VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

ARCHITA V (1BM23CS050)

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 ARCHITA V **(1BM23CS050)**, who is a bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 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.

Sneha S Bagalkot
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Course outcomes:

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

Lab program 1

Write a program to simulate the working of stack using an array with the following a) Push b) Pop c) Display The program should print appropriate messages for stack overflow, stack underflow.

```
#include<stdio.h>
#define MAX 3
int s[10],top=-1,i,item,ch;
void push();
int pop();
void display();
void main()
{
    while(1)
    {
    printf("\n 1.PUSH \n 2.POP \n 3.DISPLAY \n 4.EXIT \n");
    printf("Enter your 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");
        return;
     }
     top=top+1;
     printf("Enter the element to be pushed: ");
```

```
    top=top+1;
    printf("Enter the element to be pushed: ");
    scanf("%d",&item);
    s[top]=item;
}
int pop()
{
    if(top==-1)
    {
        printf("stack underflow");
        return(-1);
    }
    item=s[top];
    top=top-1;
    return item;
}
void display()
{
    if(top==-1)
    {
```

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

OUTPUT

```
 1.PUSH
 2.POP
 3.DISPLAY
 4.EXIT
Enter your choice: 1
Enter the element to be pushed: 10

 1.PUSH
 2.POP
 3.DISPLAY
 4.EXIT
Enter your choice: 1
Enter the element to be pushed: 2

 1.PUSH
 2.POP
 3.DISPLAY
 4.EXIT
Enter your choice: 1
Enter the element to be pushed: 3

 1.PUSH
 2.POP
 3.DISPLAY
 4.EXIT
Enter your choice: 3
Stack contents: 3
2
10

 1.PUSH
 2.POP
 3.DISPLAY
 4.EXIT
```

Lab program 2

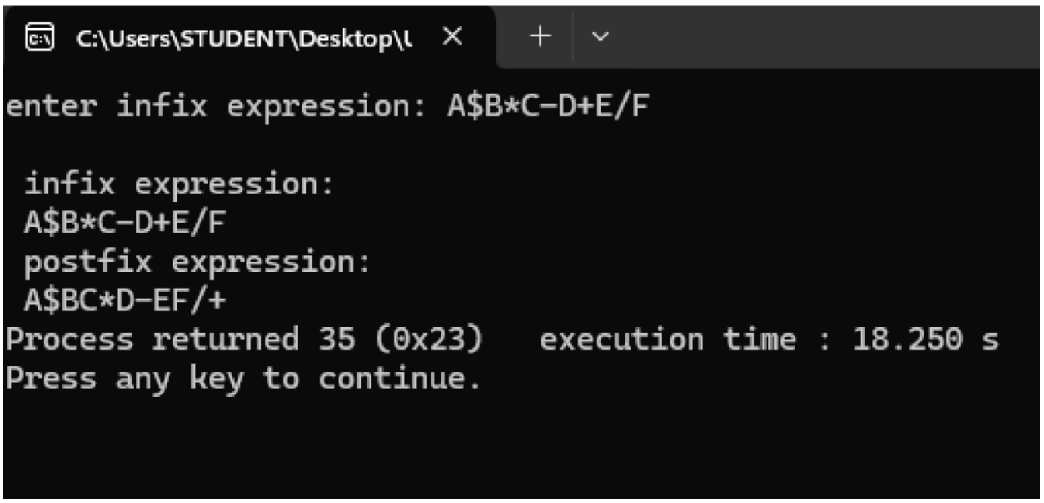
Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

```
#include<stdio.h>
#include<string.h>
void infixtopostfix();
void push(char symbol);
char pop();
int pred(char symb);
int index=0,pos=0,top=-1,length;
char symbol,temp,infix[20],postfix[20],stack[20];
void main()
{
    printf("enter infix expression: ");
    scanf("%s",infix);
     infixtopostfix();
     postfix[pos] = '\0';
    printf("\n infix expression:\n %s ",infix);
    printf("\n postfix expression:\n %s ",postfix);
}
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;
                        pos++;
                        temp=pop();
                    }
                    break;
            case'+':
            case'-':
            case'*':
            case'/':
```

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

```
    }
    while(top>0)
    {
        temp=pop();
        postfix[pos++]=temp;
    }
}
void push(char symbol)
{
    top=top+1;
    stack[top]=symbol;
}
char pop()
{
    char symb;
    symb=stack[top];
    top=top-1;
    return(symb);
}
int pred(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



LEETCODE 2


```
15.5.24                                    12
Leet Code - Clear digit :

class sol {
    String clear dig (String s)
    {
        int l = s.length();
        for (int i=0; i<l; i++)
        {
            if (isDigit (s[i]))
            {
                if (l!=0)(s.erase (i-1,2);
                    l-=2 ; i-=2;
                }
                else
                { s.erase (i,2); l--; i--; }
            }
        }

Case 1                          Case 2
Input S = "abc"                 S = "cb34"
Output "abc"                    Output " "
Expected "abc"                  Expected " "
```

Lab program 3A

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

```c
#include<stdio.h>
#define MAX 5
int queue[MAX];
int front=-1;
int rear=-1;
void insert(int value)
{
  if(rear==MAX-1)
  {
    printf("Queue is full.Cannot insert %d \n",value);
    return;
  }
  if(rear==-1 && front==-1)
  {
    front=0;
  }
  rear++;
  queue[rear]=value;
  printf("Inserted: %d\n",value);
}
int delete()
{
    if(rear==-1 && front==-1)
    {
        printf("Queue is empty.Cannot delete\n");
        return -1;
    }
    int item=queue[front];
    if(front>=rear)
    {
        front=-1;
        rear=-1;
    }
    else
    {
        front++;
```

