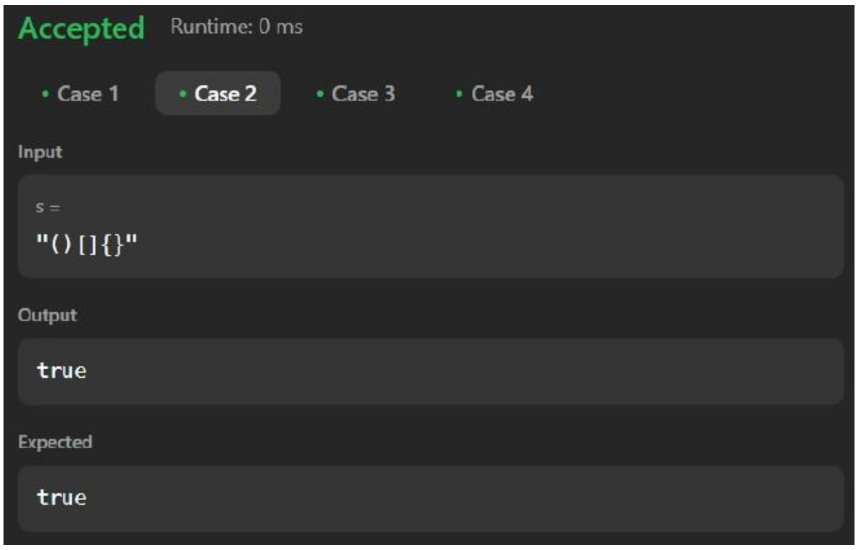
```
s =
"()"
```

Output

```
true
```

Expected

```
true
```



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3 • Case 4

Input

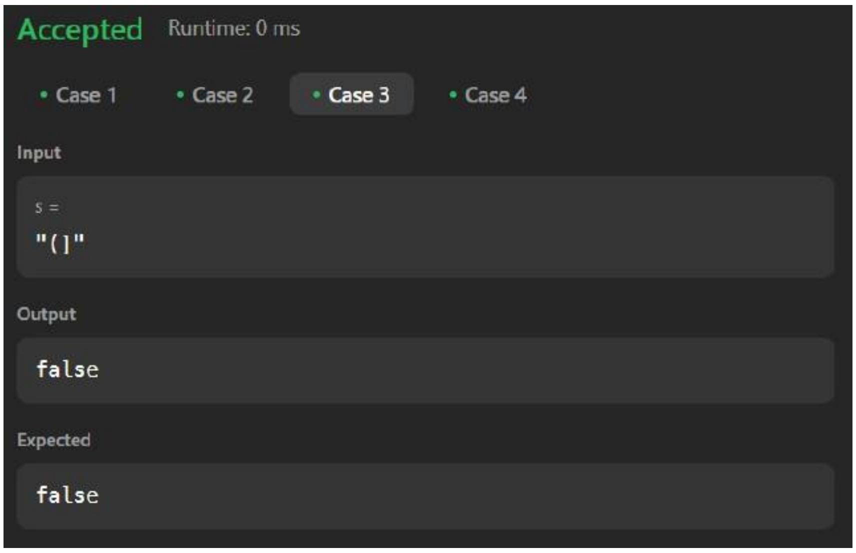
```
s =
"()[]{}"
```

Output

```
true
```

Expected

```
true
```



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3 • Case 4

Input

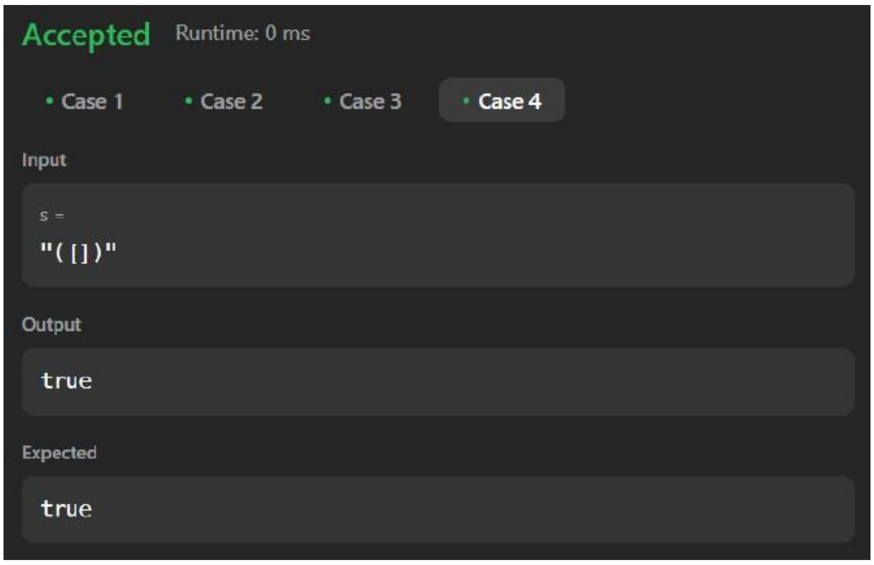
```
s =
"(]"
```

Output

```
false
```

Expected

```
false
```


Accepted Runtime: 0 ms

Case 1 Case 2 Case 3 Case 4

Input

```
s =
"([])"
```

Output

```
true
```

Expected

```
true
```

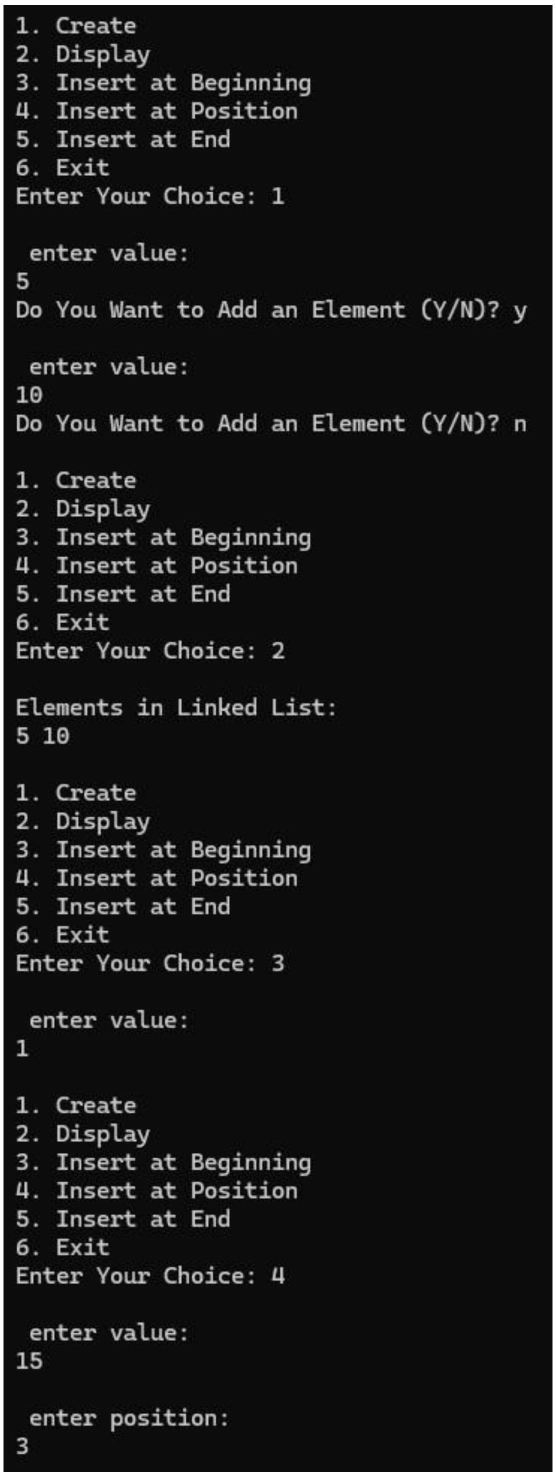
Lab program 4:

WAP to Implement Singly Linked List with following operations
a) Create a linked list.
b) Insertion of a node at first position, at any position and at end of list.
Display the contents of the linked list.

