VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Pranav Anantha Rao (1BM22CS201)

in partial fulfilment 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
Dec 2023- March 2024

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 Pranav Anantha Rao **(1BM22CS201)**, who is a bonafide student of **B. M. S. College of Engineering**. It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Data Structures Lab - **(23CS3PCDST) work** prescribed for the said degree.

Prof. Sneha S Bagalkot
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
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 size 10

int top = -1;
int stack[size];

void pop();
void push();
void display();

void main(){
   while(1){
      int choice;
      printf("Enter 1 for push \n");
      printf("Enter 2 for pop \n");
      printf("Enter 3 for display \n");
      printf("Enter choice: ");
      scanf("%d", &choice);
      switch(choice){
         case 1: push();
               break;
         case 2: pop();
               break;
         case 3: display();
               break;
         default: exit(0);
      }
   }

}

void push()
{
   int x;
   if (top == size - 1)
   {
      printf("Overflow\n");
   }
   else
   {
      printf("Enter the element to be added onto the stack: ");
      scanf("%d", &x);
      top = top + 1;
```

```
        stack[top] = x;
    }
}

void pop()
{
    if (top == -1)
    {
        printf("Underflow!!");
    }
    else
    {
        printf("Popped element: %d\n", stack[top]);
        top = top - 1;
    }
}

void display()
{
    if (top == -1)
    {
        printf("Underflow!!");
    }
    else
    {
        printf("Elements present in the stack: \n");
        for (int i = top; i >= 0; --i)
            printf("%d\n", stack[i]);
    }
}
```

Output:

```
Enter 1 for push
Enter 2 for pop
Enter 3 for display
Enter choice: 1
Enter the element to be added onto the stack: 5
Enter 1 for push
Enter 2 for pop
Enter 3 for display
Enter choice: 1
Enter the element to be added onto the stack: 6
Enter 1 for push
Enter 2 for pop
Enter 3 for display
Enter choice: 1
Enter the element to be added onto the stack: 7
Enter 1 for push
Enter 2 for pop
Enter 3 for display
Enter choice: 2
Popped element: 7
Enter 1 for push
Enter 2 for pop
Enter 3 for display
Enter choice: 3
Elements present in the stack:
6
5
```

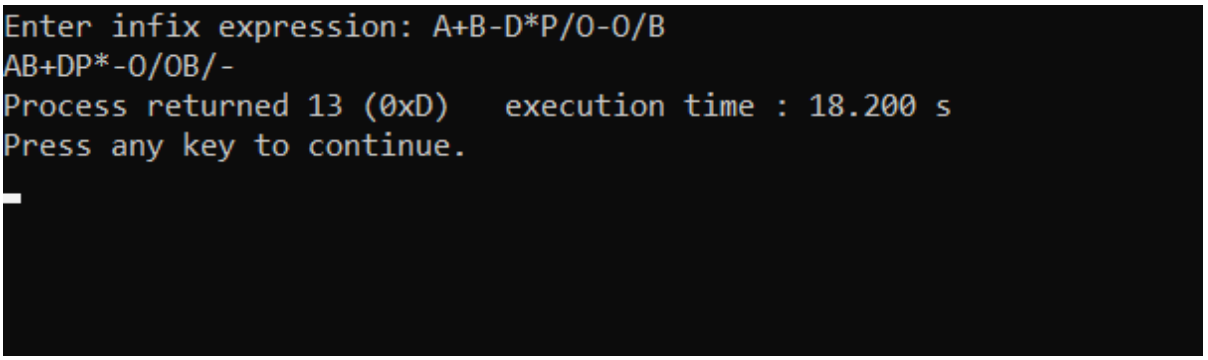
Lab Program 2:

WAP 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<ctype.h>
#define max 20
void push(char a);
char pop();
char stack[max],top =-1;
int pre(char a);

void main(){
   char
   infix[max],a;
   char post[max];
   printf("Enter infix expression:
   "); scanf("%s",infix);
   int j=0;
   push('(')
   ;
   for(int
      i=0;i<strlen(infix);i++){
      if(isalnum(infix[i])){
         post[j]=infix[i]
         ; j+=1;
      }
     else if((infix[i]=='+' || infix[i]=='-' || infix[i]=='/' || infix[i]=='*')){
            if(pre(infix[i])>pre(stack[top])){
               push(infix[i]);
            }
            else if(pre(infix[i])<=pre(stack[top])){

               while(1){
               a=pop();
               if(a=='('){
                  push(a)
                  ; break;
               }
               post[j]=a;
               j+=1;
            }
            push(infix[i]);
         }


   }
   while(top!=-1){ char y=pop()
```

```
if(y=='('){
        break;
       }
    post[j]=y;
    j+=1;


 }
 post[j]='\0';
 printf("%s",post);

 }

 void push(char
    a){
    if(top>max-1){
       printf("Stack overflow");
       exit(0);
    }
    else{
       ++top;
       stack[top]=a
       ;
    }
 }

 char pop(){
    if(top==-1)
    {
       printf("Stack
       underflow:"); exit(0);
    }
    else{
       return stack[top--];

    }
 }

 int pre(char a){
        if(a=='^'){
              return
                    3;
       }
       else if( a=='*' ||
          a=='/'){ return 2;
       }
       else if(a=='+' ||
          a=='-'){ return 1;
       }
       else{
          return 0;}
```

Output:


Enter infix expression: A+B-D*P/O-O/B
AB+DP*-O/OB/-
Process returned 13 (0xD)   execution time : 18.200 s
Press any key to continue.

