VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Anmol

Bhattarai

(1BM23CS039)

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 **Anmol Bhattarai (1BM23CS039)**, 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. Rajeshwari B S
Associate Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Implementation of stacks	3
2	Infix to Postfix & Leetcode 1	6
3	Implementation of Queues & Leetcode 2	9
4	Circular Queue	13
5	Singly Linked list insertion	16
6	Singly linked list deletion	19
7	Sort Reverse and Concatenate	25
8	Doubly Linked List	29
9	Binary Search Tree	34
10	BFS method	36
11	DFS method	39
12	Linear Probing	41

Course outcomes:

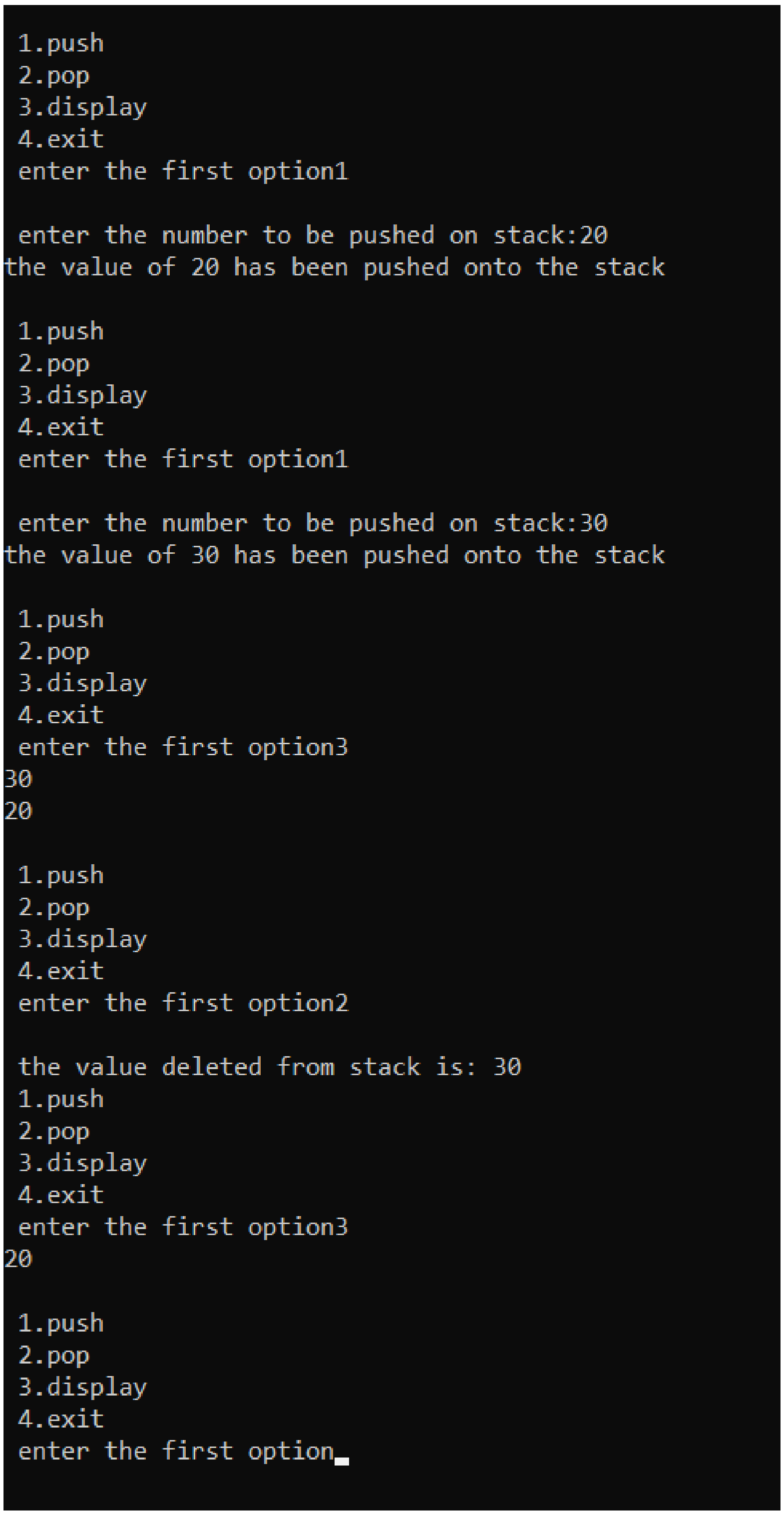
CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations 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 5

int stack[MAX];
int top = -1;
void push(int value)
{
   if(top==MAX-1)
   {
      printf("stack is overflow, cannot push %d
onto the stack\n",value);
   }
   else
   {
      top++;
      stack[top]=value;
      printf("the value of %d has been pushed
onto the stack \n", value);
   }
}
int pop()
{
   if (top==-1)
   {
      printf("stack underflow. Cannot pop from
an empty stack\n");
      return -1;
   }
   else{
      return stack[top--];
   }
}
void display()
{
   if (top==-1)
   {
      printf("stack is empty\n");

   }
   else
   {
      for (int i = top;i>=0;i--){
         printf("%d\n",stack[i]);
      }
```

```
    }
}
int main(){
    int value,op;
    do{
        printf("\n 1.push");
        printf("\n 2.pop");
        printf("\n 3.display");
        printf("\n 4.exit");

        printf("\n enter the first option");
        scanf("%d",&op);
        switch(option)
        {
            case 1:
            printf("\n enter the number to be
pushed on stack:");
            scanf("%d",&value);
            push(value);
            break;

            case 2:
            value = pop();
            if(value!=-1){
            printf("\n the value deleted from stack
is: %d",value);
            }

            case 3:
            display();
            break;

            case 4:
            printf("exiting the program\n");
            break;
        }
    }
    while(op!=4);
    return 0;
}
```

OUTPUT:

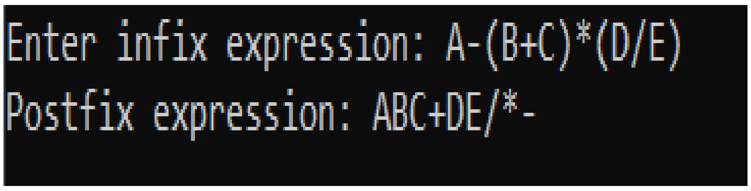


```
 1.push
 2.pop
 3.display
 4.exit
 enter the first option1

 enter the number to be pushed on stack:20
the value of 20 has been pushed onto the stack

 1.push
 2.pop
 3.display
 4.exit
 enter the first option1

 enter the number to be pushed on stack:30
the value of 30 has been pushed onto the stack

 1.push
 2.pop
 3.display
 4.exit
 enter the first option3
30
20

 1.push
 2.pop
 3.display
 4.exit
 enter the first option2

 the value deleted from stack is: 30
 1.push
 2.pop
 3.display
 4.exit
 enter the first option3
20

 1.push
 2.pop
 3.display
 4.exit
 enter the first option
```

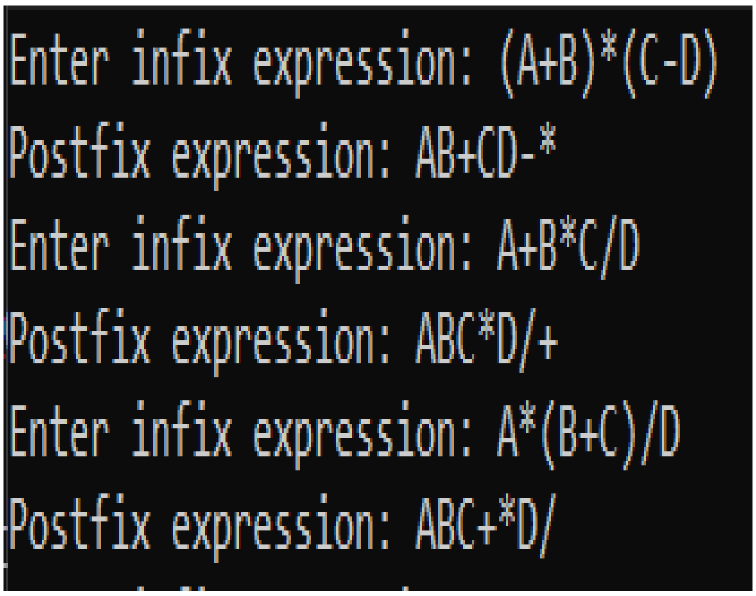
LAB Program 2: INFIX TO POSTFIX