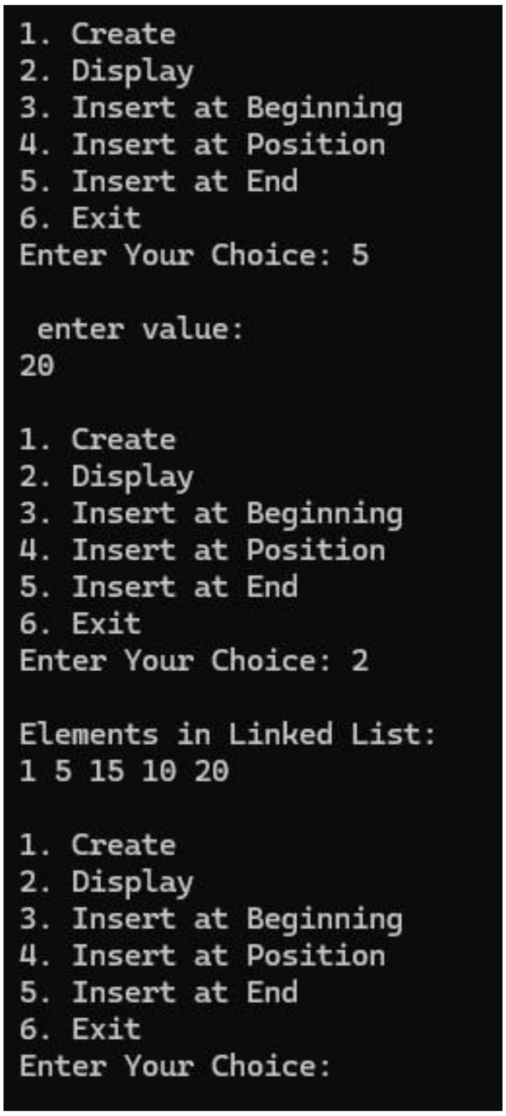
```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   int data;          // Data for the node
   struct Node *link;  // Pointer to the next node in the list
};

typedef struct Node node;
node *start = NULL;  // Start of the linked list, initially NULL
node *new1, *curr, *ptr; // Global declaration for new1, curr, and ptr

// Function prototypes
void create();
void display();
void InsertStart();
void InsertPosition();
void InsertEnd();

void main() {
   int ch;
   while (1) {
      printf("\n1. Create \n2. Display \n3. Insert at Beginning \n4. Insert at Position \n5. Insert
at End \n6. Exit");
      printf("\nEnter Your Choice: ");
      scanf("%d", &ch);

      switch (ch) {
         case 1:
            create();
            break;
         case 2:
            display();
            break;
         case 3:
            InsertStart();
            break;
            case 4:
                InsertPosition();
                break;
            case 5:
                InsertEnd();
                break;
            case 6:
                exit(0);
            default:
                printf("Enter a Number between 1 and 6.\n");
        }
    }
}

void create() {
    char ch;

    do {
        new1 = (node*)malloc(sizeof(node));
       printf("\n enter value:\n");
       scanf("%d",&new1->data);
       if (start==NULL)
       {
           start=new1;
           curr=new1;
       }
       else {
           curr->link = new1;
           curr=new1;
       }

        printf("Do You Want to Add an Element (Y/N)? ");
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
    curr->link=NULL;
}

void InsertStart() {
    new1 = (node*)malloc(sizeof(node));
    printf("\n enter value:\n");
    scanf("%d",&new1->data);
    if(start==NULL)
    {
        start=new1;
        new1->link=NULL;
        return;
    }
    else {
        new1->link=start;
        start=new1;
        return;
    }
}

void InsertPosition() {
    new1 = (node*)malloc(sizeof(node));
    printf("\nEnter value to insert: ");
    scanf("%d", &new1->data);

    if (start == NULL) {
        start = new1;
        new1->link = NULL;
        return;
    }

    int i = 1, pos;
    ptr = start;
    printf("\nEnter position: ");
    scanf("%d", &pos);

    if (pos == 1) {
        new1->link = start;
        start = new1;
        return;
    }

    while (ptr != NULL && i < pos - 1) {
        ptr = ptr->link;
        i++;
    }

    if (ptr == NULL) {
        printf("Position is out of range.\n");
        return;
    }

    new1->link = ptr->link;
    ptr->link = new1;
```

```
}

void InsertEnd() {
    new1 = (node*)malloc(sizeof(node));
    printf("\n enter value:\n");
    scanf("%d",&new1->data);
    if(start==NULL)
    {
        start=new1;
        new1->link=NULL;
        return;
    }

    ptr=start;
    while(ptr->link !=NULL)
    {
        ptr=ptr->link;
    }
    ptr->link=new1;
    new1->link=NULL;
    return;
}

// Display all nodes in the linked list
void display() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }

    ptr = start;
    printf("\nElements in Linked List: \n");

    while (ptr != NULL) {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
    printf("\n");
}
```

Output:



```
1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 1

 enter value:
5
Do You Want to Add an Element (Y/N)? y

 enter value:
10
Do You Want to Add an Element (Y/N)? n

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
5 10

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 3

 enter value:
1

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 4

 enter value:
15

 enter position:
3
```



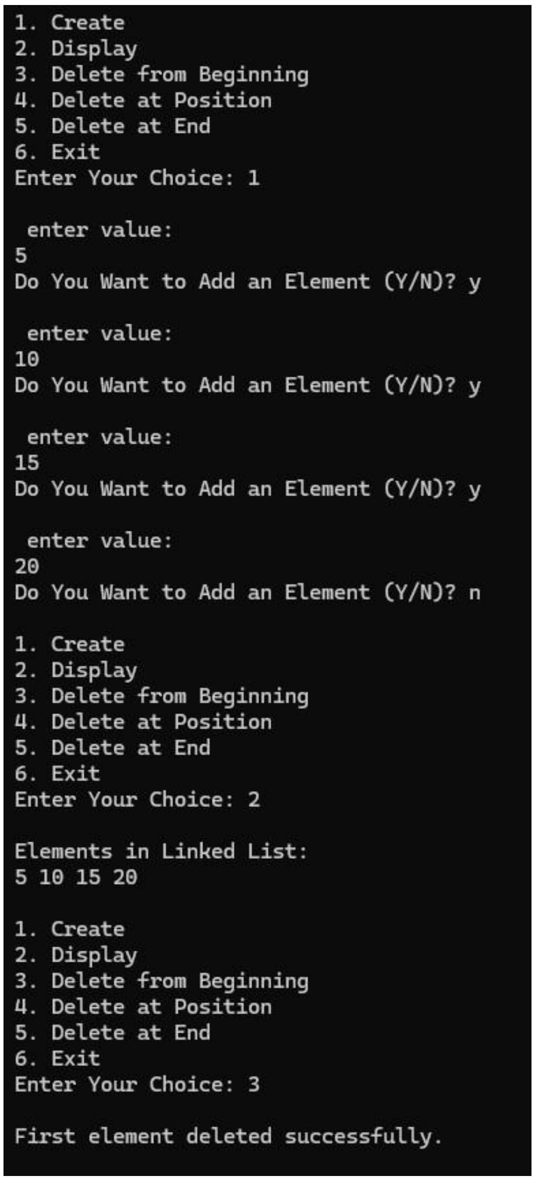
```
1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 5

 enter value:
20

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
1 5 15 10 20

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice:
```

Lab Program 5:

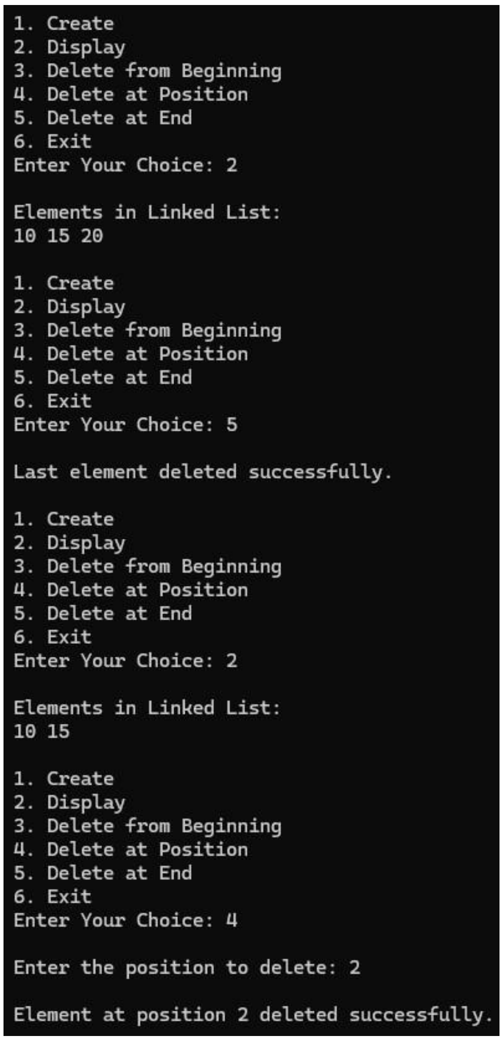
WAP to Implement Singly Linked List with following operations
a) Create a linked list.
b) Deletion of first element, specified element and last element in the list.
c) Display the contents of the linked list.

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

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

typedef struct Node node;
node *start = NULL;
node *new1, *curr, *temp, *prev;

void create();
void display();
void DeletefromStart();
void DeleteatPosition();
void DeleteatEnd();

void main() {
    int ch;
    while (1) {
        printf("\n1. Create \n2. Display \n3. Delete from Beginning \n4. Delete at Position \n5.
Delete at End \n6. Exit");
        printf("\nEnter Your Choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                create();
                break;
            case 2:
                display();
                break;
            case 3:
                DeletefromStart();
                break;
            case 4:
                DeleteatPosition();
                break;
            case 5:
                DeleteatEnd();
                break;
            case 6:
                exit(0);
            default:
                printf("Enter a Number between 1 and 6.\n");
        }
    }
}

void create() {
    char ch;

    do {
        new1 = (node*)malloc(sizeof(node));
       printf("\n enter value:\n");
       scanf("%d",&new1->data);
       if (start==NULL)
       {
           start=new1;
           curr=new1;
       }
       else {
           curr->link = new1;
           curr=new1;
       }

        printf("Do You Want to Add an Element (Y/N)? ");
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
    curr->link=NULL;
}

void display() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }

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

void DeletefromStart() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }

    temp = start;
    start = start->link;
    free(temp);
    printf("\nFirst element deleted successfully.\n");
}

void DeleteatPosition() {
    int pos, i = 1;
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }

    printf("\nEnter the position to delete: ");
    scanf("%d", &pos);

    temp = start;
    prev = NULL;

    if (pos == 1) {
        start = temp->link;
        free(temp);
        printf("\nElement at position %d deleted successfully.\n", pos);
        return;
    }

    while (temp != NULL && i < pos) {
        prev = temp;
        temp = temp->link;
        i++;
    }

    if (temp == NULL) {
        printf("\nPosition not found.\n");
        return;
    }

    prev->link = temp->link;
    free(temp);
    printf("\nElement at position %d deleted successfully.\n", pos);
}

void DeleteatEnd() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }

    temp = start;
    prev = NULL;

    if (start->link == NULL) {
        start = NULL;
        free(temp);
        printf("\nLast element deleted successfully.\n");
        return;
    }

    while (temp->link != NULL) {
        prev = temp;
        temp = temp->link;
    }

    prev->link = NULL;
    free(temp);
    printf("\nLast element deleted successfully.\n");
}
```

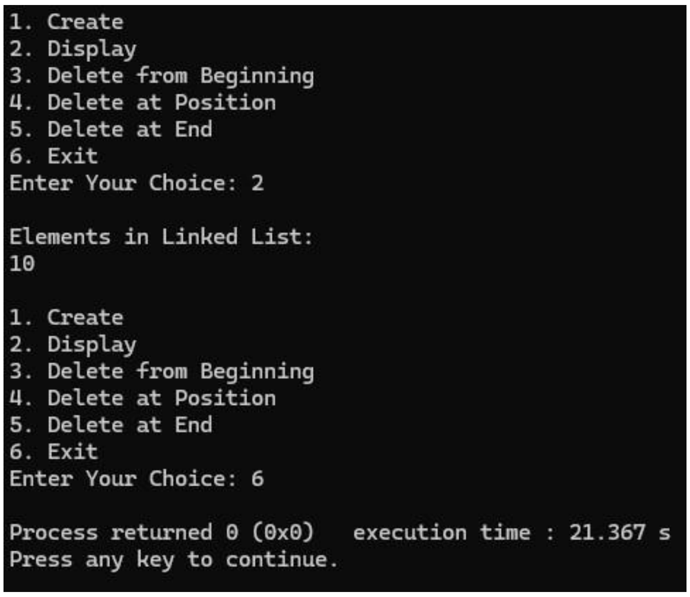
Output:



```
1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 1

 enter value:
5
Do You Want to Add an Element (Y/N)? y

 enter value:
10
Do You Want to Add an Element (Y/N)? y

 enter value:
15
Do You Want to Add an Element (Y/N)? y

 enter value:
20
Do You Want to Add an Element (Y/N)? n

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
5 10 15 20

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 3

First element deleted successfully.
```



```
1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
10 15 20

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 5

Last element deleted successfully.