Lab Program 3

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

```
#include<stdio.h>
#include<stdlib.h>
#define max 5
int queue[max];
int front=-1, rear=-1;

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

void
  main(){
  int n,m;
  printf("Enter\n1.To insert into queue\n2.To remove from queue\n3.To display
  queue:\n"); scanf("%d",&n);
  while(n!=3){
    switch(n){
      case 1: printf("Enter element to enter into queue:");
            scanf("%d",&m);
            enqueue(m);
            break;

      case 2: m=dequeue();
            printf("Removed element is
            %d\n",m); break;

      default: printf("Invalid input");
    }
    scanf("%d",&n);
    printf("\n");
  }
  if(n==3){
     display();
    }
}

void enqueue(int
  a){
  if(rear==max-1){
    printf("Queue overflow");

  }
  rear+=1;
  queue[rear]=a;
}
```

```
int dequeue(){
   if(rear==-1 || front==rear){
      printf("Queue underflow");
      exit(0);
   }
   else{
      front+=1;
      return queue[front];
   }
}

void display(){

   printf("Queue:\n");
   for(int i=front+1;i<=rear;i++){
      printf("%d\t",queue[i]);
      }
}
```

Output:

Queue Underflow

```
Enter
1.To insert into queue
2.To remove from queue
3.To display queue:
1
Enter element to enter into queue:10
1

Enter element to enter into queue:20
1

Enter element to enter into queue:30
1

Enter element to enter into queue:40
2

Removed element is 10
2

Removed element is 20
2

Removed element is 30
2

Removed element is 40
2

Queue underflow
Process returned 0 (0x0)   execution time : 18.446 s
Press any key to continue.
```

Queue Overflow

```
Enter
1.To insert into queue
2.To remove from queue
3.To display queue:
1
Enter element to enter into queue:1
1

Enter element to enter into queue:2
1

Enter element to enter into queue:3
1

Enter element to enter into queue:4
1

Enter element to enter into queue:5
1

Enter element to enter into queue:6
Queue overflow
```

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

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#define MAX 6
int cq[MAX];
int front = -1, rear = -1;

bool is_full() {
   return (rear + 1) % MAX == front;
}

bool is_empty() {
   return front == -1 && rear == -1;
```

```
}

void insert(int item)
   { if (is_full()) {
      printf("Overflow: Circular queue is full.\n");
   }

   if (is_empty()) {
      front = rear =
      0;
   } else {
      rear = (rear + 1) % MAX;
   }

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

int dequeue() {
   if (is_empty()) {
      printf("Underflow: Circular queue is
      empty.\n"); return -1;
   }
   int deletedItem =
   cq[front]; if (front == rear)
   {
      front = rear = -1;
   } else {
      front = (front + 1) % MAX;
   }

   printf("Dequeued: %d\n", deletedItem);
   return deletedItem;
}

int   main()
   {  int  n,
   ele; do {
      printf("\n1.       Insert\n2.       Delete\n3.
      Exit\n"); scanf("%d", &n);
      switch   (n)
         { case 1:
            printf("Enter the element to be inserted: ");
            scanf("%d", &ele);
            insert(ele);
            break;
         case 2:
            {
```

```
            int deletedItem = dequeue();
            if (deletedItem != -1) {
                printf("The element %d is removed.\n", deletedItem);
            }
        }
        break;
    case 3:
        printf("Thanks\n");
        break;
    default:
        printf("Please enter the right option.\n");
  }
} while (n != 3);

  return 0;
}
```

Output:

Circular Queue Underflow

```
1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 10
Enqueued: 10

1. Insert
2. Delete
3. Exit
2
Dequeued: 10
The element 10 is removed.

1. Insert
2. Delete
3. Exit
2
Underflow: Circular queue is empty.
```

Circular Queue Overflow

```
Enter the element to be inserted: 10
Enqueued: 10

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 20
Enqueued: 20

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 30
Enqueued: 30

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 40
Enqueued: 40

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 50
Enqueued: 50

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 60
Enqueued: 60

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 70
Overflow: Circular queue is full.
```

Lab Program 4

WAP to Implement Singly Linked List with following operations

a) Create a linked list.

b) Insertion of a node at first position, at any position and at end of list. Display the contents of the linked list.

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

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

Node* head = NULL;

void push();
void append();
void insert();
void display();

int main() {
    int choice;
    while (1) {
        printf("1. Insert at beginning\n");
        printf("2. Insert at end\n");
        printf("3. Insert at position\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                push();
                break;
            case 2:
                append();
                break;
            case 3:
                insert();
                break;
            case 4:
                display();
                break;
            default:
                printf("Exiting the program");
                return 0;
        }
    }
}
```

```
void push() {
   Node* temp = (Node*)malloc(sizeof(Node));
   int new_data;
   printf("Enter data in the new node: ");
   scanf("%d", &new_data);
   temp->data = new_data;
   temp->next = head;
   head = temp;
}

void append() {
   Node* temp = (Node*)malloc(sizeof(Node));
   int new_data;
   printf("Enter data in the new node: ");
   scanf("%d", &new_data);
   temp->data = new_data;
   temp->next = NULL;
   if (head == NULL) {
      head = temp;
      return;
   }
   Node* temp1 = head;
   while (temp1->next != NULL) {
      temp1 = temp1->next;
   }
   temp1->next = temp;
}

void insert() {
   Node* temp = (Node*)malloc(sizeof(Node));
   int new_data, pos;
   printf("Enter data in the new node: ");
   scanf("%d", &new_data);
   printf("Enter position of the new node: ");
   scanf("%d", &pos);
   temp->data = new_data;
   temp->next = NULL;
   if (pos == 0) {
      temp->next = head;
      head = temp;
      return;
   }
   Node* temp1 = head;
   while (pos--) {
      temp1 = temp1->next;
   }
   Node* temp2 = temp1->next;
   temp->next = temp2;
   temp1->next = temp;
}
```

```
void display() {
    Node* temp1 = head;
    while (temp1 != NULL) {
        printf("%d -> ", temp1->data);
        temp1 = temp1->next;
    }
    printf("NULL");
}
```