```
#include <stdio.h>
#define MAX 100

void push(char stack[], int *top, char c) {
    if (*top < MAX - 1) {
        stack[++(*top)] = c;
    }
}

char pop(char stack[], int *top) {
    if (*top >= 0) {
        return stack[(*top)--];
    } else {
        return '\0';
    }
}

int precedence(char c) {
    if (c == '+' || c == '-') {
        return 1;
    } else if (c == '*' || c == '/') {
        return 2;
    }
    return 0;
}

int isOperand(char c) {
    return ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z') || (c >= '0' && c <= '9'));
}

void infixtopostfix(char infix[], char postfix[]) {
    char stack[MAX];
    int top = -1;
    int i = 0, j = 0;
    char token;

    while ((token = infix[i]) != '\0') {
        if (isOperand(token)) {
            postfix[j++] = token;
        } else if (token == '(') {
            push(stack, &top, token);
        } else if (token == ')') {
            while (top != -1 && stack[top] != '(') {
                postfix[j++] = pop(stack, &top);
            }
            pop(stack, &top);
        } else {
            while (top != -1 && precedence(stack[top]) >= precedence(token)) {
                postfix[j++] = pop(stack, &top);
            }
            push(stack, &top, token);
        }
        i++;
    }

    while (top != -1) {
        postfix[j++] = pop(stack, &top);
    }
    postfix[j] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX];
    printf("Enter infix expression: ");
    scanf("%s", infix);
    infixtopostfix(infix, postfix);
    printf("Postfix expression: %s\n", postfix);
    return 0;
}
```

OUTPUT:



```
Enter infix expression: A-(B+C)*(D/E)
Postfix expression: ABC+DE/*-
```



```
Enter infix expression: (A+B)*(C-D)
Postfix expression: AB+CD-*
Enter infix expression: A+B*C/D
Postfix expression: ABC*D/+
Enter infix expression: A*(B+C)/D
Postfix expression: ABC+*D/
```

LEETCODE 1: NEXT GREATER INTEGER

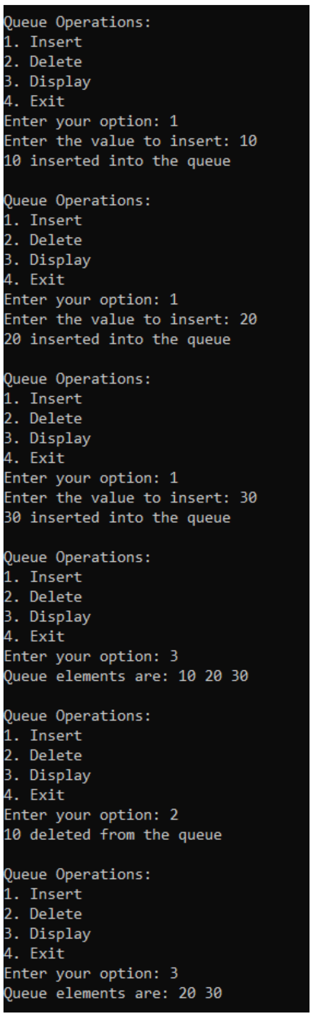
```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* nextGreaterElement(int* nums1, int nums1Size, int* nums2, int nums2Size,
                        int* returnSize)
{
    int* ans=malloc(nums1Size*sizeof(int));
    for (int i = 0; i < nums1Size; i++)
    {
        ans[i]=-1;
        for (int j = 0; j < nums2Size; j++)
        {
            if (nums1[i] == nums2[j])
            {
                for (int k=j+1; k < nums2Size; k++)
                {
                    if (nums2[k] > nums1[i])
                    {
```

```
                    ans[i] = nums2[k];
                    break;
                }
            }
            break;
        }
    }
}

*returnSize = nums1Size;
return ans;
}
```

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 queue overflow conditions

```
#include<stdio.h>
#define MAX 10


void insert(int val, int *rear, int *front, int queue[]) {
    if (*rear == MAX - 1) {
        printf("The queue is full\n");
    } else {
        if (*front == -1 && *rear == -1) {
            *front = *rear = 0;
        } else {
            *rear = *rear + 1;
        }
        queue[*rear] = val;
        printf("%d inserted into the queue\n", val);
    }
}


void delete(int *front, int *rear, int queue[]) {
    if (*front == -1 || *front > *rear) {
        printf("The queue is empty\n");
    } else {
        int val = queue[*front];
        printf("%d deleted from the queue\n", val);
        *front = *front + 1;
        if (*front > *rear) {
            *front = *rear = -1;
        }
    }
}


void display(int *rear, int *front, int queue[]) {
    if (*front == -1) {
        printf("The queue is empty\n");
    } else {
        printf("Queue elements are: ");
        for (int i = *front; i <= *rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}

int main() {
    int option = 0, val;
    int queue[MAX];
    int front = -1, rear = -1;

    while (option != 4) {
        printf("\nQueue Operations:\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your option: ");
        scanf("%d", &option);

        switch (option) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &val);
                insert(val, &rear, &front, queue);
                break;
            case 2:
                delete(&front, &rear, queue);
                break;
            case 3:
                display(&rear, &front, queue);
                break;
            case 4:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid option! Please try again.\n");
        }
    }

    return 0;
}
```

OUTPUT:



```
Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your option: 1
Enter the value to insert: 10
10 inserted into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your option: 1
Enter the value to insert: 20
20 inserted into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your option: 1
Enter the value to insert: 30
30 inserted into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your option: 3
Queue elements are: 10 20 30

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your option: 2
10 deleted from the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your option: 3
Queue elements are: 20 30
```

LEETCODE 2: NUMBER OF RECENT CALLS

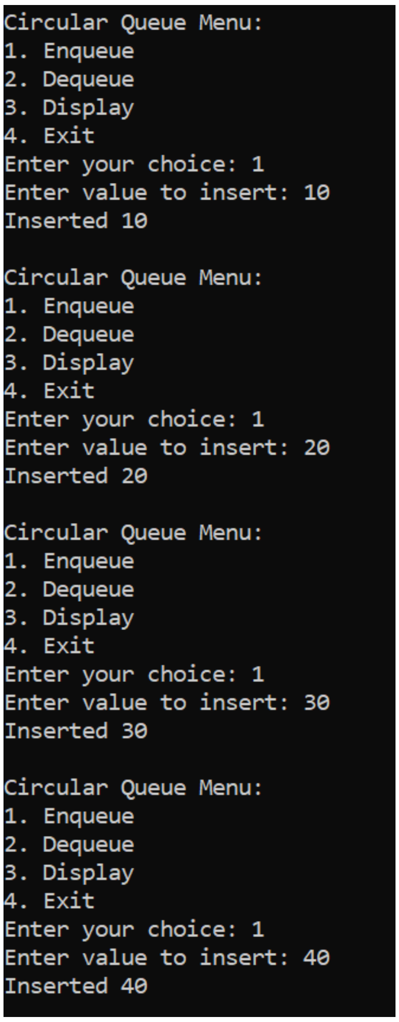
```
typedef struct {
    int callqueue[10000];
    int front;
    int rear;