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
10 15

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 4

Enter the position to delete: 2

Element at position 2 deleted successfully.
```



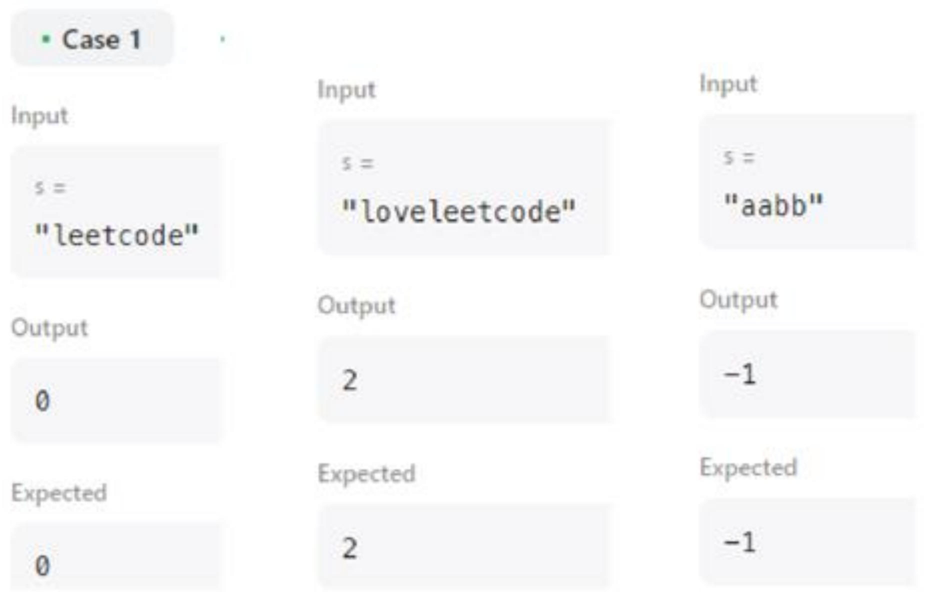
```
1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
10

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 6

Process returned 0 (0x0)   execution time : 21.367 s
Press any key to continue.
```

Leetcode:

Given a string s, find the first non repeating character in it and return its index. If it does not exist, return -1.

```
int firstUniqChar(char* s) {
    int flag = 0 , i , j;
    for( i = 0 ; s[i] ; i++ ) {
        for( j = 0 ; s[j] ; j++ , flag = 0 ) {
            if( ( s[i] == s[j] ) && ( i != j ) ) {
                flag = 1;
                break;
            }
        }
        if(flag == 0)
        return i;
    }
    return -1;
}
```

Output:



```
Case 1

Input
s =
"leetcode"
Output
0
Expected
0

Input
s =
"loveleetcode"
Output
2
Expected
2

Input
s =
"aabb"
Output
-1
Expected
-1
```

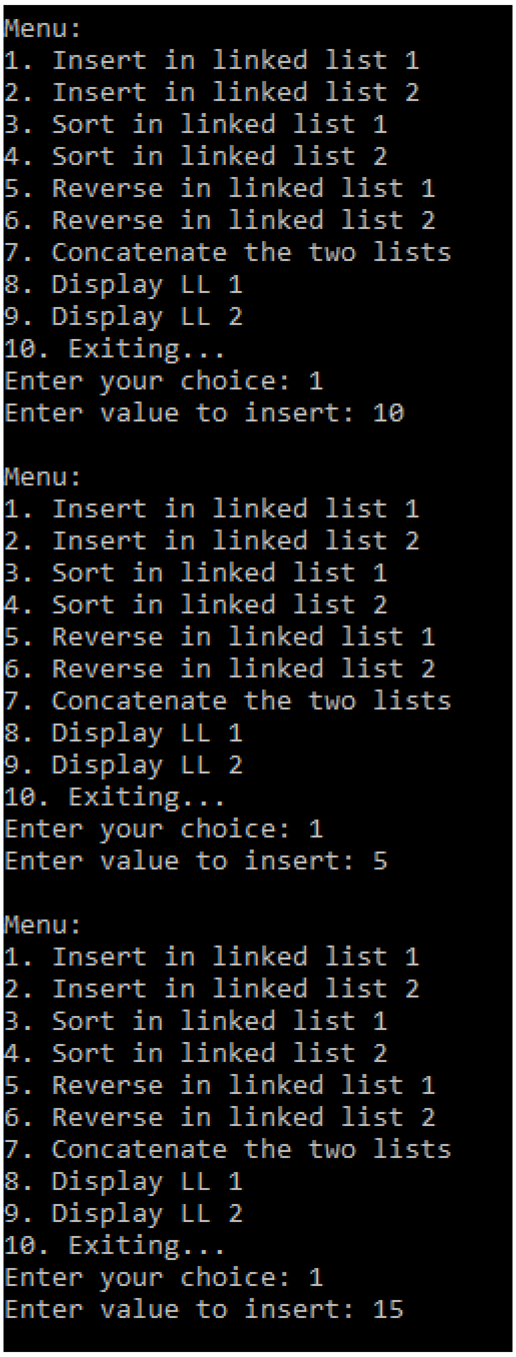
Lab program 6:

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

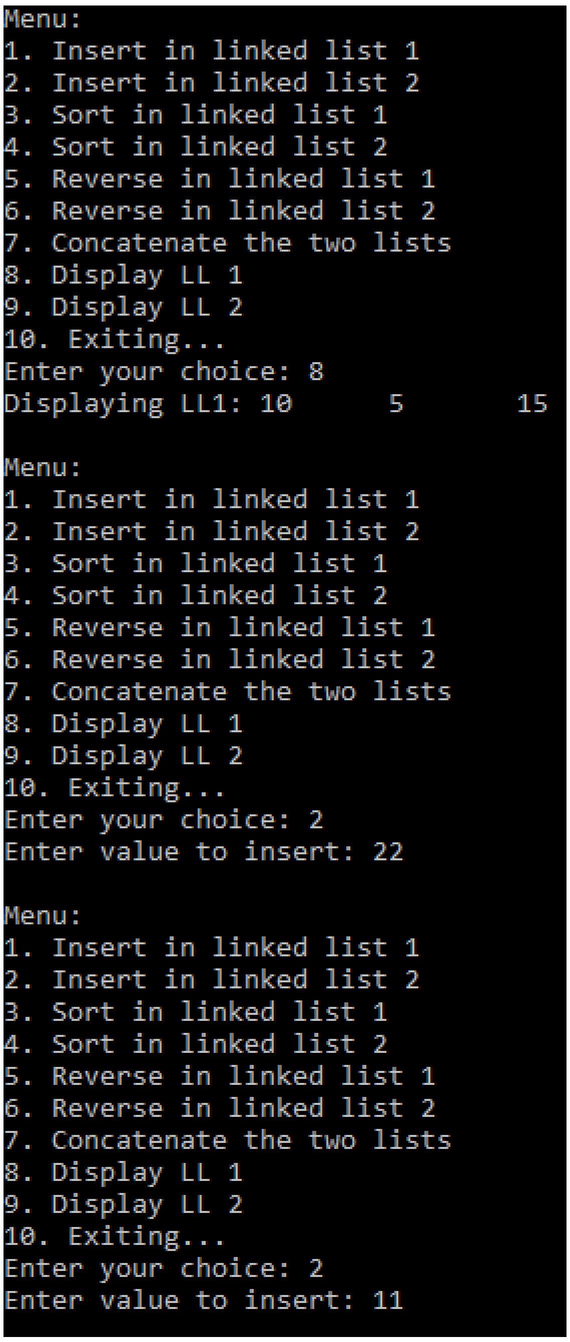
```
#include <stdio.h>
#include <stdlib.h>

struct node {
    int value;
    struct node *next;
};

typedef struct node* NODE;

NODE getnode() {
    NODE new_node = (NODE)malloc(sizeof(struct node));
    if (new_node == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    new_node->next = NULL;
    return new_node;
}

NODE insert_end(int item, NODE first) {
    NODE new_end = getnode();
    new_end->value = item;
    new_end->next = NULL;

    if (first == NULL) {
        return new_end;
    }

    NODE current = first;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = new_end;
    return first;
}

NODE reverse(NODE first) {
    NODE current = NULL;
    NODE temp;
    while (first != NULL) {
        temp = first;
        first = first->next;
        temp->next = current;
        current = temp;
    }
    return current;
}

NODE concatenate(NODE first_1, NODE first_2) {
    if (first_1 == NULL) {
        return first_2;
    }
    if (first_2 == NULL) {
        return first_1;
    }

    NODE last1 = first_1;
    while (last1->next != NULL) {
        last1 = last1->next;
    }
    last1->next = first_2;
    return first_1;
}

NODE sort(NODE first) {
    if (first == NULL || first->next == NULL) {
        return first;
    }

    NODE temp1, temp2;
    int temp_value;
    for (temp1 = first; temp1 != NULL; temp1 = temp1->next) {
        for (temp2 = temp1->next; temp2 != NULL; temp2 = temp2->next) {
            if (temp1->value > temp2->value) {
                // Swap values
                temp_value = temp1->value;
                temp1->value = temp2->value;
                temp2->value = temp_value;
            }
        }
    }
    return first;
```