```
    }
    printf("Deleted: %d\n",item);
    return item;
}
void display()
{
    if(rear==-1 && front==-1)
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue contents:");
    for(int i=front;i<=rear;i++)
    {
        printf("%d\t",queue[i]);
    }
    printf("\n");
}
int main()
{
    int choice,value;
    do
    {
        printf("\n 1.INSERT \n 2.DELETE \n 3.DISPLAY \n 4.EXIT\n");
        printf("Enter your choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
            printf("Enter the value to be inserted: ");
            scanf("%d",&value);
            insert(value);
            break;
        case 2:
            delete();
            break;
```

```
        switch(choice)
        {
        case 1:
            printf("Enter the value to be inserted: ");
            scanf("%d",&value);
            insert(value);
            break;
        case 2:
            delete();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting\n");
            break;
        default:printf("Invalid choice \n");
        }
    } while(choice!=4);
   return 0;
}
```

OUTPUT

```
 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 1
Enter the value to be inserted: 10
Inserted: 10

 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 1
Enter the value to be inserted: 20
Inserted: 20

 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 3
Queue contents:10       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: 20

 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 2
Queue is empty.Cannot delete

 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 3
Queue is empty
```

```
 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 2
Queue is empty.Cannot delete

 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 3
Queue is empty

 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 4
Exiting
```

Lab program 3B

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

```c
#include<stdio.h>
#define SIZE 5
int circularqueue[SIZE];
int front=-1;
int rear=-1;
void insert(int value)
{
    if(front=(rear+1)%SIZE)
    {
        printf("Circular queue is full.Cannot insert %d \n",value);
        return;
    }
    if(front==-1 && rear==-1)
    {
        front=0;
        rear=0;
    }
    else
    {
        rear==(rear+1)%SIZE;

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

}
int delete()
{
    if(front==-1 && rear==-1 )
    {
        printf("Circular queue is empty.Cannot delete\n");
        return -1;
    }
    int item=circularqueue[front];
    if(front==rear)
    {
```

```
        front=-1;
        rear=-1;
    }
    else
    {
        front==(front+1)%SIZE;
    }
    printf("Deleted: %d \n",item);
    return item;
}
void display()
{
    if (front==-1 && rear==-1)
    {
        printf("Circular queue is empty \n");
        return;
    }
    printf("Circular queue contents: ");
    if(front<=rear)
    {
        for(int i=front;i<=rear;i++)
        {
            printf("%d \t",circularqueue[i]);
        }
        printf("\n");
    }
    else
    {
        for(int i=front;i<SIZE;i++)
        {
            printf("%d \t",circularqueue[i]);
        }
        for(int i=0;i<=rear;i++)
        {
             printf("%d \t",circularqueue[i]);
        }
```

```
        {
            printf("%d \t",circularqueue[i]);
        }
        for(int i=0;i<=rear;i++)
        {
             printf("%d \t",circularqueue[i]);
        }
        printf("\n");
    }
}
int main()
{
    int choice,value;
    do
    {
        printf("\n 1.INSERT \n 2.DELETE \n 3.DISPLAY \n 4.EXIT\n");
        printf("Enter your choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:printf("Enter the value to be inserted: ");
                   scanf("%d",&value);
                   insert(value);
                   break;
            case 2:delete();
                   break;
            case 3:display();
                   break;
            case 4:printf("Exiting\n");
                   break;
        }
    } while(choice!=4);
return 0;
}
```

OUTPUT

```
 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 1
Enter the value to be inserted: 12
Inserted: 12

 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 1
Enter the value to be inserted: 23
Inserted: 23

 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 2
Deleted: 0

 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 2
Deleted: 0

 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 3
Circular queue contents: 0      0       0       0       0

 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
Enter your choice: 4
Exiting

Process returned 0 (0x0)   execution time : 14.938 s
Press any key to continue.
```

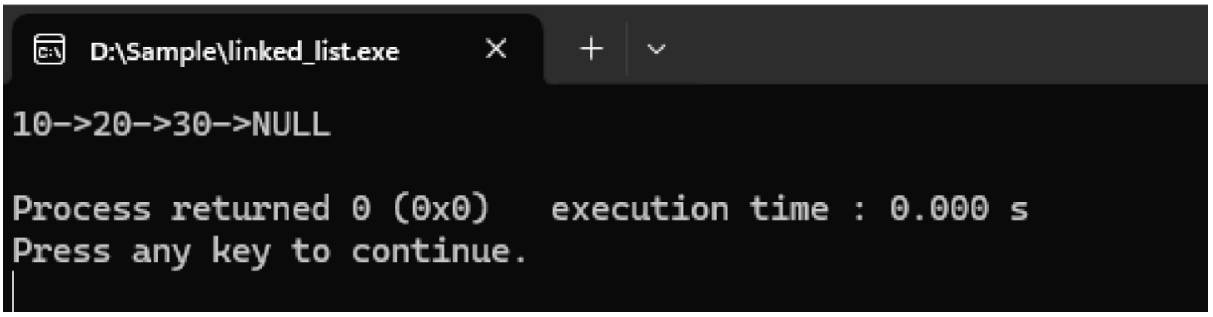
Lab program 4

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

```c
#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 insert(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 display(struct node*head)
{
    struct node*temp=head;
    while(temp!=NULL)
    {
        printf("%d->",temp->data);
        temp=temp->next;
    }
    printf("NULL\n");
}
int main()
```