} RecentCounter;



RecentCounter* recentCounterCreate() {
    RecentCounter* obj = (RecentCounter*)malloc(sizeof(RecentCounter));
    obj->front = 0;
    obj->rear = -1;
    return obj;

}

int recentCounterPing(RecentCounter* obj, int t) {
    obj->callqueue[++obj->rear]=t;

    while (obj->callqueue[obj->front] < t - 3000) {
    obj->front++;
    }

    return obj->rear - obj->front + 1;
}

void recentCounterFree(RecentCounter* obj) {
    free(obj);
}


/**
 * Your RecentCounter struct will be instantiated and called as such:
 * RecentCounter* obj = recentCounterCreate();
 * int param_1 = recentCounterPing(obj, t);
 * recentCounterFree(obj);
 */
```

Lab Program 4:
WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete &amp; Display The program should print appropriate messages for queue empty and queue overflow conditions

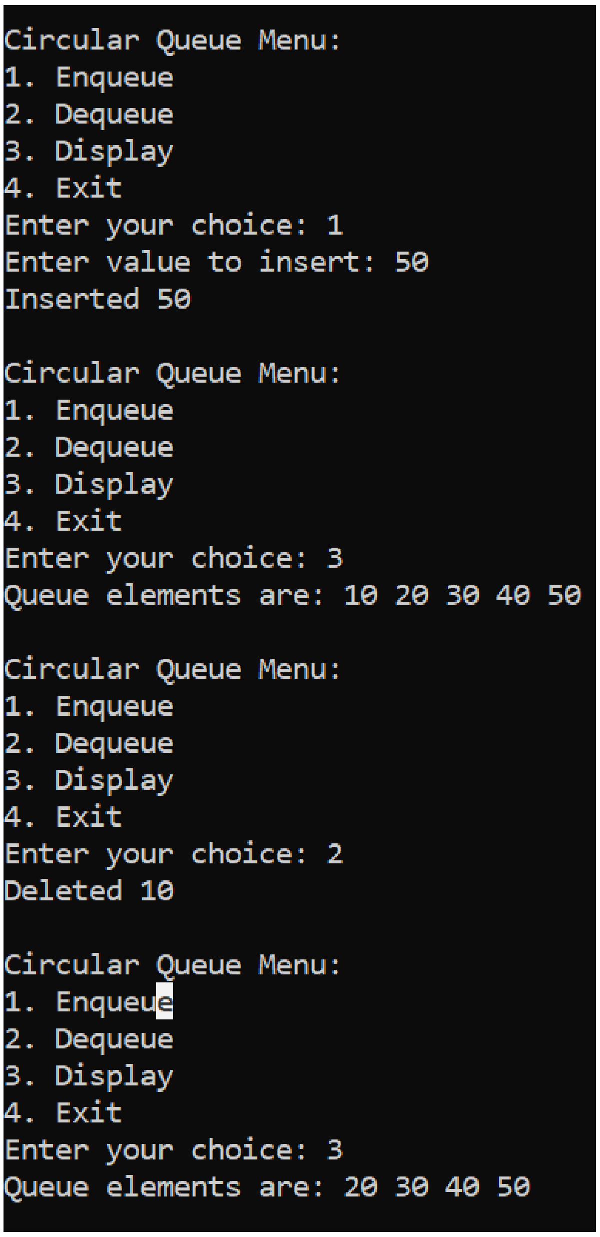
```
#include  <stdio.h>
#include <stdlib.h>
#define MAX 5

int isFull(int front, int rear) {
   return (front == (rear + 1) % MAX);
}

int isEmpty(int front) {
   return front == -1;
}

void enqueue(int queue[], int *front, int *rear, int value) {
   if (isFull(*front, *rear)) {
      printf("Queue is full!\n");
   } else {
      if (*front == -1)
         *front = 0;
      *rear = (*rear + 1) % MAX;
      queue[*rear] = value;
      printf("Inserted %d\n", value);
   }
}

void dequeue(int queue[], int *front, int *rear) {
   if (isEmpty(*front)) {
      printf("Queue is empty!\n");
   } else {
      printf("Deleted %d\n", queue[*front]);
      if (*front == *rear) {
         *front = -1;
         *rear = -1;
      } else {
         *front = (*front + 1) % MAX;
      }
   }
```

```
}

void display(int queue[], int front, int rear) {
    if (isEmpty(front)) {
        printf("Queue is empty!\n");
    } else {
        printf("Queue elements are: ");
        int i = front;
        while (i != rear) {
            printf("%d ", queue[i]);
            i = (i + 1) % MAX;
        }
        printf("%d\n", queue[i]);
    }
}

int main() {
    int queue[MAX];
    int front = -1, rear = -1;
    int choice, value;

    while (1) {
        printf("\nCircular Queue Menu:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                enqueue(queue, &front, &rear, value);
                break;

            case 2:
                dequeue(queue, &front, &rear);
                break;

            case 3:
                display(queue, front, rear);
                break;

            case 4:
                printf("Exiting...\n");
                exit(0);

            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

OUTPUT:



```
Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 10
Inserted 10

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 20
Inserted 20

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 30
Inserted 30

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 40
Inserted 40
```



```
Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 50
Inserted 50

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 10 20 30 40 50

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 10

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 20 30 40 50
```

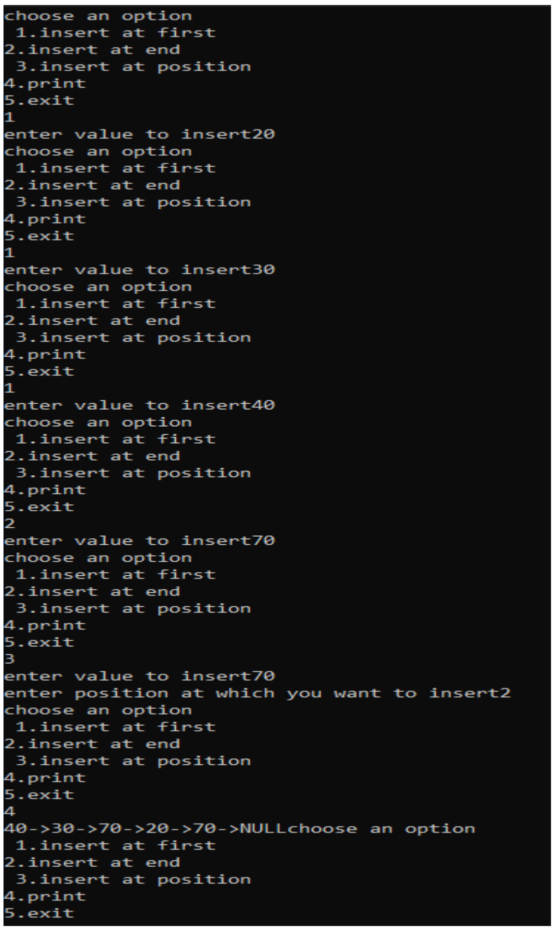
Lab Program 5:
WAP to Implement Singly Linked List with following operations
a) Create a linked list.
b) Insertion of a node at first position, at any position and at end of list.
Display the contents of the linked list.