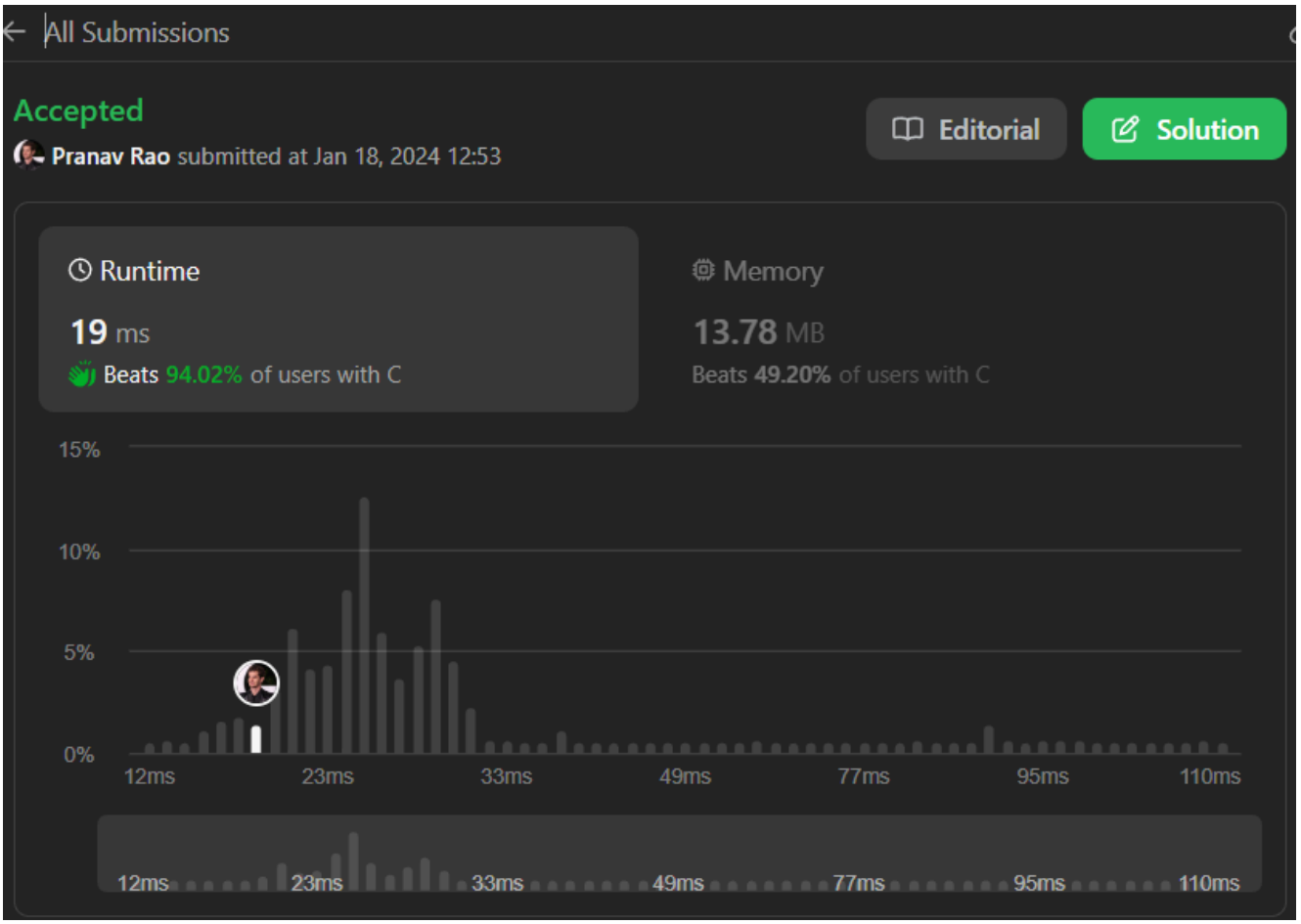
Output:

```
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 1
Enter data in the new node: 1
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 1
Enter data in the new node: 3
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 1
Enter data in the new node: 2
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 4
2 -> 3 -> 1 -> NULL1. Insert at beginning
```

Minstack Question [LeetCode]

```
#include<stdio.h>
#include<stdlib.h>
#define max 1000

typedef struct {
    int top;
    int st[max];
    int min[max];
} MinStack;

MinStack* minStackCreate() {
    MinStack* stack = (MinStack*)malloc(sizeof(MinStack));
    stack->top = -1;
    return stack;
}

void minStackPush(MinStack* obj, int val) {
    if(obj->top == max-1){
        printf("Stack Full\n");
        return;
    }
    obj->st[++obj->top] = val;

    if(obj->top > 0)
    {
        if(obj->min[obj->top - 1] < val)
            obj->min[obj->top] = obj->min[obj->top - 1];
        else
            obj->min[obj->top] = val;
    }
    else
        obj->min[obj->top] = val;
}

void minStackPop(MinStack* obj) {
    if(obj->top == -1)
    {
        printf("Stack empty\n");
        return;
    }
    else {
        obj->top -= 1;
    }
}

int minStackTop(MinStack* obj) {
    if(obj->top == -1)
    {
        printf("Stack empty\n");
        return -1;
```

```
    }
    return obj->st[obj->top];
}

int minStackGetMin(MinStack* obj) {
    if(obj->top == -1)
    {
        printf("min Stack empty\n");
        return -1;
    }
    return obj->min[obj->top];
}

void minStackFree(MinStack* obj) {
    free(obj);
}
```



