VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Aryan Navlani (1BM23CS055)

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 Aryan Navlani**(1BM23CS055)**, 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.

Sneha S	**Dr. Kavitha Sooda**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Course outcomes:

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

Lab program 1:

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

Code-

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

Enter your choice \n");

scanf("%d", &ch);

switch(ch)

{

case 1:

        Push();

        break;


case 2:

        item = pop();

        if(item != -1){

            printf("Popper element = %d\n", item);

        }

        break;

case 3:
```

```
        Display();
        break;
case 4:
        exit(0);
}
}
getch();
}
void Push()
{
if(TOP == MAX-1){
printf("STACK OVERFLOW \n"); return;
}
printf("Enter element to be pushed \n");
scanf("%d", &item);
TOP = TOP+1;
s[TOP] = item;
}
int pop(){
if(TOP == -1){
    return -1;
}
item = s[TOP];
TOP = TOP - 1;
return item;
}
void Display()
{
```

```
if(TOP == -1){
    printf("STACK IS EMPTY\n"); return ;
    }
    printf("STACK CONTENTS\n"); for(i
    = TOP; i >= 0; i--){
        printf("%d\n", s[i]);
    }
}
```

Output-

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

 1 PUSH
 2 POP
 3 DISPLAY
 4 EXIT
 Enter your choice
3
STACK CONTENTS
4

 1 PUSH
 2 POP
 3 DISPLAY
 4 EXIT
 Enter your choice
2
Popper element = 4

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

 1 PUSH
 2 POP
 3 DISPLAY
 4 EXIT
 Enter your choice

4

Process returned 0 (0x0)   execution time : 17.587 s
Press any key to continue.
```

Lab program 2;

Write a program the converts infix expression into postfix expression.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int index = 0, pos = 0, top = -1, length;
char symbol, temp, infix[20],postfix[20],stack[20];
void infixtopostfix();
void push(char symbol);
char pop();
int pred(char symbol);
void main()
{
clrsrc();
printf("Enter infix expression : \n ");
scanf("%s", infix);
infixtopostfix();
printf("\n Infix expression \n %s", infix); printf("\n
Postfix expression :\n %s", postfix); getch();
}
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'^':
          while(pred(stack[top]) >= pred(symbol)){
            temp = pop();
            prefix[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 symbol;  symbol
    = stack[top]; top =
    top-1;
    return synbol;
}
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-

```
Enter infix expression:
A*B+C/D-E

Infix expression: A*B+C/D-E
Postfix expression: AB*CD/+E-#

Process returned 32 (0x20)   execution time : 33.517 s
Press any key to continue.
```

Lab Program 3;

Write C code for queue implementation that includes dequeue and enqueue methods.

```
#include <stdio.h>

#include <stdlib.h>

#define SIZE 5

int queue[SIZE], front = -1, rear = -1; int

isFull() {

    return rear == SIZE - 1;

}

int isEmpty() {

    return front == -1 || front > rear;

}

void enqueue(int value) { if

    (isFull()) {

        printf("Queue is full! Cannot insert %d\n", value);

    } else {

        if (front == -1) {

            front = 0;

        }

        rear++;

        queue[rear] = value;

        printf("Inserted %d into the queue.\n", value);

    }

}

void dequeue() {

    if (isEmpty()) {

        printf("Queue is empty! Cannot dequeue.\n");

    } else {

        printf("Deleted %d from the queue.\n", queue[front]); front++;
```

```
        if (front > rear) {
            front = rear = -1;
        }
    }
}
void display() {
    if (isEmpty()) {
        printf("Queue is empty!\n");
    } else {
        printf("Queue elements: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}
int main() {
    int choice, value;
    while (1) {
        printf("\nQueue Menu:\n");
        printf("1. Insert (Enqueue)\n");
        printf("2. Delete (Dequeue)\n");
        printf("3. Display Queue\n");
        printf("4. Exit\n"); printf("Enter
        your choice: "); scanf("%d",
        &choice);
        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
```

```
                scanf("%d", &value);
                enqueue(value); break;
            case 2:
                dequeue();
                break;
            case 3:
                display(); break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

Output-

```
Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 1
Enter the value to insert: 2
Inserted 2 into the queue.

Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 1
Enter the value to insert: 3
Inserted 3 into the queue.

Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 1
Enter the value to insert: 4
Inserted 4 into the queue.

Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 2
Deleted 2 from the queue.

Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 3
Queue elements: 3 4

Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 4

Process returned 0 (0x0)   execution time : 15.008 s
Press any key to continue.
```

Lab Program 4:

Implementation of circular queue.

```
#include <stdio.h>
#define MAX_SIZE 5 int
queue[MAX_SIZE]; int
front = -1, rear = -1; int
isFull()
{
    return (rear + 1) % MAX_SIZE == front;
}
int isEmpty()
{
    return front == -1;
}
void enqueue(int data)
{
    if (isFull()) {
        printf("Queue overflow\n");
        return;
    }
    if (front == -1) {
        front = 0;
    }
    rear = (rear + 1) % MAX_SIZE;
    queue[rear] = data;
    printf("Element %d inserted\n", data);
}
int dequeue()
```

```
    if (isEmpty()) {
        printf("Queue underflow\n");
        return -1;
    }
    int data = queue[front]; if
    (front == rear) {
        front = rear = -1;
    }
    else {
        front = (front + 1) % MAX_SIZE;
    }
    return data;
}
void display()
{
    if (isEmpty()) {
        printf("Queue is empty\n"); return;
    }
    printf("Queue elements: "); int
    i = front;
    while (i != rear) { printf("%d
        ", queue[i]);  i = (i + 1) %
        MAX_SIZE;
    }
    printf("%d\n", queue[rear]);
}
void main()
{
```

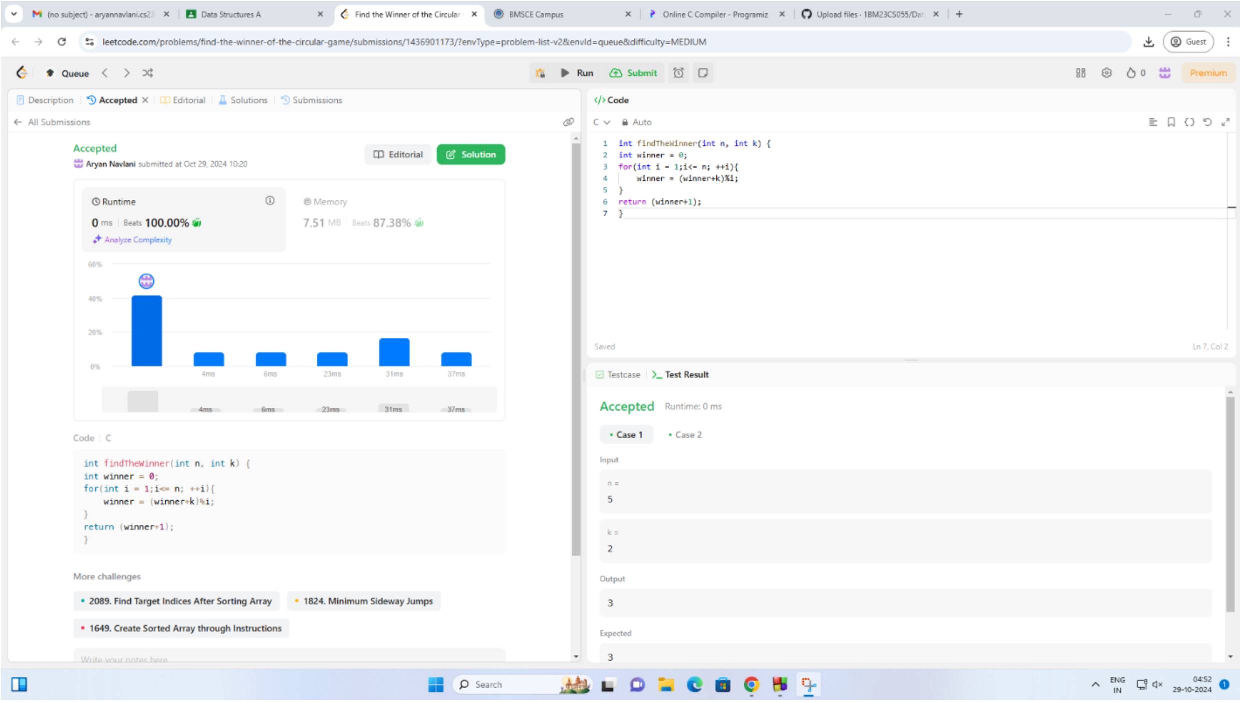
```
    int choice, values; printf("Queue
    Menu: \n");
    printf("1.Insert(Enqueue): \n");
    printf("2.Delete(Deque): \n");
    printf("3.Display Queue \n");
    printf("4.EXIT \n");
    printf("Enter the number choice : ");
    scanf("%d", &choice);  switch(choice)
    {
      case 1:
        printf("Enter the value to insert");
        scanf("%d", &values); enque(values);
        break;
      case 2:
        dequeue();
        break;
      case 3:
        display();
        break;
      case 4:
        exit(0);
      default:
        printf("Invalid Choice! please try again \n");
    }
}
```