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

```
};
struct Node* createNode(int data)
{
    struct Node*newNode=(struct Node*)malloc(sizeof(struct Node));
    newNode->data=data;
    newNode->next=NULL;
    return newNode;
}
void insertatfirst(struct Node**head,int data)
{
    struct Node*newNode=createNode(data);
    newNode->next=*head;
    *head=newNode;
}
void insertatend(struct Node**head,int data)
{
    struct Node*newNode=createNode(data);
    if(*head==NULL)
    {
        *head=newNode;
    }
    else
    {
        struct Node* temp=*head;
        while(temp->next!=NULL)
        {
            temp=temp->next;
        }
        temp->next=newNode;
    }
}
void insertAtposition(struct Node**head,int data,int position)
{
    struct Node*newNode=createNode(data);
    if(position==0)
    {
        insertatfirst(head,data);
        return;
    }
     struct Node*temp=*head;
     for(int i=0;temp!=NULL&&i<position-1;i++)
     {
         temp=temp->next;
      }
      newNode->next=temp->next;
      temp->next=newNode;
}
void print(struct Node**head)
{
    struct Node*temp=*head;
    while(temp!=NULL)
    {
        printf("%d->",temp->data);
        temp=temp->next;
    }
    printf("NULL");
}
int main()
{
    struct Node*head=NULL;
    int choice,value,position;
    do
    {
        printf("choice a otion 1.insert at first,2.insert at end,3.insert at position,4.print,5.exit");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:printf("enter value to insert");
                   scanf("%d",&value);
                   insertatfirst(&head,value);
                   break;
            case 2:printf("enter value to insert");
                   scanf("%d",&value);
                   insertatend(&head,value);
                   break;
            case 3:printf("enter value to insert");
                   scanf("%d",&value);
                   printf("enter position at which you want to insert");
                   scanf("%d",&position);
                   insertAtposition(&head,value,position);
                   break;
            case 4:print(&head);
                   break;
            case 5:printf("exiting...");
```

```
                break;
            default:printf("invalid option ,choice again");

        }
    }while(choice!=5);
}
```

OUTPUT:



```
choose an option
 1.insert at first
2.insert at end
 3.insert at position
4.print
5.exit
1
enter value to insert20
choose an option
 1.insert at first
2.insert at end
 3.insert at position
4.print
5.exit
1
enter value to insert30
choose an option
 1.insert at first
2.insert at end
 3.insert at position
4.print
5.exit
1
enter value to insert40
choose an option
 1.insert at first
2.insert at end
 3.insert at position
4.print
5.exit
2
enter value to insert70
choose an option
 1.insert at first
2.insert at end
 3.insert at position
4.print
5.exit
3
enter value to insert70
enter position at which you want to insert2
choose an option
 1.insert at first
2.insert at end
 3.insert at position
4.print
5.exit
4
40->30->70->20->70->NULLchoose an option
 1.insert at first
2.insert at end
 3.insert at position
4.print
5.exit
```

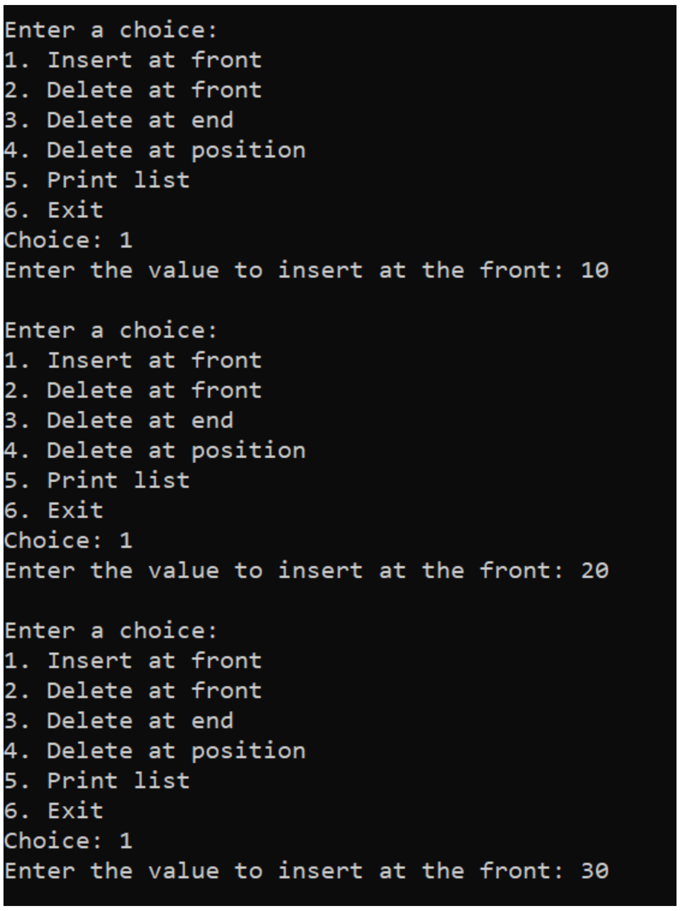
Lab Program 6:
WAP to Implement Singly Linked List with following operations
a) Create a linked list.
b) Deletion of first element, specified element and last element in the list.
c) Display the contents of the linked list.

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

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertatfirst(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

void deleteatfirst(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = *head;
    *head = temp->next;
    free(temp);
}

void deleteatend(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = *head;
    if (temp->next == NULL) {
        free(temp);
        *head = NULL;
        return;
    }
    while (temp->next->next != NULL) {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
```

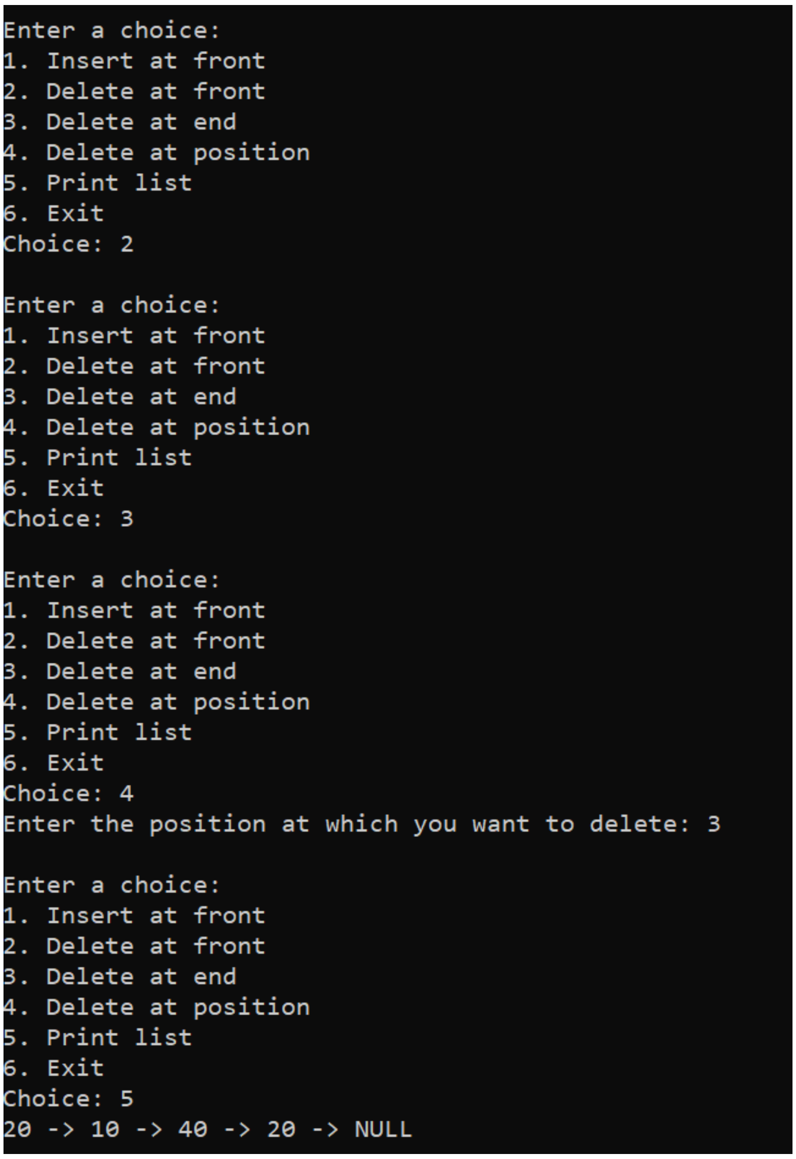
```
}