Lab Program 5

WAP to Implement Singly Linked List with following operations

- **Create a linked list.**
- **Deletion of first element, specified element and last element in the list.**
- **Display the contents of the linked list.**

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

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

Node* head = NULL;

void pop();
void end_delete();
void delete_at_pos();
void display();
void append();

int main() {
    int choice;
    while (1) {
        printf("1. Insert at end\n");
        printf("2. Delete from beginning\n");
        printf("3. Delete from end\n");
        printf("4. Delete at position\n");
        printf("5. Display\n");
        printf("6. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                append();
                break;
            case 2:
                pop();
```

```
            break;
        case 3:
            end_delete();
            break;
        case 4:
            delete_at_pos();
            break;
        case 5:
            display();
            break;
        default:
            printf("Exiting the program");
            return 0;
      }
   }
}

void append() {
   Node* temp = (Node*)malloc(sizeof(Node));
   int new_data;
   printf("Enter data in the new node: ");
   scanf("%d", &new_data);
   temp->data = new_data;
   temp->next = NULL;
   if (head == NULL) {
      head = temp;
      return;
   }
   Node* temp1 = head;
   while (temp1->next != NULL) {
      temp1 = temp1->next;
   }
   temp1->next = temp;
}

void pop() {
   if(head == NULL){
      printf("Linked List is empty");
      return;
```

```
    }
    head = head->next;
}

void end_delete() {
    if(head == NULL){
        printf("Linked List is empty");
        return;
    }
    if(head->next == NULL){
        head = NULL;
        return;
    }
    Node* temp = head;
    Node* temp1 = head;
    temp = temp->next;
    while(temp->next != NULL){
        temp = temp->next;
        temp1 = temp1->next;
    }
    temp1->next = NULL;
}

void delete_at_pos() {
    int pos;
    printf("Enter position: ");
    scanf("%d", &pos);
    Node* temp = head;
    while(--pos){
        temp = temp->next;
        if(temp->next = NULL){
            printf("Not enough elements");
        }
    }
    Node* temp1 = temp;
    temp1 = temp1->next;
    if(temp1->next == NULL){
        printf("Not enough elements");
    }
```

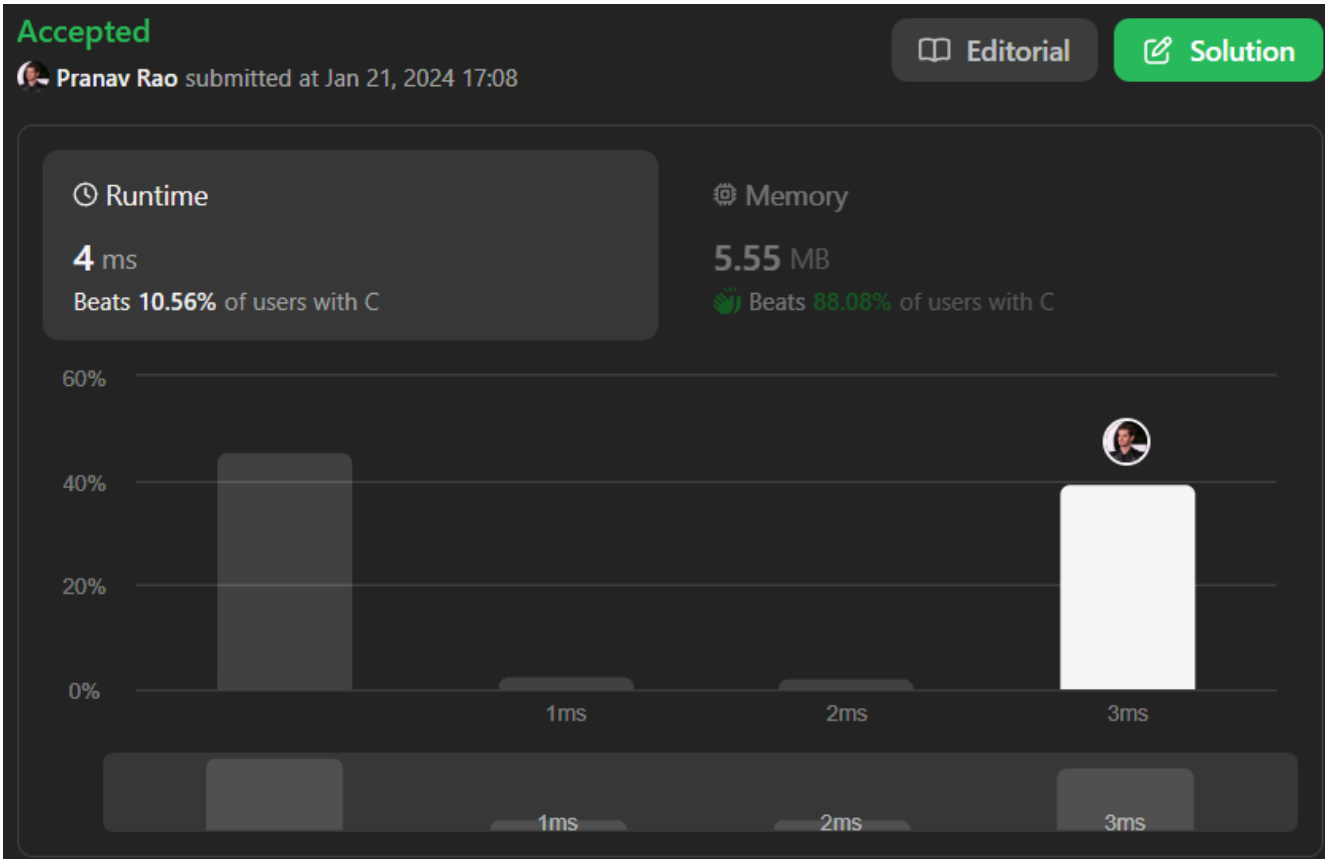
```
    temp1 = temp1->next;
    temp->next = temp1;
}

void display() {
    Node* temp1 = head;
    while (temp1 != NULL) {
        printf("%d -> ", temp1->data);
        temp1 = temp1->next;
    }
    printf("NULL\n");
}
```

Output:

```
Enter choice: 1
Enter data in the new node: 4
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete at position
5. Display
6. Exit
Enter choice: 1
Enter data in the new node: 5
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete at position
5. Display
6. Exit
Enter choice: 2
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete at position
5. Display
6. Exit
Enter choice: 3
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete at position
5. Display
6. Exit
Enter choice: 4
Enter position: 1
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete at position
5. Display
6. Exit
Enter choice: 5
2 -> 4 -> NULL
```

Reverse Linked List Question [LeetCode]

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
    if(head==NULL) return head;
        if(head->next==NULL) return head;
        struct ListNode* start = head;
        int r1 = right;
        int l1 = left;
        struct ListNode* prevStart = NULL;
        while(--l1){
            prevStart = start;
            start = start->next;
        }
        struct ListNode* temp1 = NULL;
        struct ListNode* temp2 = NULL;
        struct ListNode* temp3 = start;
        r1 = right;
        while(r1-- > left - 1){
            temp1 = temp2;
            temp2 = temp3;
            temp3 = temp3->next;
            temp2->next = temp1;
        }
        if(left > 1){
            prevStart->next = temp2;
        } else {
            head = temp2;
        }
        start->next = temp3;
        return head;
}
```


Accepted
Pranav Rao submitted at Jan 21, 2024 17:08
Editorial
Solution
Runtime
4 ms
Beats 10.56% of users with C
Memory
5.55 MB
Beats 88.08% of users with C
60%
40%
20%
0%
1ms
2ms
3ms
1ms
2ms
3ms

Lab Program 6:

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

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

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

struct Node* head1 = NULL;
struct Node* head2 = NULL;

void insertAtEnd(struct Node* *head) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    int newVal;
    scanf("%d", &newVal);
    newNode->data = newVal;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* tail = *head;
    while (tail->next != NULL) {
        tail = tail->next;
    }
    tail->next = newNode;
}

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

void sortList(struct Node* *head) {
    struct Node* current = *head;
    struct Node* temp = NULL;
    while (current != NULL) {
        temp = current->next;
```

```
        while (temp != NULL) {
            if (current->data > temp->data) {
                int swap = current->data;
                current->data = temp->data;
                temp->data = swap;
            }
            temp = temp->next;
        }
        current = current->next;
    }
}

void 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;
    }

    *head = prev;
}

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

int main() {
    int l1, l2;

    printf("Enter length of First linked list: ");
    scanf("%d", &l1);

    printf("Enter length of Second linked list: ");
    scanf("%d", &l2);

    printf("Enter first linked list: ");
    for(int i = 0; i < l1; i++){
        insertAtEnd(&head1);
    }
```

```
    printf("Enter second linked list: ");
    for(int i = 0; i < l1; i++){
        insertAtEnd(&head2);
    }

    printf("First linked list: ");
    display(head1);

    printf("\nSecond linked list: ");
    display(head2);

    sortList(&head1);
    printf("\nSorted linked list: ");
    display(head1);

    reverseList(&head2);
    printf("\nReversed linked list: ");
    display(head2);

    concatenate(head1, head2);
    printf("\nConcatenated linked list: ");
    display(head1);

    return 0;
}
```

Output:

```
Enter length of First linked list: 5
Enter length of Second linked list: 5
Enter first linked list: 1
2
3
4
5
Enter second linked list: 2
3
4
1
5
First linked list: 1 -> 2 -> 3 -> 4 -> 5 -> NULL
Second linked list: 2 -> 3 -> 4 -> 1 -> 5 -> NULL
Sorted linked list: 1 -> 2 -> 3 -> 4 -> 5 -> NULL
Reversed linked list: 5 -> 1 -> 4 -> 3 -> 2 -> NULL
Concatenated linked list: 1 -> 2 -> 3 -> 4 -> 5 -> 5 -> 1 -> 4 -> 3 -> 2 -> NULL
```

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

Stack Implementation

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

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

struct Node* top = NULL;

void push(int element) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = element;
    newNode->next = top;
    top = newNode;
}

void pop() {
    if (top == NULL) {
        printf("Stack is empty.\n");
        return;
    }
    struct Node* temp = top;
    top = top->next;
    free(temp);
}

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

int main() {
    int ch;
    while(1){
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
```

```
        printf("Enter choice");
        scanf("%d", &ch);
        switch(ch){
           case 1: {int ele;
                    printf("Enter element: ");
                    scanf("%d", &ele);
                    push(ele);
                    break;}
           case 2: pop();
                    break;
           case 3: display();
                    break;
           default: return 0;
        }
     }
     return 0;
}
```

Output:

```
1. Push
2. Pop
3. Display
4. Exit
Enter choice1
Enter element: 5
1. Push
2. Pop
3. Display
4. Exit
Enter choice1
Enter element: 4
1. Push
2. Pop
3. Display
4. Exit
Enter choice2
1. Push
2. Pop
3. Display
4. Exit
Enter choice3
Stack:
5
```