```
Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. EXIT
Enter your choice: 1
Enter the value to insert: 1
Element 1 inserted
Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. EXIT
Enter your choice: 1
Enter the value to insert: 2
Element 2 inserted
Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. EXIT
Enter your choice: 1
Enter the value to insert: 3
Element 3 inserted
Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. EXIT
Enter your choice: 1
Enter the value to insert: 4
Element 4 inserted
Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. EXIT
Enter your choice: 1
Enter the value to insert: 5
Element 5 inserted
Queue Menu:
```

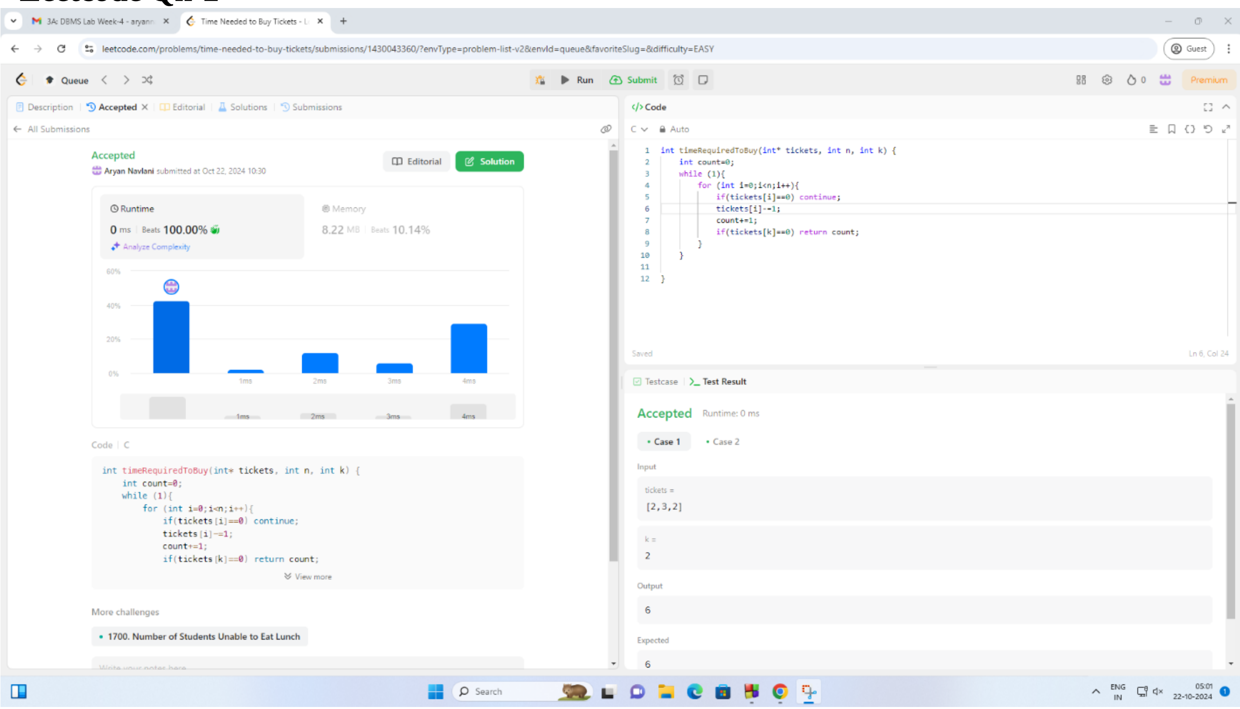
```
Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. EXIT
Enter your choice: 2
Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. EXIT
Enter your choice: 2
Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. EXIT
Enter your choice: 3
Queue elements: 3 4 5
Queue Menu:
1. Insert (Enqueue)
2. Delete (Dequeue)
3. Display Queue
4. EXIT
Enter your choice: 4

Process returned 0 (0x0)   execution time : 55.100 s
Press any key to continue.
```

Leetcode Question - 1



Leetcode Qn-2



Lab program 5

Singly linked list Implementation.