void deleteatposition(struct Node** head, int position) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = *head;

    if (position == 0) {
        deleteatfirst(head);
        return;
    }

    for (int i = 0; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL) {
        printf("Position out of range\n");
        return;
    }

    struct Node* next = temp->next->next;
    free(temp->next);
    temp->next = next;
}

void print(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
```

```
int choice, position, data;
insertatfirst(&head, 10);
insertatfirst(&head, 20);
insertatfirst(&head, 30);
insertatfirst(&head, 40);
print(head);
do {
    printf("\nEnter a choice:\n");
    printf("1. Insert at front\n");
    printf("2. Delete at front\n");
    printf("3. Delete at end\n");
    printf("4. Delete at position\n");
    printf("5. Print list\n");
    printf("6. Exit\n");
    printf("Choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the value to insert at the front: ");
            scanf("%d", &data);
            insertatfirst(&head, data);
            break;
        case 2:
            deleteatfirst(&head);
            break;
        case 3:
            deleteatend(&head);
            break;
        case 4:
            printf("Enter the position at which you want to delete: ");
            scanf("%d", &position);
            deleteatposition(&head, position);
            break;
        case 5:
            print(head);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Please choose a valid option\n");
        }

    } while (choice != 6);

    return 0;
}
```

OUTPUT:



```
Enter a choice:
1. Insert at front
2. Delete at front
3. Delete at end
4. Delete at position
5. Print list
6. Exit
Choice: 1
Enter the value to insert at the front: 10

Enter a choice:
1. Insert at front
2. Delete at front
3. Delete at end
4. Delete at position
5. Print list
6. Exit
Choice: 1
Enter the value to insert at the front: 20

Enter a choice:
1. Insert at front
2. Delete at front
3. Delete at end
4. Delete at position
5. Print list
6. Exit
Choice: 1
Enter the value to insert at the front: 30
```



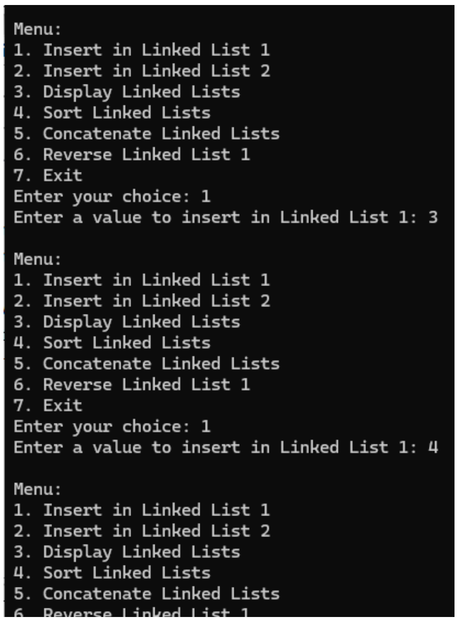
```
Enter a choice:
1. Insert at front
2. Delete at front
3. Delete at end
4. Delete at position
5. Print list
6. Exit
Choice: 2

Enter a choice:
1. Insert at front
2. Delete at front
3. Delete at end
4. Delete at position
5. Print list
6. Exit
Choice: 3

Enter a choice:
1. Insert at front
2. Delete at front
3. Delete at end
4. Delete at position
5. Print list
6. Exit
Choice: 4
Enter the position at which you want to delete: 3

Enter a choice:
1. Insert at front
2. Delete at front
3. Delete at end
4. Delete at position
5. Print list
6. Exit
Choice: 5
20 -> 10 -> 40 -> 20 -> NULL
```

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

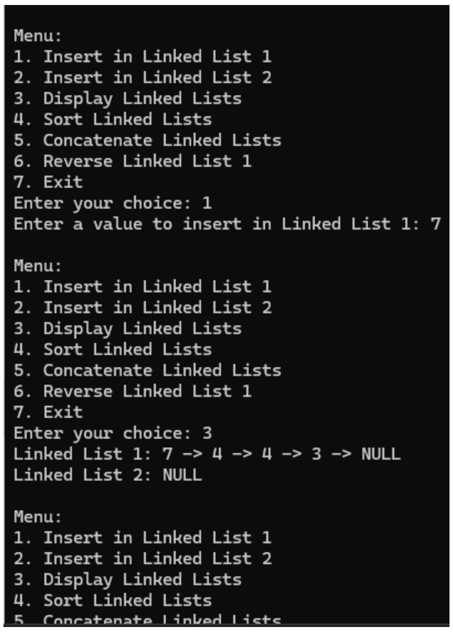
```
#include <stdio.h>
#include <stdlib.h>

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

Node *createNode(int value) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

Node *insertAtBeginning(Node *head, int value) {
    Node *newNode = createNode(value);
    newNode->next = head;
    return newNode;
}

Node *concat(Node *head1, Node *head2) {
    Node *temp = head1;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = head2;
    return head1;
}

Node *sort(Node *head) {
    Node *temp, *current;
    int t;
    current = head;
    while (current != NULL) {
        temp = head;
        while (temp->next != NULL) {
            if (temp->data > temp->next->data) {
                t = temp->data;
                temp->data = temp->next->data;
                temp->next->data = t;
            }
            temp = temp->next;
        }
        current = current->next;
    }
    return head;
}

Node *reverse(Node *head) {
    Node *prev = NULL, *temp, *next;
    temp = head;
    while (temp != NULL) {
        next = temp->next;
        temp->next = prev;
        prev = temp;
        temp = next;
    }
    head = prev;
    return head;
}

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

int main() {
    Node *list1 = NULL;
    Node *list2 = NULL;
    int choice, value;

    do {
        printf("\nMenu:\n");
        printf("1. Insert in Linked List 1\n");
        printf("2. Insert in Linked List 2\n");
        printf("3. Display Linked Lists\n");
        printf("4. Sort Linked Lists\n");
        printf("5. Concatenate Linked Lists\n");
        printf("6. Reverse Linked List 1\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
           case 1:
              printf("Enter a value to insert in Linked List 1: ");
              scanf("%d", &value);
              list1 = insertAtBeginning(list1, value);
              break;

           case 2:
              printf("Enter a value to insert in Linked List 2: ");
              scanf("%d", &value);
              list2 = insertAtBeginning(list2, value);
              break;

           case 3:
              printf("Linked List 1: ");
              displayLinkedList(list1);
              printf("Linked List 2: ");
              displayLinkedList(list2);
              break;

           case 4:
              printf("Sorting Linked Lists...\n");
              list1 = sort(list1);
              list2 = sort(list2);
              printf("Sorted Linked List 1: ");
              displayLinkedList(list1);
              printf("Sorted Linked List 2: ");
              displayLinkedList(list2);
              break;

           case 5:
              printf("Concatenating Linked List 2 to Linked List 1...\n");
              list1 = concat(list1, list2);
```

```
                printf("Concatenated Linked List: ");
                displayLinkedList(list1);
                list2 = NULL;  // Clear list2 after concatenation
                break;

            case 6:
                printf("Reversing Linked List 1...\n");
                list1 = reverse(list1);
                printf("Reversed Linked List 1: ");
                displayLinkedList(list1);
                break;

            case 7:
                printf("Exiting program.\n");
                break;