```
}

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

    NODE temp = first;
    while (temp != NULL) {
        printf("%d\t", temp->value);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    NODE first_1 = NULL;
    NODE first_2 = NULL;
    int choice, item, pos;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert in linked list 1\n");
        printf("2. Insert in linked list 2\n");
        printf("3. Sort in linked list 1\n");
        printf("4. Sort in linked list 2\n");
        printf("5. Reverse in linked list 1\n");
        printf("6. Reverse in linked list 2\n");
        printf("7. Concatenate the two lists\n");
        printf("8. Display LL 1\n");
        printf("9. Display LL 2\n");
        printf("10. Exiting...\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                // Insert in linked list 1
                printf("Enter value to insert: ");
                scanf("%d", &item);
                first_1 = insert_end(item, first_1);
                break;
            case 2:
                printf("Enter value to insert: ");
                scanf("%d", &item);
                first_2 = insert_end(item, first_2);
                break;
            case 3:
                printf("Sorting LL1\n");
                first_1 = sort(first_1);
                break;
            case 4:
                printf("Sorting LL2\n");
                first_2 = sort(first_2);
                break;
            case 5:
                printf("LL1 being reversed\n");
                first_1 = reverse(first_1);
                break;
            case 6:
                printf("LL2 being reversed\n");
                first_2 = reverse(first_2);
                break;
            case 7:
                first_1 = concatenate(first_1, first_2);
                first_2 = NULL;  // Optionally clear second list
                break;
            case 8:
                printf("Displaying LL1: ");
                display(first_1);
                break;
            case 9:
                printf("Displaying LL2: ");
                display(first_2);
                break;
            case 10: exit(0);
                break;
            default:
                printf("Invalid choice.\n");
        }
    }

    return 0;
}
```

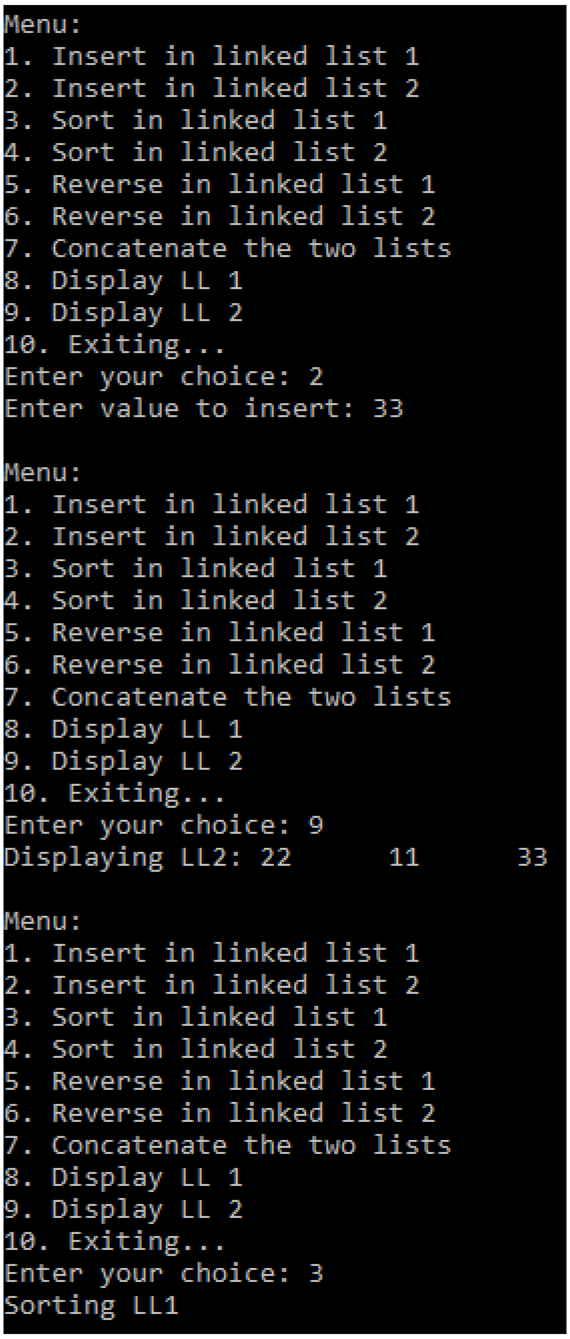
Output:



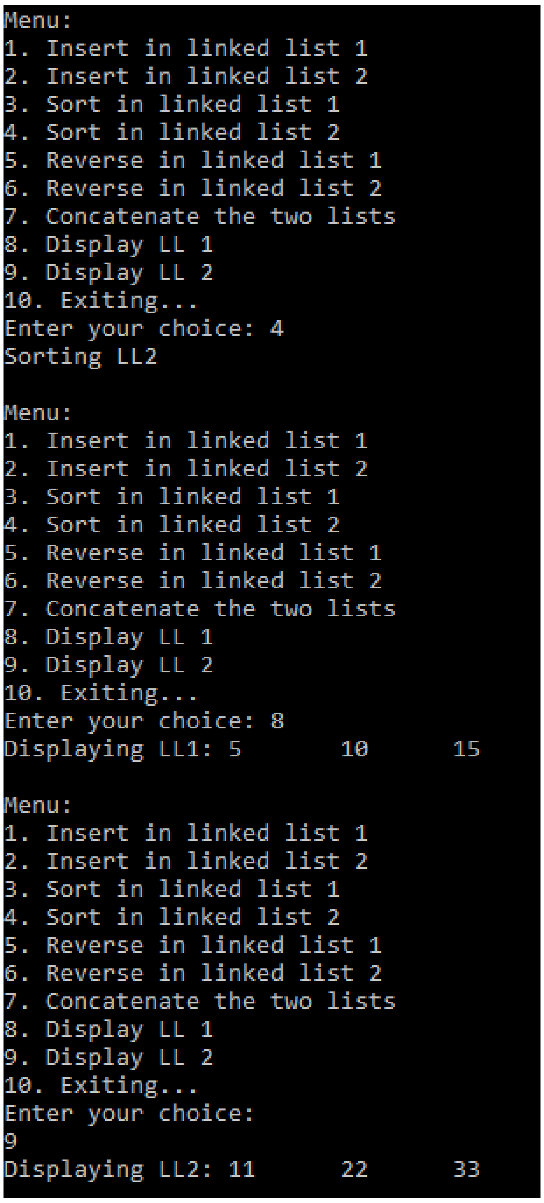
```
Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 1
Enter value to insert: 10

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 1
Enter value to insert: 5

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 1
Enter value to insert: 15
```



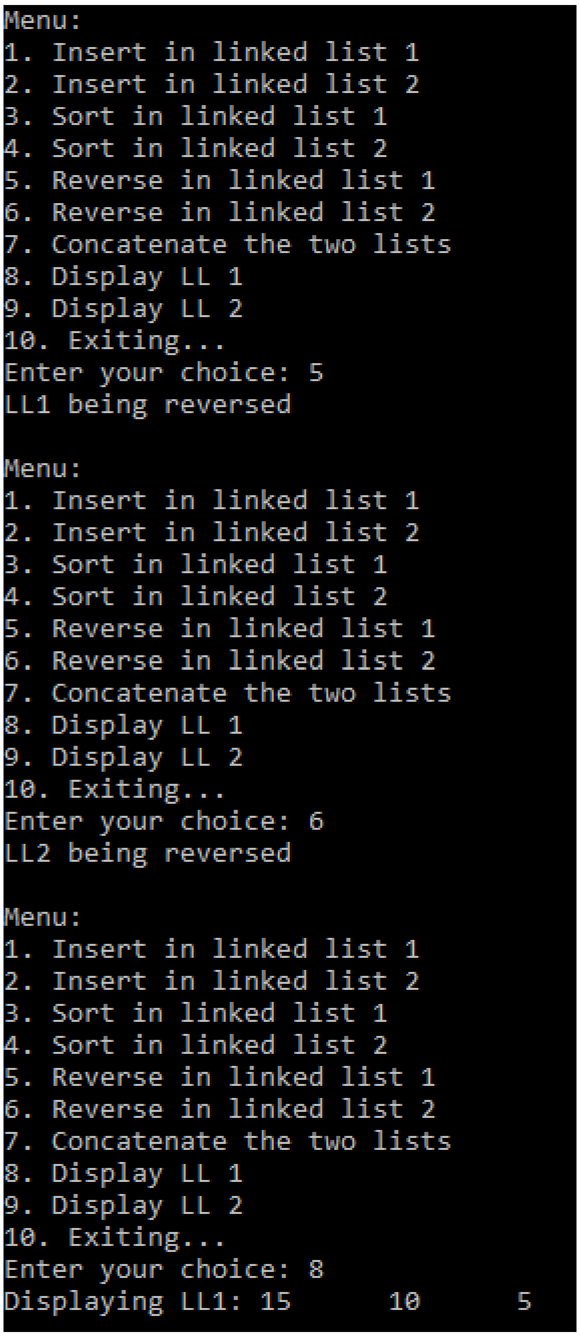
```
Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 8
Displaying LL1: 10      5       15

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 2
Enter value to insert: 22