```
        }
        temp->next=newnode;
    }
}
void display(struct node*head)
{
    struct node*temp=head;
    while(temp!=NULL)
    {
        printf("%d->",temp->data);
        temp=temp->next;
    }
    printf("NULL\n");
}
int main()
{
    struct node*head=NULL;
    insert (&head,10);
    insert (&head,20);
    insert (&head,30);
    display(head);
    return 0;
}
```

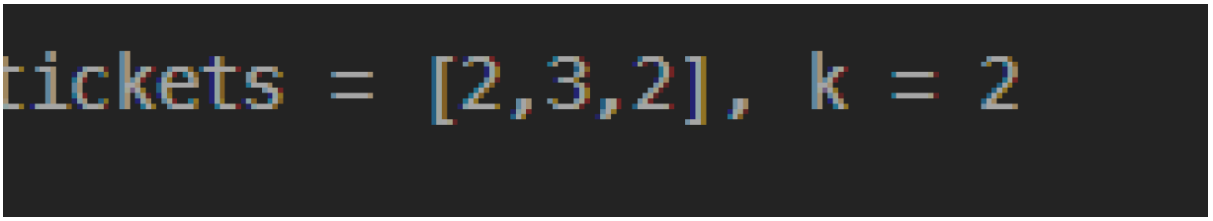
OUTPUT


D:\Sample\linked_list.exe
10->20->30->NULL
Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.

LEETCODE 4

```
1
2  public class Solution {
3      public int TimeRequiredToBuy(int[] tickets, int k) {
4          int n = tickets.Length;
5          int time = 0;
6
7          for (int i = 0; i < n; i++) {
8              if (i <= k) {
9                  time += Math.Min(tickets[k], tickets[i]);
10             } else {
11                 time += Math.Min(tickets[k] - 1, tickets[i]);
12             }
13         }
14
15         return time;
16     }
17 }
```

OUTPUT



Lab program 5

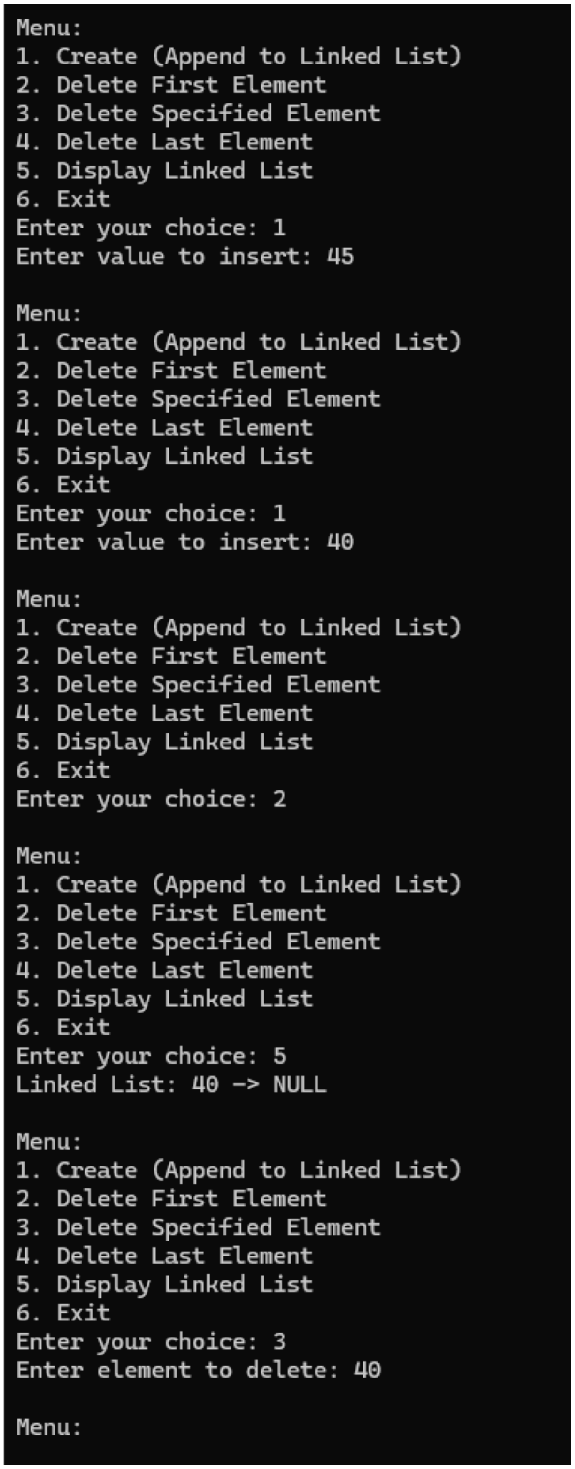
WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.

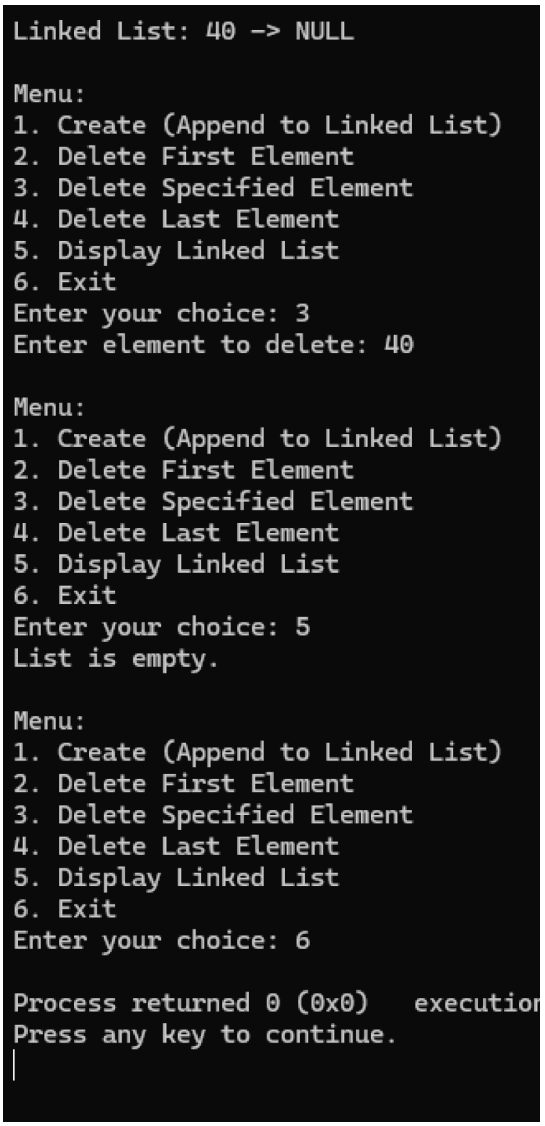
```c
#include <stdio.h>
#include <stdlib.h>