```
#include<stdio.h>

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

node *ptr,*new,*curr1,*start = NULL;

void create(){
    new = (node*)malloc(sizeof(node)); char
    ch;
    do{
        printf("Enter value\n"); scanf("%d",
        &new1->data); if(start == NULL){
            start = new; curr
            = new;
        }else{
            curr->link = new1; curr
            = new1;
            printf("Do you want to add another element(Y/N)");
            scanf("%c", &ch);
        }
    }while(ch == 'y'||ch == 'Y');
```

```
}
void Display(){ if(start
    == NULL){
        printf("Linked list is empty");
    }
    printf("Elements in the list");
    tenp = start;
    while(temp != NULL){
        printf("%d", temp->data);
        temp = temp->link;
    }
}
void insert_beg(){
    new1 = (node*)malloc(sizeof(node));
    printf("Enter element"); scanf("%d",
    &new1 -> data); if(start == NULL){
        start = new1;
        new1->link = NULL;
        return;
    }
    new1-> = start;
    start = new1;
}
void insert_post(int pos){
    new = (node*)malloc(sizeof(node));
    printf("Enter element"); scanf("%d",
    &new1->data);
    if(pos == 1){
```

```
        new1->link = start;
        start = new1; return;
    }
    while(temp != NULL && i < pos){ temp
        = temp->linl;
        pos++;
    }
    if(temp == NULL){
        printf("Enter posistion greate than number of element\n");
    }
    new1->link - temp-.link;
    temp->link = new1;
}
void delete_first(){
    if(start == NULL){
        printf("Linked List Empty \n"); return;
    }
    node  temp  =  startl
    start   =   start->link;
    free(temp);
}
void main(){
    while(1){
        printf("1 Create \n 2.Display  \n3.Insert_Beg \n 4.Insert_pos  \n   5.Deletion
\n6.Exit");
        int c;
        print("Enter option");
        scanf("%d", &c);
```

```
        switch(c){
           case 1:
              create();
              break;
           case 2:
              Display();
              break;
           case3:
              insert_beg();
              break;
           case 4:
              int pos;
              printf("Enter position");
              scanf("%d", &pos);
              insert_post(pos);
              break;
           case 5:
              delete_first();
              break;
           default:
              exit(0);
        }

     }
  }
```

Output-

```
"C:\Users\AYUSH ADITYA\Doc
1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Delete First Node
6. Exit
Enter option: 1
Enter value
5
Do you want to add another element (Y/N)? Y
Enter value
2
Do you want to add another element (Y/N)? Y
Enter value
4
Do you want to add another element (Y/N)? N
1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Delete First Node
6. Exit
Enter option: 2
Elements in the list: 4
1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Delete First Node
6. Exit
Enter option: 3
Enter element: 1
1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Delete First Node
6. Exit
Enter option: 4
Enter position: 3
Enter element: 1

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Delete First Node
6. Exit
Enter option: 4
Enter position: 3
Enter element: 1
1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Delete First Node
6. Exit
Enter option: 5
1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Delete First Node
6. Exit
Enter option: 6