            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 7);

    return 0;
}
```

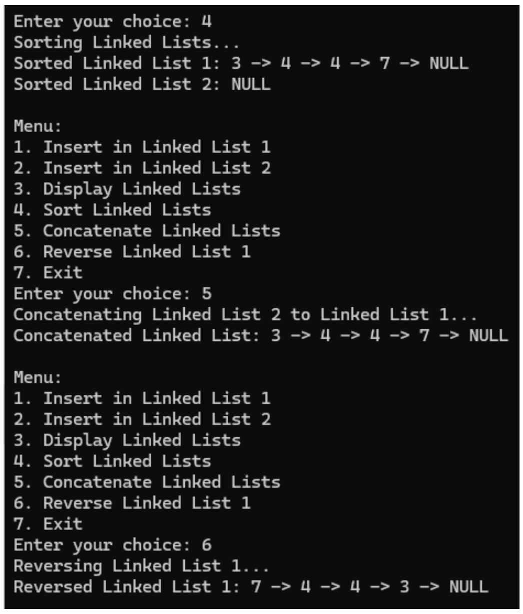
OUTPUT:



```
Menu:
1. Insert in Linked List 1
2. Insert in Linked List 2
3. Display Linked Lists
4. Sort Linked Lists
5. Concatenate Linked Lists
6. Reverse Linked List 1
7. Exit
Enter your choice: 1
Enter a value to insert in Linked List 1: 3

Menu:
1. Insert in Linked List 1
2. Insert in Linked List 2
3. Display Linked Lists
4. Sort Linked Lists
5. Concatenate Linked Lists
6. Reverse Linked List 1
7. Exit
Enter your choice: 1
Enter a value to insert in Linked List 1: 4

Menu:
1. Insert in Linked List 1
2. Insert in Linked List 2
3. Display Linked Lists
4. Sort Linked Lists
5. Concatenate Linked Lists
6. Reverse Linked List 1
```



```
Menu:
1. Insert in Linked List 1
2. Insert in Linked List 2
3. Display Linked Lists
4. Sort Linked Lists
5. Concatenate Linked Lists
6. Reverse Linked List 1
7. Exit
Enter your choice: 1
Enter a value to insert in Linked List 1: 7

Menu:
1. Insert in Linked List 1
2. Insert in Linked List 2
3. Display Linked Lists
4. Sort Linked Lists
5. Concatenate Linked Lists
6. Reverse Linked List 1
7. Exit
Enter your choice: 3
Linked List 1: 7 -> 4 -> 4 -> 3 -> NULL
Linked List 2: NULL

Menu:
1. Insert in Linked List 1
2. Insert in Linked List 2
3. Display Linked Lists
4. Sort Linked Lists
5. Concatenate Linked Lists
```



```
Enter your choice: 4
Sorting Linked Lists...
Sorted Linked List 1: 3 -> 4 -> 4 -> 7 -> NULL
Sorted Linked List 2: NULL

Menu:
1. Insert in Linked List 1
2. Insert in Linked List 2
3. Display Linked Lists
4. Sort Linked Lists
5. Concatenate Linked Lists
6. Reverse Linked List 1
7. Exit
Enter your choice: 5
Concatenating Linked List 2 to Linked List 1...
Concatenated Linked List: 3 -> 4 -> 4 -> 7 -> NULL

Menu:
1. Insert in Linked List 1
2. Insert in Linked List 2
3. Display Linked Lists
4. Sort Linked Lists
5. Concatenate Linked Lists
6. Reverse Linked List 1
7. Exit
Enter your choice: 6
Reversing Linked List 1...
Reversed Linked List 1: 7 -> 4 -> 4 -> 3 -> NULL
```

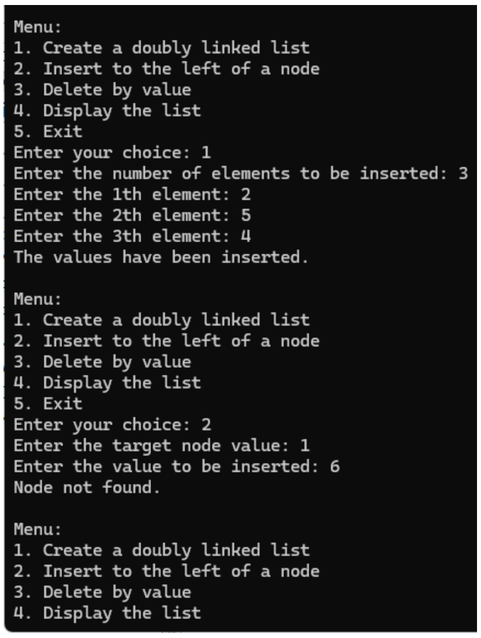
Lab Program 8: WAP to Implement doubly link list with primitive operations
a) Create a doubly linked list.
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value
Display the contents of the list

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

struct Node
```

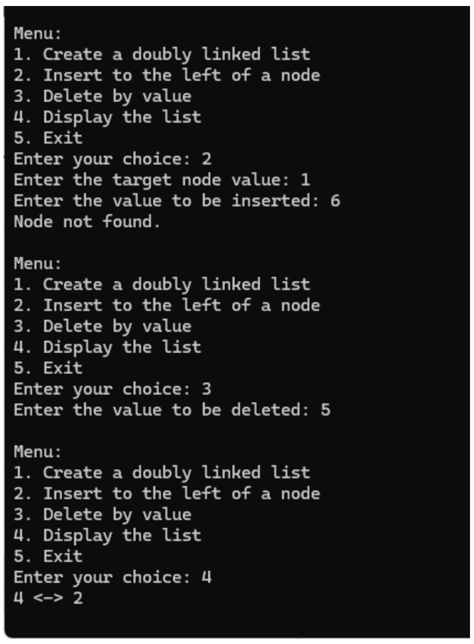
```
{
    int data;
    struct Node* next;
    struct Node* prev;
};

struct Node* insert(struct Node* head, int data)
{
    struct Node* newnode = (struct Node*)malloc(sizeof(struct Node));
    newnode->data = data;
    newnode->next = head;
    newnode->prev = NULL;
    if (head != NULL)
        head->prev = newnode;
    return newnode;
}

struct Node* create(struct Node* head)
{
    int n;
    printf("Enter the number of elements to be inserted: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        int value;
        printf("Enter the %dth element: ", i + 1);
        scanf("%d", &value);
        head = insert(head, value);
    }
    printf("The values have been inserted.\n");
    return head;
}

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

struct Node* insertToLeft(struct Node* head, int target, int data)
{
    struct Node* temp = head;
    while (temp && temp->data != target)
        temp = temp->next;

    if (!temp)
        printf("Node not found.\n");
    else
    {
        struct Node* newnode = (struct Node*)malloc(sizeof(struct Node));
        newnode->data = data;
        newnode->next = temp;
        newnode->prev = temp->prev;

        if (temp->prev)
            temp->prev->next = newnode;
        else
            head = newnode;

        temp->prev = newnode;
    }
    return head;
}

struct Node* deleteByValue(struct Node* head, int target)
{
    struct Node* temp = head;
    while (temp && temp->data != target)
        temp = temp->next;

    if (!temp)
        printf("Element not found.\n");
    else
    {
        if (temp->prev)
            temp->prev->next = temp->next;
        else
```

```
            head = temp->next;

        if (temp->next)
            temp->next->prev = temp->prev;

        free(temp);
    }
    return head;
}

int main()
{
    struct Node* head = NULL;
    int choice, value, target;

    do
    {
        printf("\nMenu:\n");
        printf("1. Create a doubly linked list\n");
        printf("2. Insert to the left of a node\n");
        printf("3. Delete by value\n");
        printf("4. Display the list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            head = create(head);
            break;
        case 2:
            printf("Enter the target node value: ");
            scanf("%d", &target);
            printf("Enter the value to be inserted: ");
            scanf("%d", &value);
            head = insertToLeft(head, target, value);
            break;
        case 3:
            printf("Enter the value to be deleted: ");
            scanf("%d", &value);
            head = deleteByValue(head, value);
            break;
          case 4:
            display(head);
            break;
          case 5:
            printf("Terminating the program.\n");
            break;
          default:
            printf("Please enter a valid choice.\n");
            break;
          }
      } while (choice != 5);

      return 0;
  }
```