Queue Implementation

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

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

struct Node* front = NULL;
struct Node* rear = NULL;

void enqueue(int element) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = element;
    newNode->next = NULL;

    if (rear == NULL) {
        front = rear = newNode;
        return;
    }
    rear->next = newNode;
    rear = newNode;
}

void dequeue() {
    if (front == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    struct Node* temp = front;
    front = front->next;

    if (front == NULL)
        rear = NULL;

    free(temp);
}

int getFront() {
    if (front == NULL) {
        printf("Queue is empty.\n");
        return -1;
    }
    return front->data;
}

void display() {
    if (front == NULL) {
        printf("Queue is empty.\n");
```

```
        return;
    }
    struct Node* temp = front;
    printf("Queue:\n");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int ch;
    while(1){
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter choice");
        scanf("%d", &ch);
        switch(ch){
            case 1: {int ele;
                    printf("Enter element: ");
                    scanf("%d", &ele);
                    enqueue(ele);
                    break;}
            case 2: dequeue();
                    break;
            case 3: display();
                    break;
            default: return 0;
        }
    }

    return 0;
}
```

Lab Programs 7:

WAP to Implement doubly link list with primitive operations

- **Create a doubly linked list.**
- **Insert a new node to the left of the node.**
- **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* prev;
   struct Node* next;
};

struct Node* createNode() {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   int data;
   printf("Enter data in node");
   scanf("%d", &data);
   newNode->data = data;
   newNode->prev = NULL;
   newNode->next = NULL;
   return newNode;
}

void insertNode(struct Node* *head) {
   int pos;
   printf("Enter position of new node");
   scanf("%d", &pos);
   struct Node* newNode = createNode();
   struct Node* temp = (*head);
   while(--pos){
      if(temp->next != NULL)
         temp = temp->next;
      else{
         printf("List too short");
         return;
      }
   }
   if (temp->next == NULL) {
      newNode->next = *head;
      (*head)->prev = newNode;
      (*head) = newNode;
   } else {
      temp->prev->next = newNode;
      newNode->prev = temp->prev;
      temp->prev = newNode;
```

```
        newNode->next = temp;
    }
}

void deleteNode(struct Node* *head) {
    int data;
    printf("Enter data in node to be deleted");
    scanf("%d", &data);
    struct Node* current = *head;
    while (current != NULL) {
        if (current->data == data) {
            if (current->prev != NULL) {
                current->prev->next = current->next;
            } else {
                *head = current->next;
            }
            if (current->next != NULL) {
                current->next->prev = current->prev;
            }
            free(current);
            return;
        }
        current = current->next;
    }
    printf("Node with value %d not found", data);
}

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

int main() {
    int choice;
    struct Node* head = NULL;
    while (1) {
        printf("1. Create a list\n");
        printf("2. Insert a node\n");
        printf("3. Delete a node\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                head = createNode();
```

```
            break;
        case 2:
            insertNode(&head);
            break;
        case 3:
            deleteNode(&head);
            break;
        case 4:
            display(head);
            break;
        default:
            printf("Exiting the program");
            return 0;
    }
  }
}
```

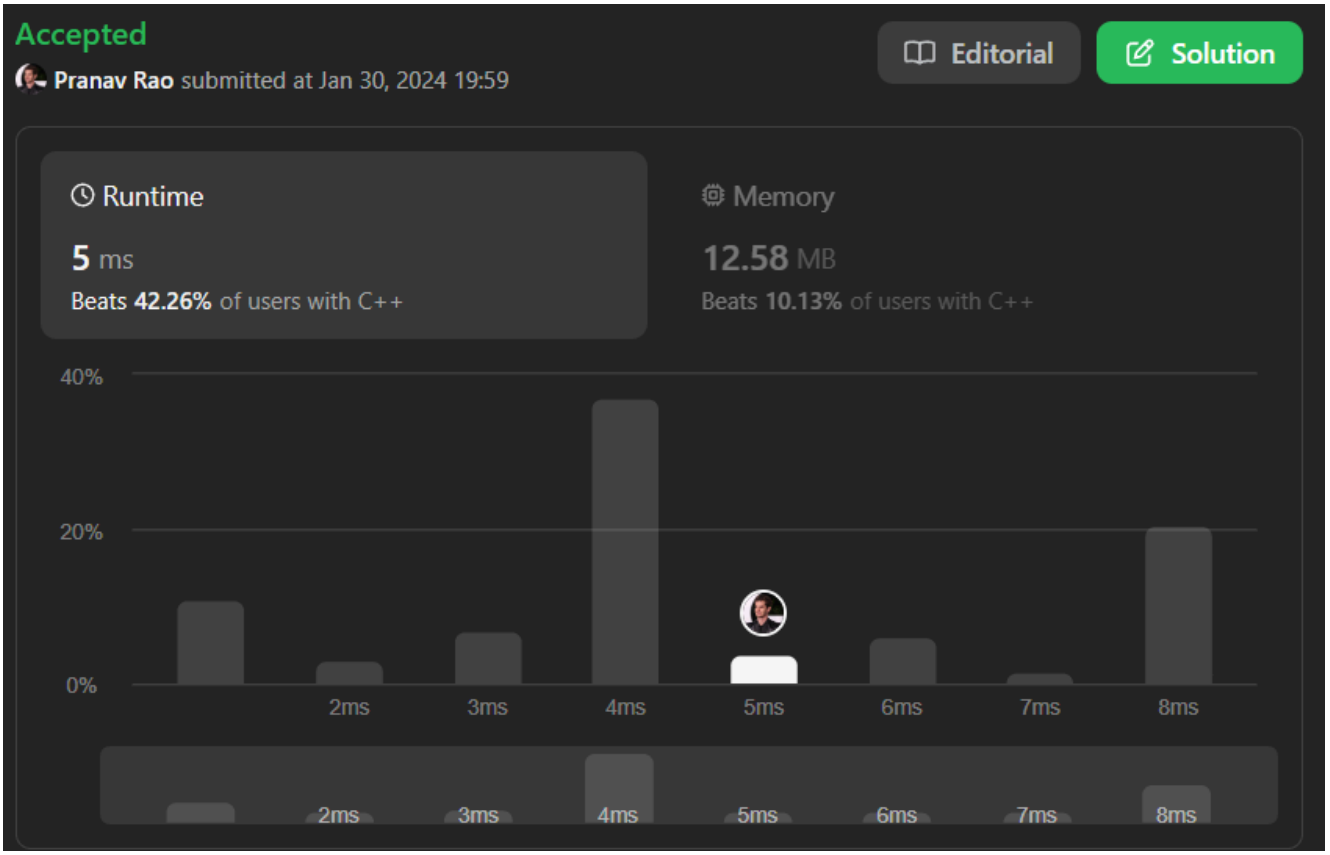
Output:

```
1. Create a list
2. Insert a node
3. Delete a node
4. Display
5. Exit
Enter choice: 1
Enter data in node1
1. Create a list
2. Insert a node
3. Delete a node
4. Display
5. Exit
Enter choice: 2
Enter position of new node1
Enter data in node2
1. Create a list
2. Insert a node
3. Delete a node
4. Display
5. Exit
Enter choice: 3
Enter data in node to be deleted2
1. Create a list
2. Insert a node
3. Delete a node
4. Display
5. Exit
Enter choice: 4
Doubly Linked List: 1 -> NULL
```