Process returned 0 (0x0)   execution time : 48.243 s
Press any key to continue.
```

Lab program 6

Stack and queue implementation using linked list.

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

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

// Stack implementation
struct Stack {
    struct Node* top;
};

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

int pop(struct Stack* stack) { if
    (isStackEmpty(stack)) {
```

```
        printf("Stack is empty\n"); return -
        1;
    }
    struct Node* temp = stack->top;
    int data = temp->data;
    stack->top = temp->next;
    free(temp);
    return data;
}

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

struct Queue {
    struct Node* front;
    struct Node* rear;
};
int isQueueEmpty(struct Queue* queue) {
    return queue->front == NULL;
}

void enqueue(struct Queue* queue, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
```

```
    newNode->next = NULL;
    if (isQueueEmpty(queue)) {
        queue->front = queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
    printf("Enqueued %d to queue\n", data);
}

int dequeue(struct Queue* queue) { if
    (isQueueEmpty(queue)) {
        printf("Queue is empty\n"); return -
        1;
    }
    struct Node* temp = queue->front; int
    data = temp->data;
    queue->front = temp->next; free(temp);

    if (queue->front == NULL) {
        queue->rear = NULL;
    }
    return data;
}

int peekQueue(struct Queue* queue) { if
    (isQueueEmpty(queue)) {
        printf("Queue is empty\n");
```

```
        return -1;
    }
    return queue->front->data;
}

int main() {
    struct   Stack   stack;
    stack.top  =  NULL;
    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);
    printf("Top of stack: %d\n", peek(&stack));
    printf("Popped from stack: %d\n", pop(&stack));
    printf("Popped from stack: %d\n", pop(&stack));
    printf("Popped from stack: %d\n", pop(&stack));
    printf("Popped from stack: %d\n", pop(&stack)); struct
    Queue queue;
    queue.front = queue.rear = NULL;
    enqueue(&queue, 10);
    enqueue(&queue, 20);
    enqueue(&queue, 30);
    printf("Front of queue: %d\n", peekQueue(&queue);
    printf("Dequeued from queue: %d\n", dequeue(&queue));
    printf("Dequeued from queue: %d\n", dequeue(&queue));
    printf("Dequeued from queue: %d\n", dequeue(&queue));
    printf("Dequeued from queue: %d\n", dequeue(&queue)); // Queue is empty now return 0;
}
```

Output-

```
Pushed 10 to stack
Pushed 20 to stack
Pushed 30 to stack
Top of stack: 30
Popped from stack: 30
Popped from stack: 20
Popped from stack: 10
Stack is empty
Popped from stack: -1
Enqueued 10 to queue
Enqueued 20 to queue
Enqueued 30 to queue
Front of queue: 10
Dequeued from queue: 10
Dequeued from queue: 20
Dequeued from queue: 30
Queue is empty
Dequeued from queue: -1

Process returned 0 (0x0)   execution time : 2.584 s
Press any key to continue.
```

Lab program 7

Implementation of circular 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 insertAtEnd(struct Node** head, int data) { struct

    Node* newNode = createNode(data);


    if (*head == NULL) {

        newNode->next = newNode; // Point to itself if the list is empty

        *head = newNode;

    } else {

        struct Node* temp = *head;

        // Traverse to the last node (which points back to head) while

        (temp->next != *head) {

            temp = temp->next;

        }

        temp->next = newNode;

        newNode->next = *head; // Make it circular

    }
```

```
}
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        newNode->next = newNode; // Point to itself if the list is empty
        *head = newNode;
    } else {
        struct Node* temp = *head;
        // Traverse to the last node
        while (temp->next != *head) {
            temp = temp->next;
        }
        // Now the last node points to the new node
        temp->next = newNode;
        newNode->next = *head;
        *head = newNode; // Update head to the new node
    }
}
void insertAtPosition(struct Node** head, int data, int position) { struct
    Node* newNode = createNode(data);
    if (position == 1) {
        insertAtBeginning(head, data); return;
    }
    struct Node* temp = *head; int
    count = 1;
    while (temp != NULL && count < position - 1) { temp
        = temp->next;
        count++;
```

```
        if (temp == *head) break;
    }
    if (temp == NULL || temp->next == *head) {
        insertAtEnd(head, data);
    } else {
        newNode->next = temp->next;
        temp->next = newNode;
    }
}
void deleteFirst(struct Node**head){
   if(head == NULL){
     printf("LL is Empty");
    }
    if(head->next == head->data){
       free(head);
       head = NULL:
    }
   return;
}
struct  Node*  temp  =  head;
while(temp->next  !=  start){
temp = temp->head;
  temp = head;
}
temp->next = head->next;
free(head);
head = temp->next;
}
```

```
void deleteLast(struct Node*head){  if(head
  == NULL){
    printf("LL is emopty");
    return;
  }
  if(head->next == head){
    free(head);
    head = NULL;
    return;
  }
  struct Node* temp = head;
  while(temp->next->next != start){
  temp = temp->next;
  }
  free(temp->next);
  temp->next = start;
}
void deleteSpecific(struct Node* head, int ele){
  if(head == NULL){
    prinntf("Linked List is Empty"); return;
  }
  printf("Enter element");
  scanf("%d", ele);
  if(head->date == ele && head->next == start){
    free(head);
    head = NULL;
    return;
  }
```