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

void deleteElement(struct Node** head, int key) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }

    if ((*head)->data == key) {
        struct Node* temp = *head;
        *head = (*head)->next;
        free(temp);
        return;
```

```
    if ((*head)->data == key) {
        struct Node* temp = *head;
        *head = (*head)->next;
        free(temp);
        return;
    }

    struct Node* temp = *head;
    while (temp->next != NULL && temp->next->data != key) {
        temp = temp->next;
    }

    if (temp->next != NULL) {
        struct Node* nodeToDelete = temp->next;
        temp->next = temp->next->next;
        free(nodeToDelete);
    } else {
        printf("Element %d not found in the list.\n", key);
    }
}

void deleteLast(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }

    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        return;
    }

    struct Node* temp = *head;
    while (temp->next->next != NULL) {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
}

void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
```

```
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}


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

    while (1) {
        printf("\nMenu:\n");
        printf("1. Create (Append to Linked List)\n");
        printf("2. Delete First Element\n");
        printf("3. Delete Specified Element\n");
        printf("4. Delete Last Element\n");
        printf("5. Display Linked List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                append(&head, value);
                break;
            case 2:
                deleteFirst(&head);
                break;
            case 3:
                printf("Enter element to delete: ");
                scanf("%d", &key);
                deleteElement(&head, key);
                break;
            case 4:
                deleteLast(&head);
                break;
            case 5:
                displayList(head);
                break;
            case 6:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

OUTPUT

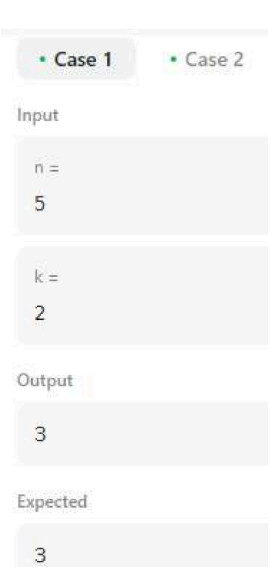

Menu:
1. Create (Append to Linked List)
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 1
Enter value to insert: 45
Menu:
1. Create (Append to Linked List)
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 1
Enter value to insert: 40
Menu:
1. Create (Append to Linked List)
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 2
Menu:
1. Create (Append to Linked List)
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 5
Linked List: 40 -> NULL
Menu:
1. Create (Append to Linked List)
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 3
Enter element to delete: 40
Menu:

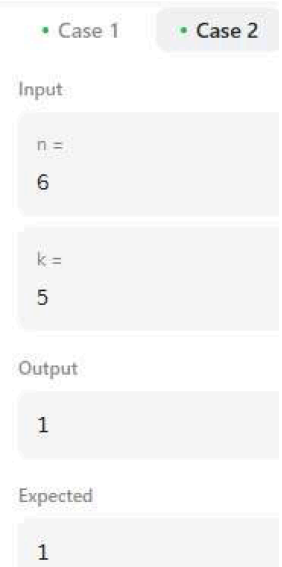

Linked List: 40 -> NULL
Menu:
1. Create (Append to Linked List)
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 3
Enter element to delete: 40
Menu:
1. Create (Append to Linked List)
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 5
List is empty.
Menu:
1. Create (Append to Linked List)
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 6
Process returned 0 (0x0)
Press any key to continue.

LEETCODE 5

```
class Solution {
public:
    int findTheWinner(int n, int k) {
    int *a = (int*)malloc(n * sizeof(int));
    for (int i = 0; i < n; ++i) {
        a[i] = i + 1;
    }
    int size = n;
    int start = 0;
    while (size > 1) {
        start = (start + (k - 1)) % size;
        for (int j = start; j < size - 1; ++j) {
            a[j] = a[j + 1];
        }
        --size;
    }
    int winner = a[0];
    free(a);
    return winner;
}
};
```