Split Linked List into parts [LeetCode]

```
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize)
   { struct ListNode* ptr=head;
   *returnSize=k;
   int count=0;

   while(ptr!=NULL)
      { count++;
      ptr=ptr->next;
   }

   int nums=count/k,a=count%k;
   struct ListNode **L=(struct ListNode**)calloc(k,sizeof(struct ListNode*));
   ptr=head;
   for(int i=0;i<k;i++){
      L[i] = ptr;

      int segmentSize = nums + (a-- > 0 ? 1 : 0);
      for (int j = 1; j < segmentSize; j++) {
         ptr = ptr->next;
```

```
        }

        if (ptr != NULL) {
            struct ListNode* next =
            ptr->next; ptr->next = NULL;
            ptr = next;
        }
    }
    return L;
```




```
}
```

Lab Program 8:

Write a program

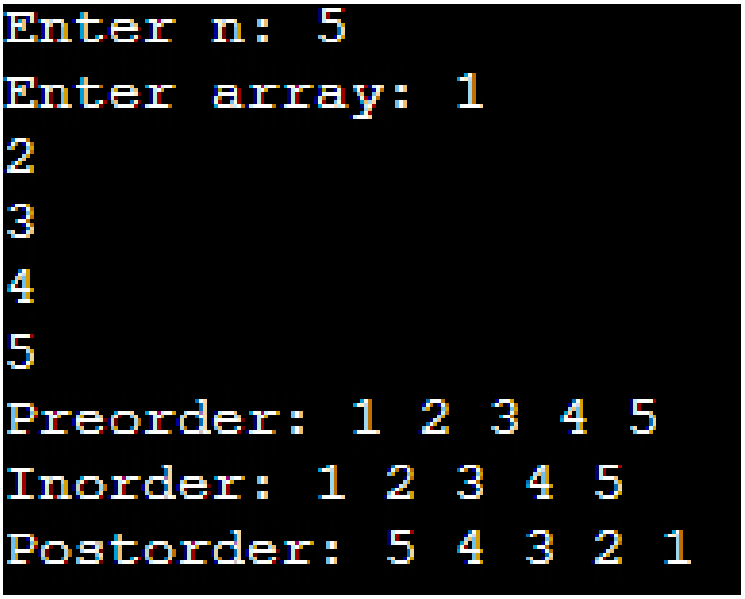
- **To construct a binary Search tree.**
- **To traverse the tree using all the methods i.e., in-order, preorder and post order**
- **To display the elements in the tree**

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

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

Node* root = NULL;

void insert(int a);
void inorder();
void preorder();
void postorder();

void main(){
   int n;
   printf("Enter n: ");
   scanf("%d", &n);
   int a[n];
   printf("Enter array: ");
   for(int i = 0; i < n; i++){
      scanf("%d", &a[i]);
      insert(a[i]);
   }
   printf("Preorder: ");
   preorder(root);
   printf("\nInorder: ");
   inorder(root);
   printf("\nPostorder: ");
   postorder(root);
}

void insert(int a){
```

```
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->left = NULL;
    newNode->right = NULL;
    newNode->data = a;
    Node* temp = root;
    Node* pare = NULL;
    while(temp != NULL){
        pare = temp;
        if(a > temp->data){
            temp = temp->right;
        }
        else if(a < temp->data){
            temp = temp->left;
        }
        else{
            printf("Invalid data");
            return;
        }
    }
    if(pare == NULL){
        root = newNode;
    }
    else if(a < pare->data){
        pare->left = newNode;
    }
    else{
        pare->right = newNode;
    }
}

void inorder(Node* temp){
    if(temp != NULL){
        inorder(temp->left);
        printf("%d ", temp->data);
        inorder(temp->right);
    }
}

void preorder(Node* temp){
    if(temp != NULL){
        printf("%d ", temp->data);
        preorder(temp->left);
```