OUTPUT:



```
Menu:
1. Create a doubly linked list
2. Insert to the left of a node
3. Delete by value
4. Display the list
5. Exit
Enter your choice: 1
Enter the number of elements to be inserted: 3
Enter the 1th element: 2
Enter the 2th element: 5
Enter the 3th element: 4
The values have been inserted.

Menu:
1. Create a doubly linked list
2. Insert to the left of a node
3. Delete by value
4. Display the list
5. Exit
Enter your choice: 2
Enter the target node value: 1
Enter the value to be inserted: 6
Node not found.

Menu:
1. Create a doubly linked list
2. Insert to the left of a node
3. Delete by value
4. Display the list
```



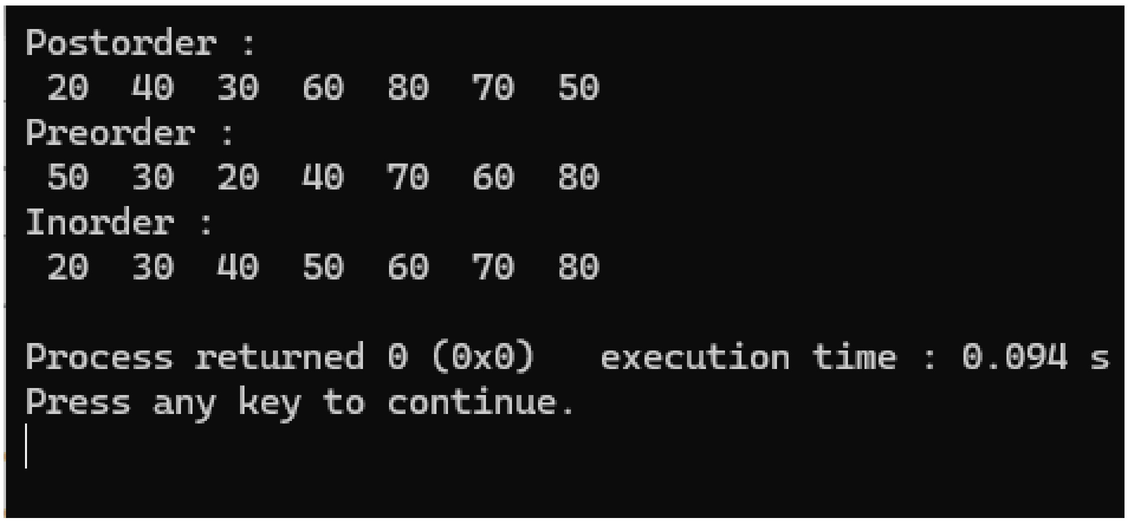
```
Menu:
1. Create a doubly linked list
2. Insert to the left of a node
3. Delete by value
4. Display the list
5. Exit
Enter your choice: 2
Enter the target node value: 1
Enter the value to be inserted: 6
Node not found.

Menu:
1. Create a doubly linked list
2. Insert to the left of a node
3. Delete by value
4. Display the list
5. Exit
Enter your choice: 3
Enter the value to be deleted: 5

Menu:
1. Create a doubly linked list
2. Insert to the left of a node
3. Delete by value
4. Display the list
5. Exit
Enter your choice: 4
4 <-> 2
```

Lab Program 9: Write a program
a) To construct a binary Search tree.
b) To traverse the tree using all the methods i.e., in-order, preorder and post order To display the elements in the tree.

```
#include  <stdio.h>
#include <stdlib.h>
struct Node{
int data;
struct Node *left, *right;
};
struct Node* newnode(int value)
{
struct Node* temp= (struct Node*)malloc(sizeof(struct Node));
temp->data = value;
temp->left = temp->right = NULL;
return temp;
}
struct Node* insertNode(struct Node* node, int value)
{
if (node == NULL) {
return newnode(value);
}
if (value < node->data)
{
node->left = insertNode(node->left, value);
}
else if (value > node->data)
{
node->right = insertNode(node->right, value);
}

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

}
}
void inOrder(struct Node* root)
{
if (root != NULL)
{
inOrder(root->left);
printf(" %d ", root->data);
inOrder(root->right);
}
}
void preOrder(struct Node* root)
{
if (root != NULL)
{
printf(" %d ", root->data);
preOrder(root->left);
preOrder(root->right);
}
}
int main()

{
struct Node* root = NULL;
root = insertNode(root, 50);
insertNode(root, 30);
```

```
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 70);
    insertNode(root, 60);
    insertNode(root, 80);
    printf("Postorder :\n");
    postOrder(root);
    printf("\n");

    printf("Preorder :\n");
    preOrder(root);
    printf("\n");
    printf("Inorder :\n");
    inOrder(root);
    printf("\n");
    return 0;
    }
```

OUTPUT:



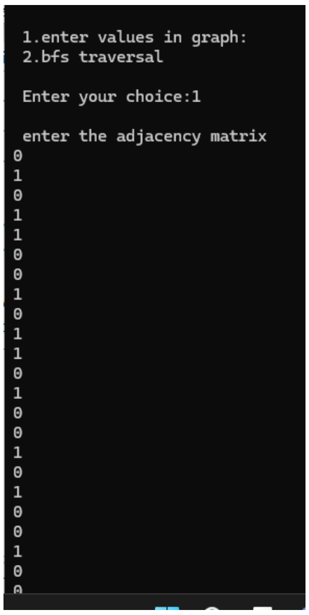
```
Postorder :
 20  40  30  60  80  70  50
Preorder :
 50  30  20  40  70  60  80
Inorder :
 20  30  40  50  60  70  80

Process returned 0 (0x0)   execution time : 0.094 s
Press any key to continue.
```

Lab Program 10: Write a program to traverse a graph using BFS method.

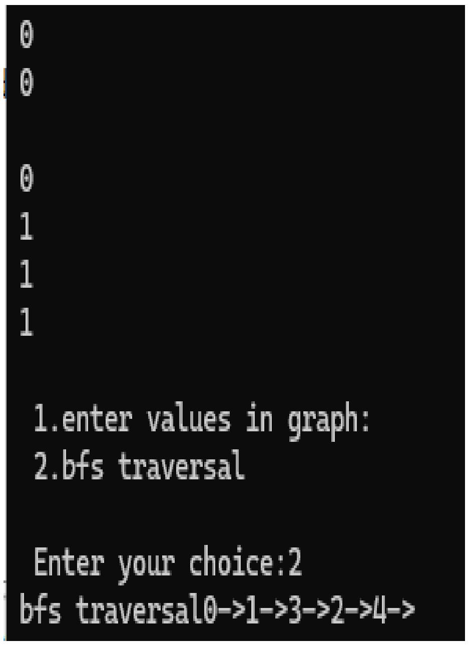
```
#include<stdio.h>
#define MAX 5
void bfs(int adj[][MAX],int visited[], int start)
{

   int queue[MAX], rear = -1, front = -1, i, k;
```

```
    for(k=0;k<MAX;k++)
    {
        visited[k] = 0;
    }

    queue[++rear] = start;
    ++front;
    visited[start] = 1;

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

int main()
{
    int visited [MAX] = {0};
    int adj [MAX][MAX],i,j,option;

    do
    {
        printf("\n 1.enter values in graph:");
        printf("\n 2.bfs traversal");
        printf("\n \n Enter your choice:");
        scanf("%d", &option);

        switch(option)
        {
        case 1:
            printf("\n enter the adjacency matrix");
            for(int i = 0;i<MAX;i++)
                for(int j = 0;j<MAX;j++)
                scanf("%d", &adj[i][j]);
            break;

        case 2:
            printf("bfs traversal");
            bfs(adj, visited, 0);
            break;
        }
    }
    while(option!=3);
    return 0;
}
```