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 2
Enter value to insert: 11
```



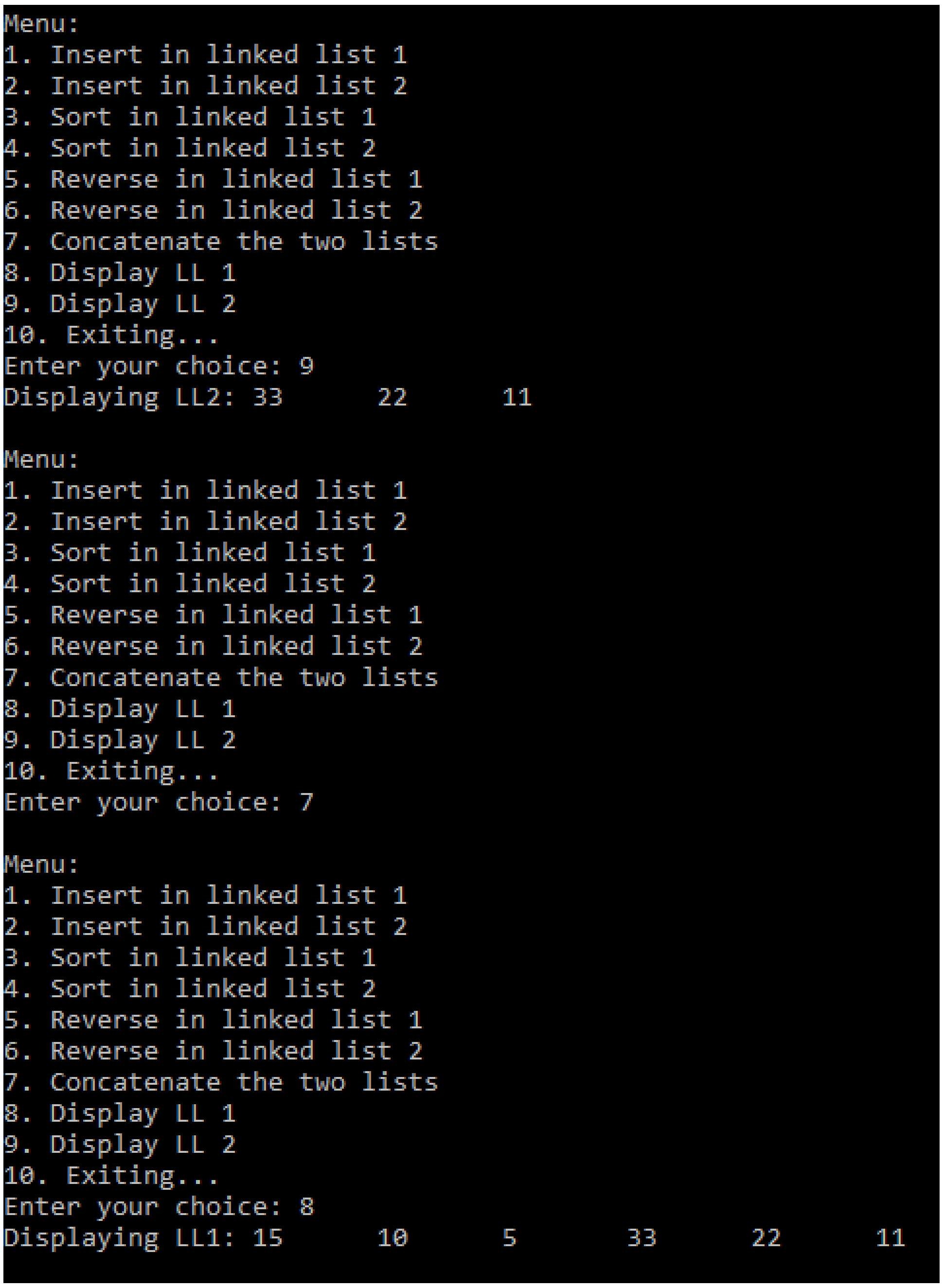
```
Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 2
Enter value to insert: 33

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 9
Displaying LL2: 22      11      33

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 3
Sorting LL1
```

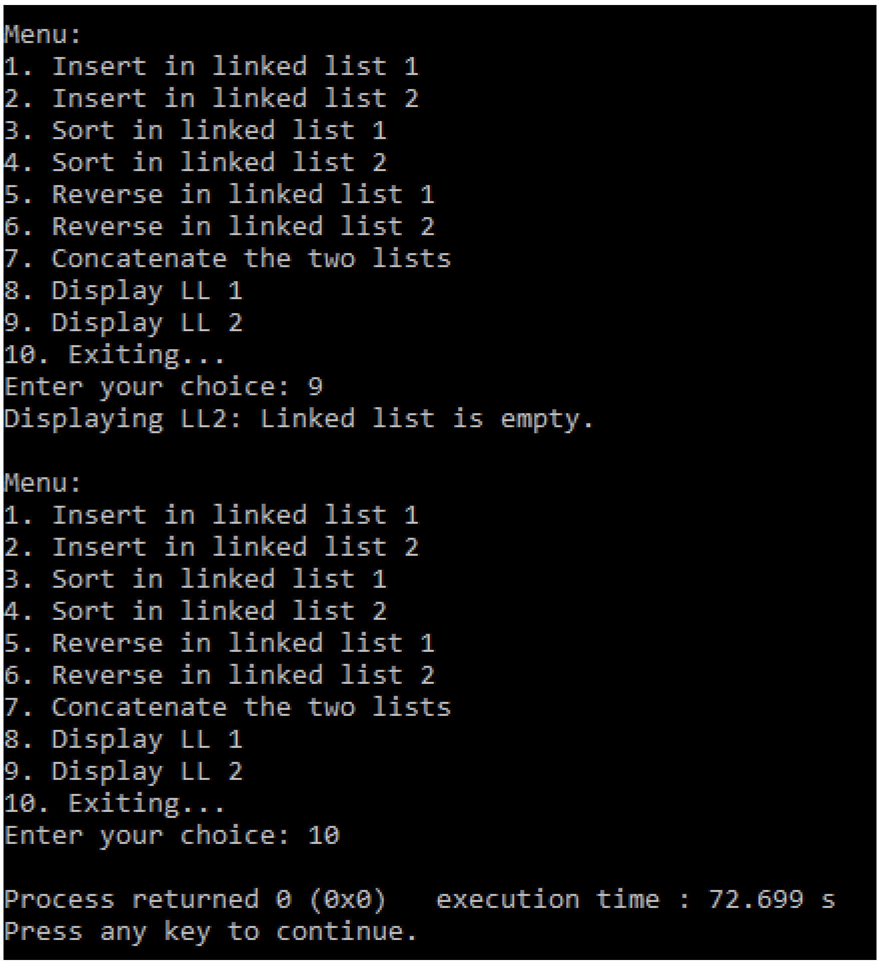


```
Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 4
Sorting LL2

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 8
Displaying LL1: 5       10      15

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice:
9
Displaying LL2: 11      22      33
```



```
Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 5
LL1 being reversed

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 6
LL2 being reversed

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 8
Displaying LL1: 15      10      5
```


```
Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 9
Displaying LL2: 33      22      11

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 7

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 8
Displaying LL1: 15      10      5       33      22      11
```



```
Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 9
Displaying LL2: Linked list is empty.

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
10. Exiting...
Enter your choice: 10

Process returned 0 (0x0)   execution time : 72.699 s
Press any key to continue.
```

Lab program 6:

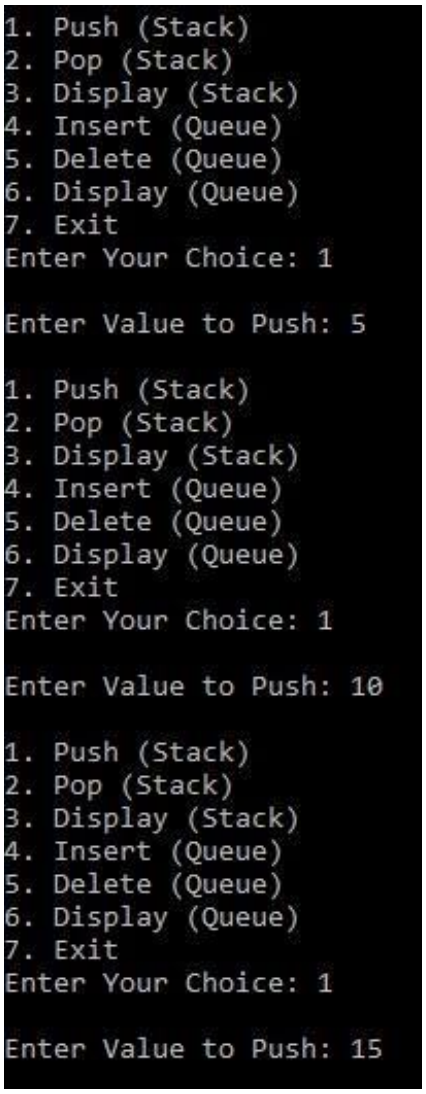
WAP to Implement Single Link List to simulate Stack & Queue Operations.

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

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

typedef struct Node node;

node *top = NULL;
node *temp, *new1 ;       // Top pointer for stack
int max_size = 3;        // Set the maximum stack size
int current_size = 0;    // Variable to track the current stack size

// Stack function prototypes
void push();
void pop();
void displayStack();

// Push function to add an element to the stack
void push() {
    if (current_size == max_size) {
        printf("\nStack is Full. Cannot push more elements.\n");
        return;
    }

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

    printf("\nEnter Value to Push: ");
    scanf("%d", &new1->data);
    new1->link = top;
    top = new1;
    current_size++; // Increment the stack size
}
```