OUTPUT





Lab program 6 A

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

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

// Define the structure of a node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the end of the list
struct Node* insertEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        return newNode;
    }
    struct Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    return head;
}
```

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

// Function to sort the linked list (Bubble Sort)
struct Node* sortList(struct Node* head) {
    if (head == NULL) return head;
    struct Node* current = head;
    struct Node* index = NULL;
    int temp;
    while (current != NULL) {
        index = current->next;
        while (index != NULL) {
            if (current->data > index->data) {
                // Swap data
                temp = current->data;
                current->data = index->data;
                index->data = temp;
            }
            index = index->next;
        }
        current = current->next;
```

```c
struct Node* reverseList(struct Node* head) {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

// Function to concatenate two linked lists
struct Node* concatenate(struct Node* head1, struct Node* head2) {
    if (head1 == NULL) return head2;
    if (head2 == NULL) return head1;

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

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

    // Creating first linked list
    list1 = insertEnd(list1, 5);
```

```
    list1 = insertEnd(list1, 9);
    list1 = insertEnd(list1, 3);

    // Creating second linked list
    list2 = insertEnd(list2, 8);
    list2 = insertEnd(list2, 2);
    list2 = insertEnd(list2, 7);

    printf("List 1: ");
    printList(list1);

    printf("List 2: ");
    printList(list2);

    // Sorting list1
    list1 = sortList(list1);
    printf("\nSorted List 1: ");
    printList(list1);

    // Reversing list1
    list1 = reverseList(list1);
    printf("Reversed List 1: ");
    printList(list1);

    // Concatenating list1 and list2
    struct Node* concatenatedList = concatenate(list1, list2);
    printf("\nConcatenated List: ");
    printList(concatenatedList);

    return 0;
}
```

OUTPUT

```
List 1: 5 -> 1 -> 9 -> 3 -> NULL
List 2: 8 -> 2 -> 7 -> NULL

Sorted List 1: 1 -> 3 -> 5 -> 9 -> NULL
Reversed List 1: 9 -> 5 -> 3 -> 1 -> NULL

Concatenated List: 9 -> 5 -> 3 -> 1 -> 8 -> 2 -> 7 -> NULL

Process returned 0 (0x0)   execution time : 0.054 s
Press any key to continue.
```

Lab program 6 B

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

Stack operation-

```c
#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));
    if (!newNode) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

int isEmpty(struct Node* top) {
    return top == NULL;
}

void push(struct Node** top, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *top;
    *top = newNode;
    printf("%d pushed onto the stack\n", data);
}
```

```c
int pop(struct Node** top) {
    if (isEmpty(*top)) {
        printf("Stack underflow\n");
        return -1;
    }
    struct Node* temp = *top;
    int poppedData = temp->data;
    *top = (*top)->next;
    free(temp);
    return poppedData;
}

int peek(struct Node* top) {
    if (isEmpty(top)) {
        printf("Stack is empty\n");
        return -1;
    }
    return top->data;
}


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

```
    printf("\n");
}


int main() {
    struct Node* stackTop = NULL;

    int choice, value;
    do {
        printf("\n--- Stack Operations ---\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Peek\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&stackTop, value);
                break;
            case 2:
                value = pop(&stackTop);
                if (value != -1)
                    printf("Popped value: %d\n", value);
                break;
            case 3:
                value = peek(stackTop);
```

```
        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&stackTop, value);
                break;
            case 2:
                value = pop(&stackTop);
                if (value != -1)
                    printf("Popped value: %d\n", value);
                break;
            case 3:
                value = peek(stackTop);
                if (value != -1)
                    printf("Top element: %d\n", value);
                break;
            case 4:
                display(stackTop);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

OUTPUT

```
--- Stack Operations ---
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 22
22 pushed onto the stack

--- Stack Operations ---
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 30
30 pushed onto the stack

--- Stack Operations ---
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 3
Top element: 30

--- Stack Operations ---
1. Push
2. Pop
3. Peek
4. Display
```