OUTPUT:



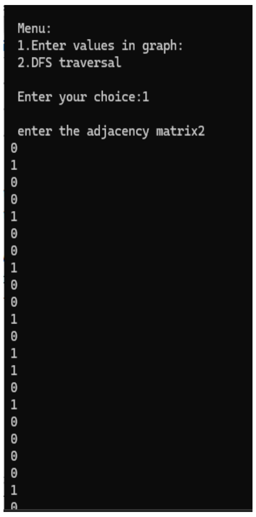
```
 1.enter values in graph:
 2.bfs traversal

 Enter your choice:1

 enter the adjacency matrix
0
1
0
1
1
0
0
1
0
1
1
0
1
0
0
1
0
1
0
0
1
0
```



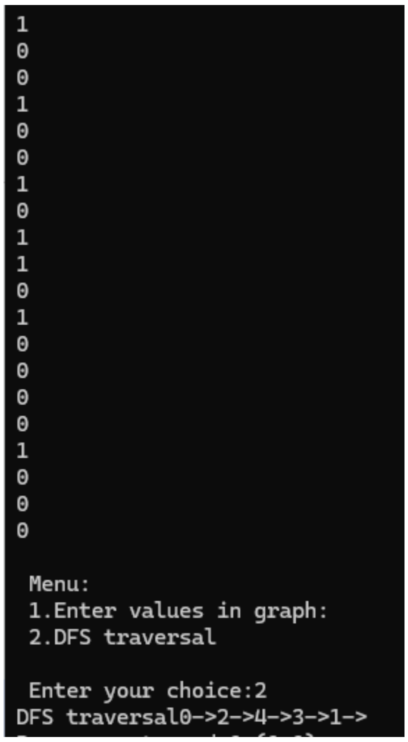
```
0
0

0
1
1
1

 1.enter values in graph:
 2.bfs traversal

 Enter your choice:2
bfs traversal0->1->3->2->4->
```

Lab Program 11:
Write a program to check whether given graph is connected or not using DFS method.

```
#include<stdio.h>
#define MAX 5
void dfs(int adj[][MAX],int visited[], int start)
{

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

   for(k=0;k<MAX;k++)
   {
      visited[k] = 0;
   }

   stack[++top] = start;

   visited[start] = 1;

   while(top!=-1)
   {
      start = stack[top--];
      printf("%d->",start);
      for(i = 0;i<MAX;i++)
      {
            if(adj[start][i] && visited[i]==0)
            {
                stack[++top] = i;
                visited[i] = 1;
            }
        }
    }
}

int main()
{
    int visited [MAX] = {0};
    int adj [MAX][MAX],i,j,option;

    do
    {
        printf("\n Menu:");
        printf("\n 1.Enter values in graph:");
        printf("\n 2.DFS traversal");
        printf("\n \n Enter your choice:");
        scanf("%d", &option);

        switch(option)
        {
        case 1:
            printf("\n enter the adjacency matrix");
            for(int i = 0;i<MAX;i++)
                for(int j = 0;j<MAX;j++)
                scanf("%d", &adj[i][j]);
            break;

        case 2:
            printf("DFS traversal");
            dfs(adj, visited, 0);
            break;
        }
    }
    while(option!=2);
    return 0;
}
```
OUTPUT:



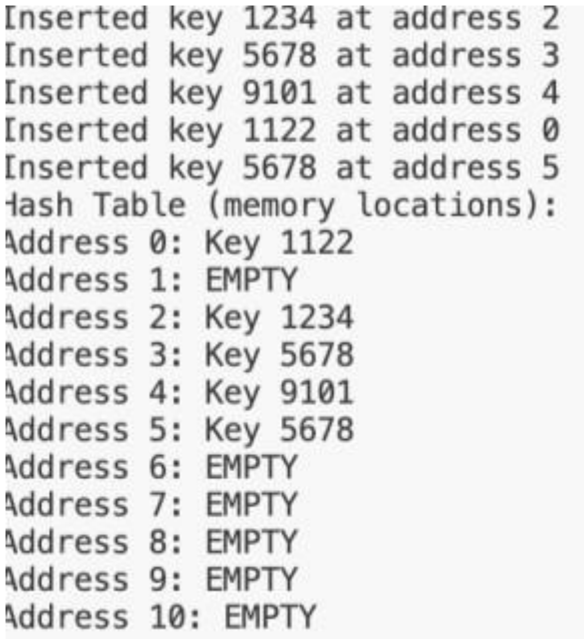
```
 Menu:
 1.Enter values in graph:
 2.DFS traversal

 Enter your choice:1

 enter the adjacency matrix2
0
1
0
0
1
0
0
1
0
0
1
0
1
1
0
1
0
0
0
0
1
0
```



```
1
0
0
1
0
0
1
0
1
1
0
1
0
0
0
0
1
0
0
0

 Menu:
 1.Enter values in graph:
 2.DFS traversal

 Enter your choice:2
DFS traversal0->2->4->3->1->
```

Lab Program 12: Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. 10 5 10 Assume that file F is maintained in memorybyaHash 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 anddevelop a Program in C that uses Hash function H: K -> L as H(K)=Kmod m(remainder method), and implement hashing technique to map a given key Kto theaddress space L. Resolve the collision (if any) using linear probing.

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_EMPLOYEES 100

#define EMPTY -1

typedef struct {

int key;

char name[50];

} Employee;

int m = 11;

int *hashTable;

Employee employees[MAX_EMPLOYEES];

int hashFunction(int key) {

return key % m;

}

void initializeHashTable() {

hashTable = (int*) malloc(m * sizeof(int));

for (int i = 0; i < m; i++) {

hashTable[i] = EMPTY;

}

}

int linearProbe(int key) {

int index = hashFunction(key);

int startIdx = index;

while (hashTable[index] != EMPTY) {

index = (index + 1) % m;

if (index == startIdx) {

return -1;
}
}
return index;
}
void insertEmployee(int key, char* name) {
int index = linearProbe(key);
if (index != -1) {
hashTable[index] = key;
printf("Inserted key %d at address %d\n", key, index);
} else {
printf("Error: Hash table is full, unable to insert key %d\n", key);
}
}
void displayHashTable() {
printf("Hash Table (memory locations):\n");
for (int i = 0; i < m; i++) {
if (hashTable[i] != EMPTY) {
printf("Address %d: Key %d\n", i, hashTable[i]);
} else {
printf("Address %d: EMPTY\n", i);
}
}
}
int main() {
initializeHashTable();
insertEmployee(1234, "John");
insertEmployee(5678, "Avinash");
insertEmployee(9101, "Amy");
insertEmployee(1122, "Balaji");

insertEmployee(5678, "Chandan");

displayHashTable();

free(hashTable);

return 0;

}
```



```
Inserted key 1234 at address 2
Inserted key 5678 at address 3
Inserted key 9101 at address 4
Inserted key 1122 at address 0
Inserted key 5678 at address 5
Hash Table (memory locations):
Address 0: Key 1122
Address 1: EMPTY
Address 2: Key 1234
Address 3: Key 5678
Address 4: Key 9101
Address 5: Key 5678
Address 6: EMPTY
Address 7: EMPTY
Address 8: EMPTY
Address 9: EMPTY
Address 10: EMPTY
```