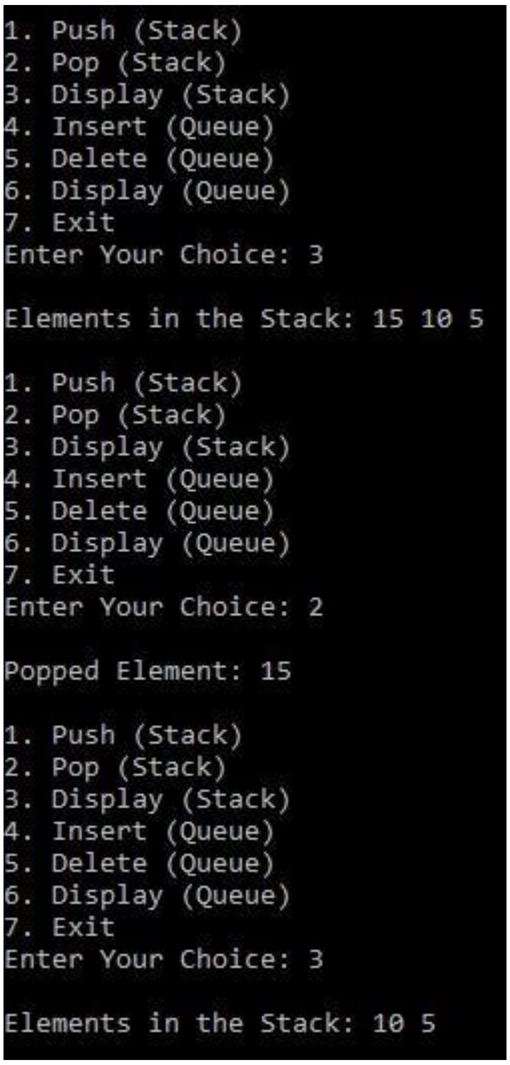
```
// Pop function to remove an element from the stack
void pop() {
    if (top == NULL) {
        printf("\nStack Underflow.\n");
        return;
    }

    temp = top;
    printf("\nPopped Element: %d\n", temp->data);
    top = top->link;
    free(temp);
    current_size--; // Decrement the stack size
}

// Display the stack
void displayStack() {
    if (top == NULL) {
        printf("\nThe Stack is Empty.\n");
        return;
    }

    printf("\nElements in the Stack: ");
    temp = top;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->link;
    }
    printf("\n");
}


node *front = NULL, *rear = NULL;
node *new1, *temp;

// Queue function prototypes
void insert();
void del();
void displayQueue();

// Insert function to add an element to the queue
void insert() {
    node *new1 = (node *)malloc(sizeof(node));
    if (new1 == NULL) {
        printf("\nMemory allocation failed\n");
        return;
    }

    printf("\nEnter Value to Insert: ");
    scanf("%d", &new1->data);
    new1->link = NULL;

    if (rear == NULL) {
        front = rear = new1;
        return;
    }

    rear->link = new1;
    rear = new1;
}

// Delete function to remove an element from the queue
void del() {
    if (front == NULL) {
        printf("\nQueue is Empty.\n");
        return;
    }

    temp = front;
    printf("\nDeleted Element: %d\n", temp->data);
    front = front->link;
    free(temp);
}

// Display the queue
void displayQueue() {
    if (front == NULL) {
        printf("\nThe Queue is Empty.\n");
        return;
    }

    printf("\nElements in the Queue: ");
    temp = front;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->link;
    }
    printf("\n");
}
```

```
// Main function
int main() {
    int ch;

    while (1) {
        printf("\n1. Push (Stack) \n2. Pop (Stack) \n3. Display (Stack)");
        printf("\n4. Insert (Queue) \n5. Delete (Queue) \n6. Display (Queue) \n7. Exit");
        printf("\nEnter Your Choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                insert();
                break;
            case 5:
                del();
                break;
            case 6:
                displayQueue();
                break;
            case 7:
                exit(0);
            default:
                printf("\nEnter a valid choice.\n");
        }
    }
}
```

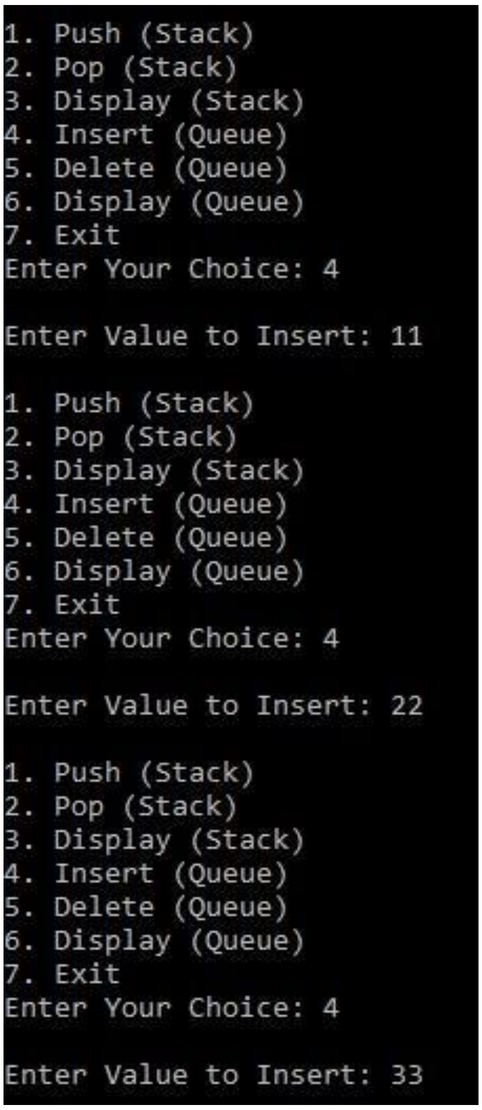
Output:



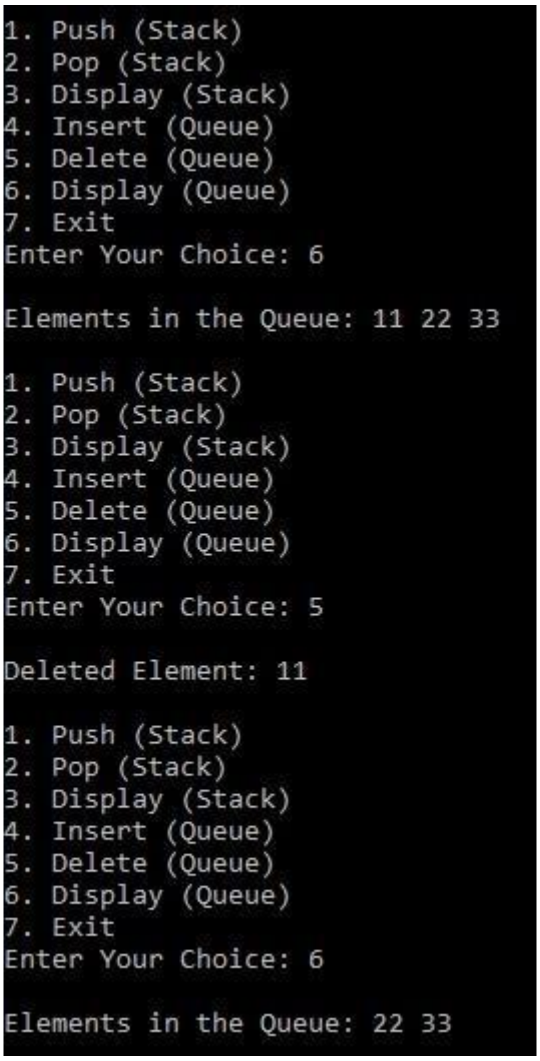
```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1

Enter Value to Push: 5

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1

Enter Value to Push: 10

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1

Enter Value to Push: 15
```



```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 3

Elements in the Stack: 15 10 5

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 2

Popped Element: 15

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 3

Elements in the Stack: 10 5
```



```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4

Enter Value to Insert: 11

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4

Enter Value to Insert: 22

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4

Enter Value to Insert: 33
```



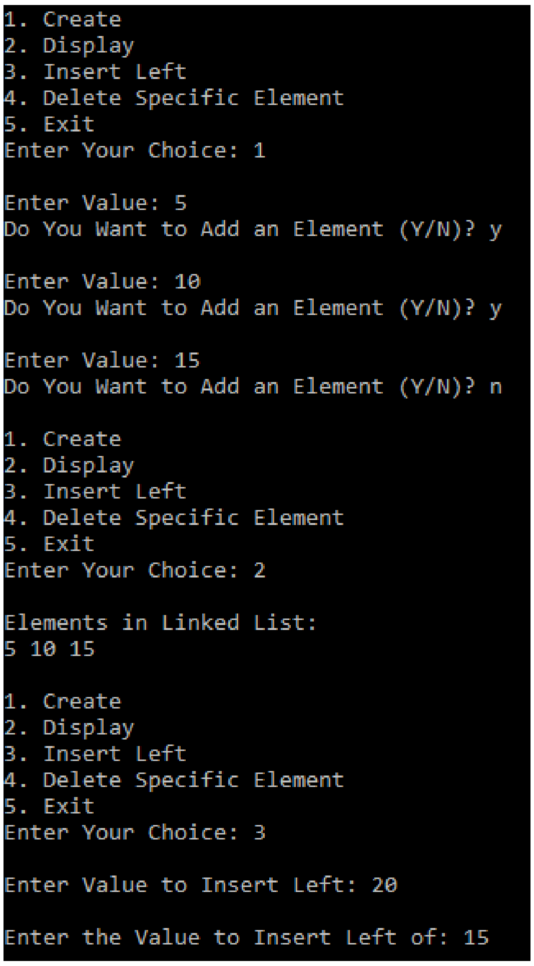
```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 6

Elements in the Queue: 11 22 33

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 5

Deleted Element: 11

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 6

Elements in the Queue: 22 33
```

Lab program 7:

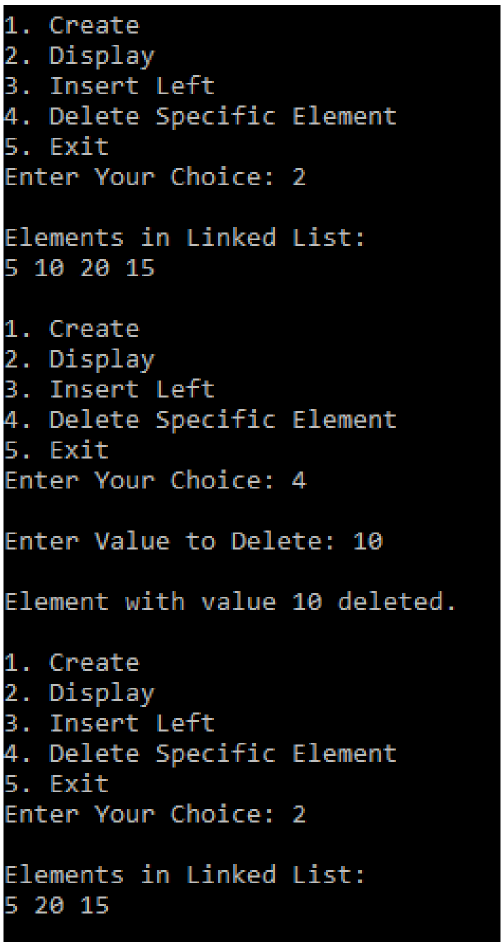
WAP to implement doubly link list with primitive operations
a) Create a doubly linked list.
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value
d) Display the contents of the list