Queue operation-

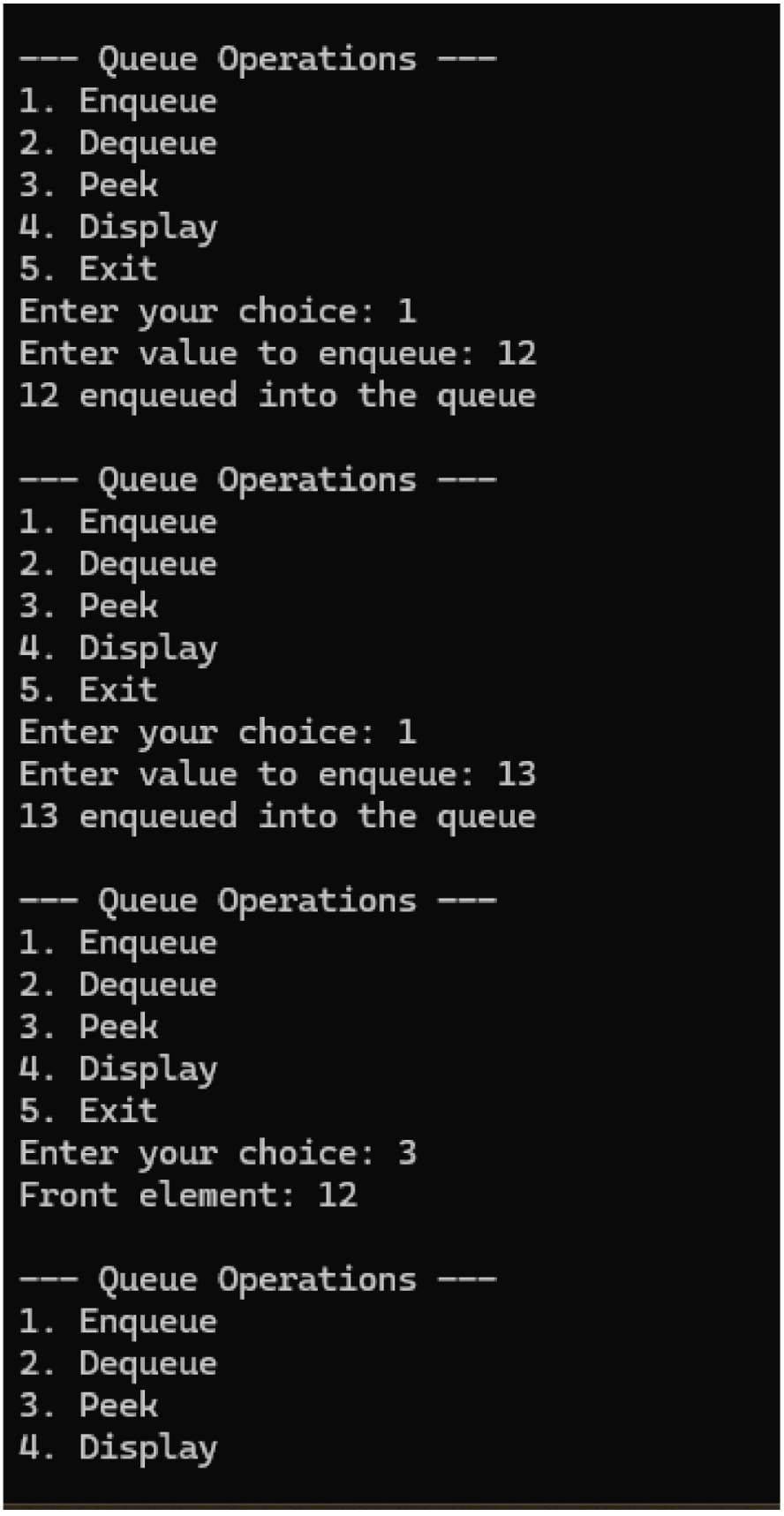
```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};

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

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

struct Queue* createQueue() {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = q->rear = NULL;
    return q;
}

int isEmpty(struct Queue* q) {
    return q->front == NULL;
```

```
    return q->front == NULL;
}

void enqueue(struct Queue* q, int data) {
    struct Node* newNode = createNode(data);
    if (q->rear == NULL) { // If queue is empty
        q->front = q->rear = newNode;
        printf("%d enqueued into the queue\n", data);
        return;
    }
    q->rear->next = newNode;
    q->rear = newNode;
    printf("%d enqueued into the queue\n", data);
}

int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue underflow\n");
        return -1;
    }
    struct Node* temp = q->front;
    int dequeuedData = temp->data;
    q->front = q->front->next;

    if (q->front == NULL)
        q->rear = NULL;

    free(temp);
    return dequeuedData;
}
```

```c
int peek(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    return q->front->data;
}

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


int main() {
    struct Queue* queue = createQueue();

    int choice, value;
    do {
        printf("\n--- Queue Operations ---\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
```

```c
        switch (choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(queue, value);
                break;
            case 2:
                value = dequeue(queue);
                if (value != -1)
                    printf("Dequeued value: %d\n", value);
                break;
            case 3:
                value = peek(queue);
                if (value != -1)
                    printf("Front element: %d\n", value);
                break;
            case 4:
                display(queue);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

OUTPUT


--- Queue Operations ---
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 12
12 enqueued into the queue
--- Queue Operations ---
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 13
13 enqueued into the queue
--- Queue Operations ---
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 3
Front element: 12
--- Queue Operations ---
1. Enqueue
2. Dequeue
3. Peek
4. Display

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

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

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

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

Node* createDoublyLinkedList(int n) {
    if (n <= 0) return NULL;

    int data;
    printf("Enter data for node 1: ");
    scanf("%d", &data);

    Node* head = createNode(data);
    Node* temp = head;

    for (int i = 2; i <= n; i++) {
        printf("Enter data for node %d: ", i);
        scanf("%d", &data);

        Node* newNode = createNode(data);
        temp->next = newNode;
        newNode->prev = temp;
        temp = newNode;
    }
    return head;
}


void insertLeft(Node** head, int target, int data) {
    Node* temp = *head;
    while (temp != NULL && temp->data != target) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Node with value %d not found!\n", target);
        return;
    }

    Node* newNode = createNode(data);
    newNode->next = temp;
    newNode->prev = temp->prev;
```

```
    if (temp->prev != NULL) {
        temp->prev->next = newNode;
    } else {
        *head = newNode;
    }
    temp->prev = newNode;
}


void deleteNode(Node** head, int value) {
    Node* temp = *head;

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

    if (temp == NULL) {
        printf("Node with value %d not found!\n", value);
        return;
    }

    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    } else {
        *head = temp->next;
    }

    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }

    free(temp);
    printf("Node with value %d deleted!\n", value);
}

void displayList(Node* head) {
    Node* temp = head;
    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}


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

    while (1) {
        printf("\nMenu:\n");
        printf("1. Create Doubly Linked List\n");
```

```
        printf("Node with value %d not found!\n", target);
        return;
    }

    Node* newNode = createNode(data);
    newNode->next = temp;
    newNode->prev = temp->prev;

    if (temp->prev != NULL) {
        temp->prev->next = newNode;
    } else {
        *head = newNode;
    }
    temp->prev = newNode;
}


void deleteNode(Node** head, int value) {
    Node* temp = *head;

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

    if (temp == NULL) {
        printf("Node with value %d not found!\n", value);
        return;
    }

    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    } else {
        *head = temp->next;
    }

    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }

    free(temp);
    printf("Node with value %d deleted!\n", value);
}

void displayList(Node* head) {
    Node* temp = head;
    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