```
        preorder(temp->right);
    }
}

void postorder(Node* temp){
    if(temp != NULL){
        postorder(temp->left);
        postorder(temp->right);
        printf("%d ", temp->data);
    }
}
```

Output:



Rotate List [LeetCode]

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if(head == NULL || head->next == NULL){
            return head;
        }
        int n = 0;
        ListNode* len = head;
        while(len->next != NULL){
            n++;
            len = len->next;
        }
        n++;
        k = k % n;
        while(k--){
            ListNode* prevTail = head;
            while(prevTail->next->next != NULL){
                prevTail = prevTail->next;
            }
            ListNode* tail = prevTail->next;
            tail->next = head;
            prevTail->next = NULL;
            head = tail;
        }
        return head;
    }
};
```

Lab Program 9:

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

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

#define MAX_NODES 100
#define MAX_EDGES 100

int graph[MAX_NODES][MAX_NODES];
int visited[MAX_NODES];
int queue[MAX_NODES];
int front = -1, rear = -1;

void BFS(int start, int n) {
   visited[start] = 1;
   queue[++rear] = start;

   while(front != rear) {
      int current = queue[++front];
      printf("%d ", current);

      for(int i = 0; i < n; i++) {
         if(graph[current][i] == 1 && !visited[i]) {
            visited[i] = 1;
            queue[++rear] = i;
         }
      }
   }
}

int main() {
   int n, m;
   printf("Enter the number of nodes and edges: ");
   scanf("%d %d", &n, &m);

   printf("Enter the edges:\n");
   for(int i = 0; i < m; i++) {
      int a, b;
      scanf("%d %d", &a, &b);
      graph[a][b] = 1;
      graph[b][a] = 1;
   }

   int start;
   printf("Enter the starting node: ");
   scanf("%d", &start);
```

```
    printf("BFS traversal: ");
    BFS(start, n);

    return 0;
}
```

Output:

```
Enter the number of nodes and edges: 5
4
Enter the edges:
1
2
2
3
3
4
4
5
Enter the starting node: 1
BFS traversal: 1 2 3 4
```

DFS:

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

int arr[20][20];
int visited[20];
int queue[20];
int front = -1;
int rear = -1;

void bfs(int start, int n) {
    visited[start] = 1;
    queue[++rear] = start;

    while(front != rear) {
        int current = queue[++front];
        printf("%d ", current);

        for(int i = 0; i < n; i++) {
            if(arr[current][i] == 1 && !visited[i]) {
                visited[i] = 1;
                queue[++rear] = i;
            }
        }
    }
}

void main(){
    int n, m;
    printf("Enter the number of nodes and edges: ");
```

```
    scanf("%d %d", &n, &m);

    printf("Enter the edges:\n");
    for(int i = 0; i < m; i++) {
        int a, b;
        scanf("%d %d", &a, &b);
        arr[a][b] = 1;
        arr[b][a] = 1;
    }

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

    printf("BFS traversal: ");
    bfs(start, n);
}
```

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

int arr[20][20];
int visited[20];

void dfs(int start, int n) {
    visited[start] = 1;
    for(int i = 0; i < n; i++) {
        if(arr[start][i] == 1 && !visited[i]) {
            dfs(i, n);
        }
    }
}

int isConnected(int n) {
    dfs(0, n);

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

    return 1;
}

int main() {
    int n, m;
    printf("Enter the number of nodes and edges: ");
    scanf("%d %d", &n, &m);
```

```
    printf("Enter the edges:\n");
    for(int i = 0; i < m; i++) {
        int a, b;
        scanf("%d %d", &a, &b);
        arr[a][b] = 1;
        arr[b][a] = 1;
    }

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

    return 0;
}
```

```
Enter the number of nodes and edges: 5
4
Enter the edges:
1
2
2
3
4
3
5
4
The graph is not connected.
```

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.

```
#include <stdio.h>

#define TABLE_SIZE 10

int hash_table[TABLE_SIZE] = {0};

void insert(int key) {
   int i = 0;
   int hkey = key % TABLE_SIZE;
   int index = hkey;

   while (hash_table[index] != 0) {
      i++;
      index = (hkey + i) % TABLE_SIZE;
      if (i == TABLE_SIZE) {
         printf("Hash table is full!\n");
         return;
      }
   }

   hash_table[index] = key;
}

void search(int key) {
   int i = 0;
   int hkey = key % TABLE_SIZE;
   int index = hkey;

   while (hash_table[index] != key) {
      i++;
      index = (hkey + i) % TABLE_SIZE;
      if (i == TABLE_SIZE) {
         printf("Element not found in hash table!\n");
         return;
      }
   }

   printf("Element found at index %d\n", index);
}

int main() {
   int choice;
   int key;
```

```
    while(1){
        printf("1. Insert\n");
        printf("2. Search\n");
        printf("3. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1: printf("Enter Key: ");
                    scanf("%d", &key);
                    insert(key);
                    break;
            case 2: printf("Enter Key: ");
                    scanf("%d", &key);
                    search(key);
                    break;
            default: return 0;
        }
    }
    return 0;
}
```

Output:

```
1. Insert
2. Search
3. Exit
Enter choice: 1
Enter Key: 15
1. Insert
2. Search
3. Exit
Enter choice: 1
Enter Key: 25
1. Insert
2. Search
3. Exit
Enter choice: 2
Enter Key: 15
Element found at index 5
1. Insert
2. Search
3. Exit
Enter choice: 2
Enter Key: 25
Element found at index 6
```