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

struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

typedef struct Node node;
node *start = NULL;
node *new1, *curr, *ptr;

void create();
void display();
void InsertLeft();
void DeleteSpecificElement();

void main() {
    int ch;
    while (1) {
        printf("\n1. Create \n2. Display \n3. Insert Left \n4. Delete Specific Element \n5. Exit");
        printf("\nEnter Your Choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1: create();
                break;
            case 2: display();
                break;
            case 3: InsertLeft();
                break;
            case 4: DeleteSpecificElement();
                break;
            case 5: exit(0);
        }
    }
}

void create() {
    char ch;

    do {
        new1 = (node*)malloc(sizeof(node));
        printf("\nEnter Value: ");
        scanf("%d", &new1->data);
        new1->left = NULL;
        new1->right = NULL;

        if (start == NULL) {
            start = new1;
            curr = new1;
        } else {
            curr->right = new1;
            new1->left = curr;
            curr = new1;
        }

        printf("Do You Want to Add an Element (Y/N)? ");
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
}

void display() {
    if (start == NULL) {
        printf("\nLinked List is Empty.");
        return;
    }

    ptr = start;
    printf("\nElements in Linked List: \n");

    while (ptr != NULL) {
        printf("%d ", ptr->data);
        ptr = ptr->right;
    }
    printf("\n");
}

void InsertLeft() {
    int val;
    printf("\nEnter Value to Insert Left: ");
    scanf("%d", &val);

    new1 = (node*)malloc(sizeof(node));
    new1->data = val;
    new1->left = NULL;
    new1->right = NULL;

    printf("\nEnter the Value to Insert Left of: ");
    scanf("%d", &val);

    ptr = start;
    while (ptr != NULL && ptr->data != val) {
        ptr = ptr->right;
    }

    if (ptr != NULL) {
        new1->right = ptr;
        new1->left = ptr->left;
        if (ptr->left != NULL) {
            ptr->left->right = new1;
        }
        ptr->left = new1;
        if (ptr == start) {
            start = new1;
        }
    } else {
        printf("\nValue not found in the list.\n");
    }
}

void DeleteSpecificElement() {
    int value;
    printf("\nEnter Value to Delete: ");
    scanf("%d", &value);

    ptr = start;
    while (ptr != NULL && ptr->data != value) {
        ptr = ptr->right;
    }

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

    if (ptr->left != NULL) {
        ptr->left->right = ptr->right;
    }
    if (ptr->right != NULL) {
        ptr->right->left = ptr->left;
    }
    if (ptr == start) {
        start = ptr->right;
    }

    free(ptr);
    printf("\nElement with value %d deleted.\n", value);
}
```

Output:



```
1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 1

Enter Value: 5
Do You Want to Add an Element (Y/N)? y

Enter Value: 10
Do You Want to Add an Element (Y/N)? y

Enter Value: 15
Do You Want to Add an Element (Y/N)? n

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 2

Elements in Linked List:
5 10 15

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 3

Enter Value to Insert Left: 20

Enter the Value to Insert Left of: 15
```



```
1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 2

Elements in Linked List:
5 10 20 15

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 4

Enter Value to Delete: 10

Element with value 10 deleted.

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 2

Elements in Linked List:
5 20 15
```

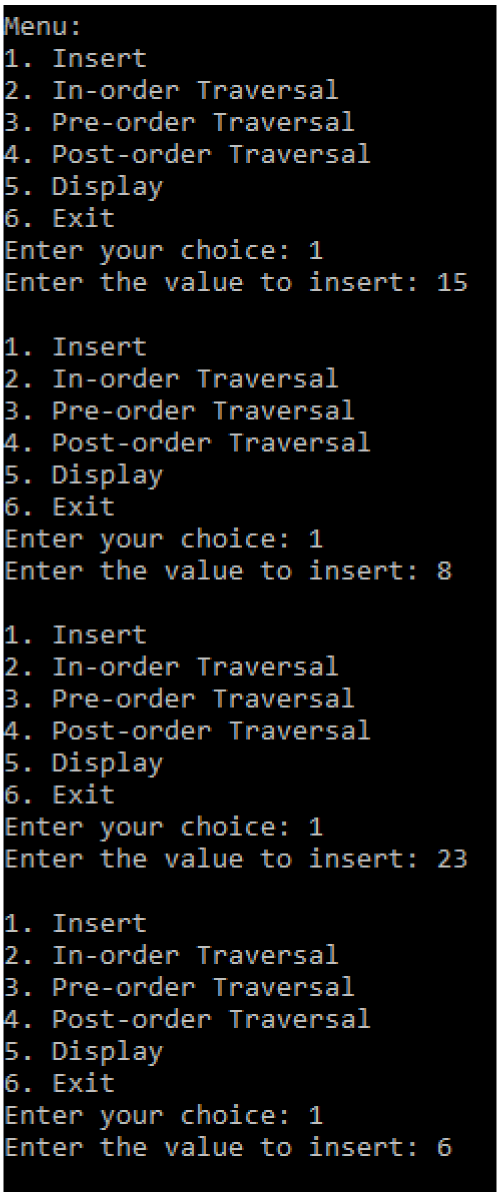
Lab program 8:

Write a program
a) To construct a binary Search tree.
b) To traverse the tree using all the methods i.e., in-order,
pre order and post order
c) To display the elements in the tree.

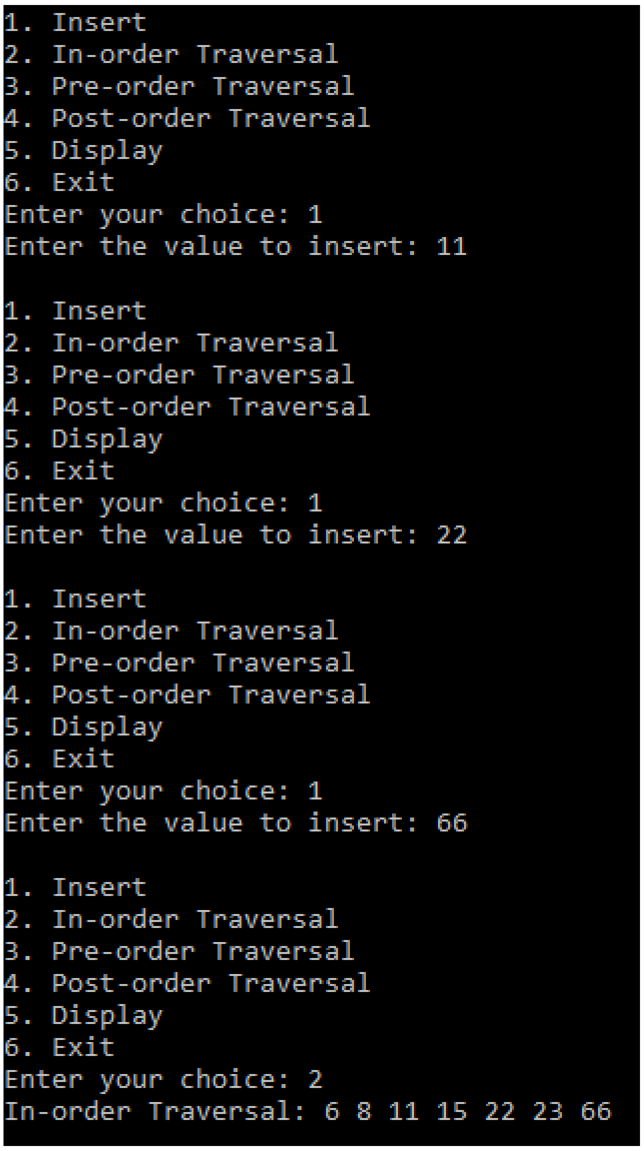
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// Define the structure for the node
typedef struct Node {
   int data;
   struct Node *left, *right;
} node;

// Function to create a new node
node* createNode(int data) {
   node* new1 = (node*)malloc(sizeof(node));
   new1->data = data;
   new1->left = new1->right = NULL;
   return new1;
}

// Function to insert a new node in the BST
node* insertNode(node* root, int data) {
   if (root == NULL) {
      return createNode(data);
   }
   if (data < root->data) {
      root->left = insertNode(root->left, data);
   } else {
      root->right = insertNode(root->right, data);
   }
   return root;
}

void inorderTraversal(node* root) {
   if (root != NULL) {
      inorderTraversal(root->left);
      printf("%d ", root->data);
      inorderTraversal(root->right);
```

```
    }
}

void preorderTraversal(node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

void postorderTraversal(node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}

// Function to find the height of the tree (for indentation calculation)
int findHeight(node* root) {
    if (root == NULL) {
        return 0;
    }
    int leftHeight = findHeight(root->left);
    int rightHeight = findHeight(root->right);
    return fmax(leftHeight, rightHeight) + 1;
}

// Function to print the tree in a visual format with spaces
void displayTree(node* root) {
    // Find the height of the tree
    int height = findHeight(root);
    // Number of spaces between nodes at the last level
    int space = (int)pow(2, height) - 1;

    // Use an array to print the tree level by level
    node* queue[100]; // Queue for level order traversal
    int front = 0, rear = 0;
    queue[rear++] = root;

    while (front < rear) {
        int levelSize = rear - front; // Number of nodes at current level
        // Print spaces between nodes
        for (int i = 0; i < levelSize; i++) {
            // For each level, print appropriate spaces between nodes
            for (int j = 0; j < space; j++) {
                printf(" ");
            }

            node* currentNode = queue[front++];
            if (currentNode != NULL) {
                printf("%d", currentNode->data);

                // Enqueue the left and right children
                if (currentNode->left != NULL) {
                    queue[rear++] = currentNode->left;
                }
                if (currentNode->right != NULL) {
                    queue[rear++] = currentNode->right;
                }
            }
            // Print space between nodes at same level
            for (int j = 0; j < space * 2; j++) {
                printf(" ");
            }
        }
        printf("\n\n");
        space /= 2; // Reduce space for the next level
    }
}

// Main function
int main() {
    node* root = NULL;
    int choice, value;
    printf("\nMenu:");
    while (1) {
        printf("\n1. Insert\n2. In-order Traversal\n3. Pre-order Traversal\n4. Post-order
Traversal\n5. Display\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                root = insertNode(root, value);
                break;
            case 2:
                printf("In-order Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Pre-order Traversal: ");
                preorderTraversal(root);
                printf("\n");
                break;
            case 4:
                printf("Post-order Traversal: ");
                postorderTraversal(root);
                printf("\n");
                break;
            case 5:
                printf("Visual Tree Representation:\n");
                displayTree(root);
                break;
            case 6:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

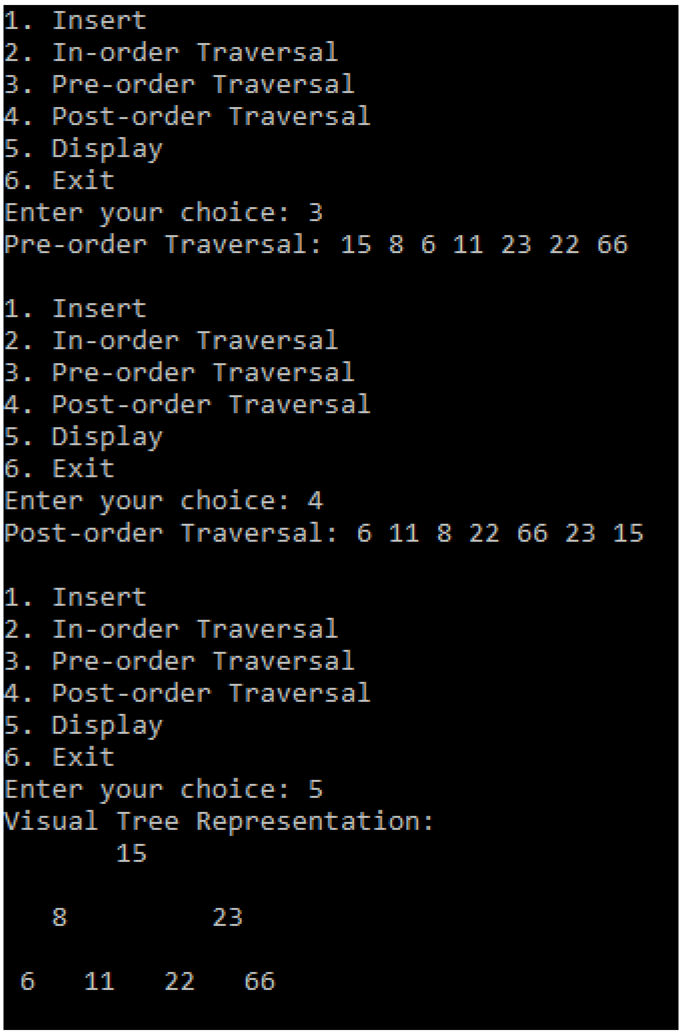
Output:



```
Menu:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 15

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 8

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 23

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 6
```



```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 11

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 22

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 66

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 2
In-order Traversal: 6 8 11 15 22 23 66
```

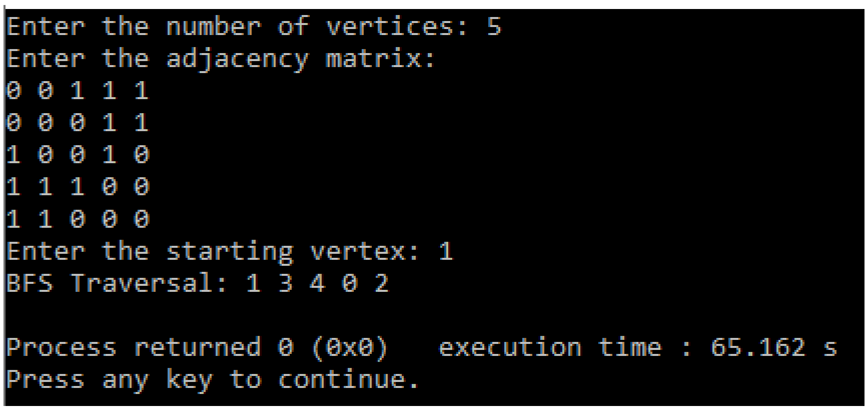

```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 3
Pre-order Traversal: 15 8 6 11 23 22 66

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 4
Post-order Traversal: 6 11 8 22 66 23 15

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 5
Visual Tree Representation:
       15

   8         23

 6   11   22   66
```

Lab program 9:

Write a program to traverse a graph using Breadth First Search (BFS) method.

```
#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
int 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);

    return 0;
}
```

Output:



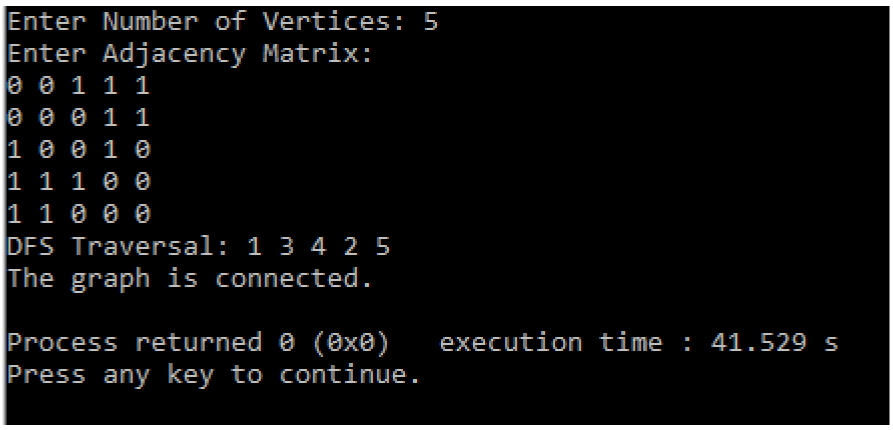
```
Enter the number of vertices: 5
Enter the adjacency matrix:
0 0 1 1 1
0 0 0 1 1
1 0 0 1 0
1 1 1 0 0
1 1 0 0 0
Enter the starting vertex: 1
BFS Traversal: 1 3 4 0 2

Process returned 0 (0x0)   execution time : 65.162 s
Press any key to continue.
```

Lab program 9:

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

```
#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");
    }
}
```

Output:



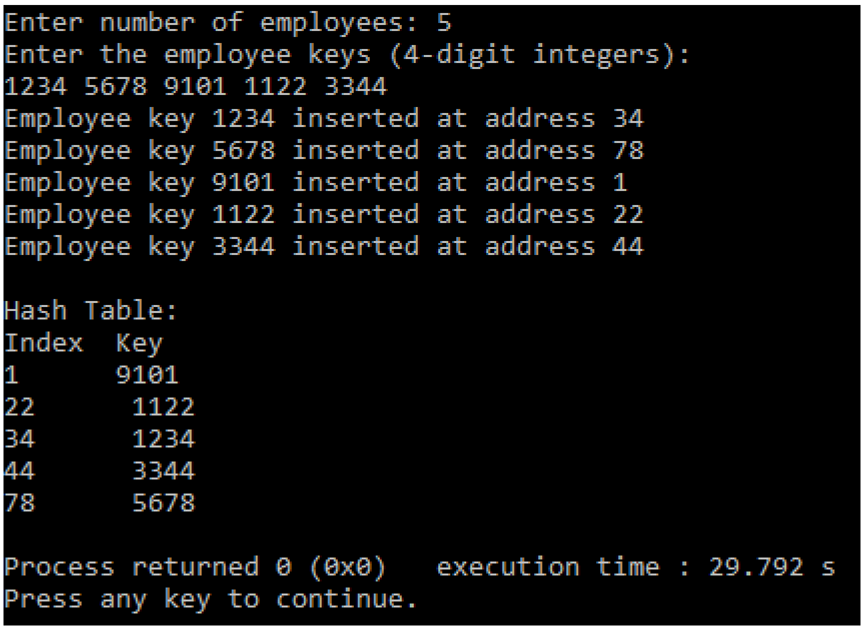
```
Enter Number of Vertices: 5
Enter Adjacency Matrix:
0 0 1 1 1
0 0 0 1 1
1 0 0 1 0
1 1 1 0 0
1 1 0 0 0
DFS Traversal: 1 3 4 2 5
The graph is connected.

Process returned 0 (0x0)   execution time : 41.529 s
Press any key to continue.
```

Lab program 10:

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

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

#define MAX_EMPLOYEES 100

// Define the Hash Table size
#define m 100  // Number of memory locations in the hash table (L)

// Define the structure for an Employee Record
typedef struct {
    int key;  // The unique key for the employee (4-digit integer)
    int address;  // The address mapped in the hash table (2-digit integer)
} EmployeeRecord;

// Define the hash table
int hashTable[m];

// Hash function: H(K) = K mod m
int hashFunction(int key) {
    return key % m;
}

// Function to insert the key into the hash table using linear probing
int insert(int key) {
    int index = hashFunction(key);  // Compute hash index

    // If the location is empty, place the key there
    while (hashTable[index] != -1) {
        // If the location is already occupied, move to the next slot (linear probing)
        index = (index + 1) % m;
    }

    // Insert the key at the found index
    hashTable[index] = key;
    return index;
}

// Function to display the hash table
void displayHashTable() {
    printf("\nHash Table:\n");
    printf("Index  Key\n");
    for (int i = 0; i < m; i++) {
        if (hashTable[i] != -1) {
            printf("%d      %d\n", i, hashTable[i]);
        }
    }
}

int main() {
    // Initialize the hash table with -1 to indicate empty slots
    for (int i = 0; i < m; i++) {
        hashTable[i] = -1;
    }

    // Sample input: employee keys (4-digit integers)
    int employeeKeys[MAX_EMPLOYEES];
    int numEmployees;

    // Input number of employees
    printf("Enter number of employees: ");
    scanf("%d", &numEmployees);

    printf("Enter the employee keys (4-digit integers):\n");
    for (int i = 0; i < numEmployees; i++) {
        scanf("%d", &employeeKeys[i]);
    }

    // Insert each employee key into the hash table
    for (int i = 0; i < numEmployees; i++) {
        int address = insert(employeeKeys[i]);
        printf("Employee key %d inserted at address %d\n", employeeKeys[i], address);
    }

    // Display the final state of the hash table
    displayHashTable();

    return 0;
}
```

Output:



```
Enter number of employees: 5
Enter the employee keys (4-digit integers):
1234 5678 9101 1122 3344
Employee key 1234 inserted at address 34
Employee key 5678 inserted at address 78
Employee key 9101 inserted at address 1
Employee key 1122 inserted at address 22
Employee key 3344 inserted at address 44

Hash Table:
Index  Key
1      9101
22      1122
34      1234
44      3344
78      5678

Process returned 0 (0x0)   execution time : 29.792 s
Press any key to continue.
```