```
        temp = temp->next;
    }
    printf("\n");
}int main() {
    Node* head = NULL;
    int choice, n, target, data, value;
    while (1) {
        printf("\nMenu:\n");
        printf("1. Create Doubly Linked List\n");
        printf("2. Insert a Node to the Left of a Specific Node\n");
        printf("3. Delete a Node by Value\n");
        printf("4. Display List\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the number of nodes: ");
                scanf("%d", &n);
                head = createDoublyLinkedList(n);
                break;
                case 2:
                if (head == NULL) {
                    printf("List is empty! Create a list first.\n");
                } else {
                    printf("Enter the target value: ");
                    scanf("%d", &target);
                    printf("Enter the value to insert: ");
                    scanf("%d", &data);
                    insertLeft(&head, target, data);
                }
                break;case 3:
                if (head == NULL) {
                    printf("List is empty! Create a list first.\n");
                } else {
                    printf("Enter the value to delete: ");
                    scanf("%d", &value);
                    deleteNode(&head, value);
                }
                break;case 4:
                if (head == NULL) {
                    printf("List is empty!\n");
                } else {
                    displayList(head);
                }
                break;case 5:
                printf("Exiting...\n");
                exit(0);default:
                printf("Invalid choice! Please try again.\n");}
    }return 0;
}
```

OUTPUT

```
Menu:
1. Create Doubly Linked List
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display List
5. Exit
Enter your choice: 1
Enter the number of nodes: 3
Enter data for node 1: 13
Enter data for node 2: 14
Enter data for node 3: 15

Menu:
1. Create Doubly Linked List
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display List
5. Exit
Enter your choice: 2
Enter the target value: 14
Enter the value to insert: 25

Menu:
1. Create Doubly Linked List
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display List
5. Exit
Enter your choice: 4
Doubly Linked List: 13 25 14 15

Menu:
1. Create Doubly Linked List
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display List
5. Exit
Enter your choice: 3
Enter the value to delete: 15
Node with value 15 deleted!

Menu:
1. Create Doubly Linked List
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display List
5. Exit
Enter your choice: 4
Doubly Linked List: 13 25 14

Menu:
1. Create Doubly Linked List
2. Insert a Node to the Left of a Specific Node
3. Delete a Node by Value
4. Display List
5. Exit
Enter your choice: 5
Exiting...
```

Lab program 8

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

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
Node* insertNode(Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
```

```c
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else if (data > root->data) {
        root->right = insertNode(root->right, data);
    }
    return root;
}
void inorder(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
void preorder(Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
```

```
    }
}
void preorder(Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}
void postorder(Node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}
void displayTree(Node* root) {
    printf("\nInorder : ");
    inorder(root);
    printf("\nPreorder : ");

    inorder(root);
    printf("\nPreorder : ");
    preorder(root);
    printf("\nPostorder : ");
    postorder(root);
}
int main() {
    Node* root = NULL;
    int choice, data;
    while (1) {
        printf("\n1. Insert\n2. Display\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &data);
                root = insertNode(root, data);
                break;
```

```
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &data);
                root = insertNode(root, data);
                break;
            case 2:
                displayTree(root);
                break;
            case 3:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
}
```

OUTPUT

```
1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter the value to insert: 12

1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter the value to insert: 13

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

Inorder : 12 13
Preorder : 12 13
Postorder : 13 12
1. Insert
2. Display
3. Exit
Enter your choice: 3
Exiting...

Process returned 0 (0x0)   execution t
Press any key to continue.
```

Lab program 9A

Write a program to traverse a graph using the BFS method.

```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 50
void bfs(int graph[MAX][MAX],int visited[MAX],int start,int n);
int main()
{
    int n;
    int graph[MAX][MAX];
    int visited[MAX]={0};
    int start;
    printf("Enter the number of vertices: ");
    scanf("%d",&n);
    printf("Enter the adjacency matrix of the graph:\n");
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            scanf("%d",&graph[i][j]);
        }
    }
    printf("Enter the starting node (0 to %d): ",n-1);
    scanf("%d",&start);
    printf("BFS traversal starting from node %d:\n",start);
    bfs(graph,visited,start,n);
    return 0;
}
void bfs(int graph[MAX][MAX],int visited[MAX],int start,int n)
{
    int queue[MAX],front=0,rear=0;
    queue[rear++]=start;
    visited[start]=1;
    while(front<rear)
    {
        int current=queue[front++];
        printf("%d",current);
        for(int i=0;i<n;i++)
        {
            if(graph[current][i]==1 &&!visited[i])
            {
```

```c
    printf("Enter the starting node (0 to %d): ",n-1);
    scanf("%d",&start);
    printf("BFS traversal starting from node %d:\n",start);
    bfs(graph,visited,start,n);
    return 0;
}
void bfs(int graph[MAX][MAX],int visited[MAX],int start,int n)
{
    int queue[MAX],front=0,rear=0;
    queue[rear++]=start;
    visited[start]=1;
    while(front<rear)
    {
        int current=queue[front++];
        printf("%d",current);
        for(int i=0;i<n;i++)
        {
            if(graph[current][i]==1 &&!visited[i])
            {
                queue[rear++]=i;
                visited[i]=1;
            }
        }
    }
}
```

OUTPUT

```
PS C:\Users\STUDENT\Downloads\canteen\style> cd "c:\Users\STUDENT\Down
Enter the number of vertices: 3
Enter the adjacency matrix of the graph:
0 0 1
1 0 0
1 0 1
Enter the starting node (0 to 2): 1
BFS traversal starting from node 1:
102
PS C:\Users\STUDENT\Downloads\canteen\style>
```

Lab program 9B

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

```c
#include<stdio.h>
#include<stdlib.h>
void dfs(int vertex,int visit[],int graph[][100],int n)
{
    visit[vertex]=1;
    for(int i=0;i<n;i++)
    {
        if(graph[vertex][i]==1 && !visit[i])
        {
            dfs(i,visit,graph,n);
        }
    }
}
int main()
{
    int n;
    int graph[100][100];
    int visit[100]={0};
    printf("Enter the number of vertices:\n");
    scanf("%d",&n);
    printf("Enter the adjacency matrix of the graph:\n");
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            scanf("%d",&graph[i][j]);
        }
    }
    dfs(0,visit,graph,n);
    int isConnected=1;
    for(int i=0;i<n;i++)
    {
        if(!visit[i])
        {
            isConnected=0;
            break;
        }
    }
    if(isConnected)
```

```
    printf("Enter the number of vertices:\n");
    scanf("%d",&n);
    printf("Enter the adjacency matrix of the graph:\n");
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            scanf("%d",&graph[i][j]);
        }
    }
    dfs(0,visit,graph,n);
    int isConnected=1;
    for(int i=0;i<n;i++)
    {
        if(!visit[i])
        {
            isConnected=0;
            break;
        }
    }
    if(isConnected)
    {
        printf("The graph is connected\n");
    }
    else
    {
        printf("The graph is not connected\n");
    }
    return 0;
}
```

OUTPUT

```
PS C:\Users\STUDENT\Downloads\canteen\style> cd "c:\Users\STUDENT\Download
Enter the number of vertices:
3
Enter the adjacency matrix of the graph:
1 1 1
0 0 1
0 0 0
The graph is connected
PS C:\Users\STUDENT\Downloads\canteen\dfs> cd "c:\Users\STUDENT\Downloads\
Enter the number of vertices:
2
Enter the adjacency matrix of the graph:
0 0
1 1
The graph is not connected
PS C:\Users\STUDENT\Downloads\canteen\dfs>
```

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>
#include <string.h>
#define MAX 50
typedef struct {
    int key;
    char name[50];
} Employee;
void initializeHashTable(Employee *hashTable, int size) {
    for (int i = 0; i < size; i++) {
        hashTable[i].key = -1;
        strcpy(hashTable[i].name, "");
    }
}
int hashFunction(int key, int m) {
    return key % m;
}
void insert(Employee *hashTable, int m, int key, char *name) {
    int index = hashFunction(key, m);
    int originalIndex = index;
```

```c
    int index = hashFunction(key, m);
    int originalIndex = index;
    int i = 0;
    while (hashTable[index].key != -1 && i < m) {
        if (hashTable[index].key == key) {
            printf("Error: Duplicate key %d.\n", key);
            return;
        }
        index = (originalIndex + ++i) % m;
    }
    if (i >= m) {
        printf("Error: Hash table is full. Cannot insert key %d.\n", key);
        return;
    }
    hashTable[index].key = key;
    strcpy(hashTable[index].name, name);
    printf("Inserted key %d at index %d.\n", key, index);
}
void search(Employee *hashTable, int m, int key) {
    int index = hashFunction(key, m);
```

```
    strcpy(hashTable[index].name, name);
    printf("Inserted key %d at index %d.\n", key, index);
}
void search(Employee *hashTable, int m, int key) {
    int index = hashFunction(key, m);
    int originalIndex = index;
    int i = 0;

    while (hashTable[index].key != -1 && i < m) {
        if (hashTable[index].key == key) {
            printf("Key %d found at index %d. Name: %s\n", key, index, hashTable[index].name);
            return;
        }
        index = (originalIndex + ++i) % m;
    }

    printf("Key %d not found.\n", key);
}
void displayHashTable(Employee *hashTable, int m) {
    printf("\nHash Table:\n");
```

```
void displayHashTable(Employee *hashTable, int m) {
    printf("\nHash Table:\n");
    for (int i = 0; i < m; i++) {
        if (hashTable[i].key != -1) {
            printf("Index %d: Key = %d, Name = %s\n", i, hashTable[i].key, hashTable[i].name);
        } else {
            printf("Index %d: Empty\n", i);
        }
    }
}
int main() {
    int m;
    printf("Enter the number of memory locations (m): ");
    scanf("%d", &m);

    Employee *hashTable = (Employee *)malloc(m * sizeof(Employee));
    initializeHashTable(hashTable, m);

    int choice;
    do {
```

```
int choice;
do {
    printf("\nMenu:\n");
    printf("1. Insert\n");
    printf("2. Search\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1: {
            int key;
            char name[50];
            printf("Enter the 4-digit key: ");
            scanf("%d", &key);
            printf("Enter the name: ");
            scanf("%s", name);
            insert(hashTable, m, key, name);
            break;

        case 2: {
            int key;
            printf("Enter the 4-digit key to search: ");
            scanf("%d", &key);
            search(hashTable, m, key);
            break;
        }
        case 3:
            displayHashTable(hashTable, m);
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 4);

free(hashTable);
return 0;
```

OUTPUT

```
Enter the number of memory locations (m): 3

Menu:
1. Insert
2. Search
3. Display
4. Exit
Enter your choice: 1
Enter the 4-digit key: 1234
Enter the name: ABD
Inserted key 1234 at index 1.

Menu:
1. Insert
2. Search
3. Display
4. Exit
Enter your choice: 1
Enter the 4-digit key: 9985
Enter the name: KUL
Inserted key 9985 at index 2.

Menu:
1. Insert
2. Search
3. Display
4. Exit
Enter your choice: 2
Enter the 4-digit key to search: 1234
Key 1234 found at index 1. Name: ABD

Menu:
1. Insert
2. Search
3. Display
```

```
2. Search
3. Display
4. Exit
Enter your choice: 2
Enter the 4-digit key to search: 1234
Key 1234 found at index 1. Name: ABD

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

Hash Table:
Index 0: Empty
Index 1: Key = 1234, Name = ABD
Index 2: Key = 9985, Name = KUL

Menu:
1. Insert
2. Search
3. Display
4. Exit
Enter your choice: 4
Exiting program.

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