```
  if(start->data == ele{
    struct *Node temp = start;
    while(temp->next != start){
    temp = temp->next;
    }
    temp->next = head->next;
    free(head);
    head = temp->head;
    return;
  }
  struct Node*temp = start;
  struct Node*prev = NULL:
  while(temp->data != ele && temp->link != start){
    prev = temp;
    temp = temp->next;
  }
  prev->next = temp->next;
  free(temp);
  return;
}
void display(struct Node* head) { if
    (head == NULL) {
        printf("List is empty.\n"); return;
    }
    struct Node* temp = head;
    do {
        printf("%d -> ", temp->data); temp
        = temp->next;
```

```
    } while (temp != head);
    printf("(head)\n");
}
int main() {
    struct Node* head = NULL;
    insertAtEnd(&head, 10);
    insertAtEnd(&head, 20);
    insertAtEnd(&head, 30);
    printf("List after inserting at the end:\n");
    display(head);
    insertAtBeginning(&head, 5);
    printf("List after inserting at the beginning:\n"); display(head);
    insertAtPosition(&head, 15, 3);
    printf("List after inserting at position 3:\n");
    display(head);
    insertAtPosition(&head, 25, 100);
    printf("List after inserting at position 100 (out of bounds):\n");
    display(head);
    return 0;
}
```

Output-

```
"C:\Users\AYUSH ADITYA\Doc
List after inserting at the end:
10 -> 20 -> 30 -> (head)
List after inserting at the beginning:
5 -> 10 -> 20 -> 30 -> (head)
List after inserting at position 3:
5 -> 10 -> 15 -> 20 -> 30 -> (head)
List after inserting at position 100 (out of bounds):
5 -> 25 -> 10 -> 15 -> 20 -> 30 -> (head)
List after deleting the first element:
25 -> 10 -> 15 -> 20 -> 30 -> (head)
List after deleting the last element:
25 -> 10 -> 15 -> 20 -> (head)
List after deleting element 20:
25 -> 10 -> 15 -> (head)

Process returned 0 (0x0)   execution time : 2.257 s
Press any key to continue.
```

Lab program 8

Doubly linked list implementation.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data); newNode-
    >next = *head;

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

    *head = newNode;
}
void insertAtEnd(struct Node** head, int data) {
```

```
    struct Node* newNode = createNode(data); if
    (*head == NULL) {
        *head = newNode; return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
void deleteNode(struct Node** head, int data) { if
    (*head == NULL) {
        printf("List is empty.\n"); return;
    }
    struct Node* temp = *head;
    while (temp != NULL && temp->data != data) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Node with value %d not found.\n", data);
        return;
    }
    if (*head == temp) {
        *head = temp->next;
    }
    if (temp->prev != NULL) {
```

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

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

```
}
int main() {
    struct Node* head = NULL;
    insertAtBeginning(&head, 10);
    insertAtBeginning(&head, 20);
    insertAtEnd(&head, 30);
    insertAtEnd(&head, 40);
    displayList(head);
    deleteNode(&head, 20);
    displayList(head);
    deleteNode(&head, 10);
    displayList(head);
    freeList(head);
    return 0;
}
```

Output-

```
Doubly Linked List: 20 10 30 40
Node with value 20 deleted.
Doubly Linked List: 10 30 40
Node with value 10 deleted.
Doubly Linked List: 30 40

Process returned 0 (0x0)   execution time : 0.021 s
Press any key to continue.
```

Lab program 9

Implementation of Binary Search Tree.

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

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

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

```
    {
        root->right=insert(root->right,value);
    }
};

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

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

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

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

int main()
{
    struct Node* root = NULL;
    int num, value;

    printf("Enter the number of nodes you want to insert: ");
    scanf("%d", &num);

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

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

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

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

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

```
    return 0;

}
```

Output-

```
Enter the number of nodes you want to insert: 4
Enter 4 values to insert into the binary search tree:
1
2
3
4

Postorder traversal:
4  3  2  1
Preorder traversal:
1  2  3  4
Inorder traversal:
1  2  3  4

Process returned 0 (0x0)   execution time : 10.511 s
Press any key to continue.
```

Lab program 10

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

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


struct Queue {

    int *arr;

    int front, rear; int

    capacity;

};


struct Queue* createQueue(int capacity) {

    struct Queue *queue = (struct Queue*)malloc(sizeof(struct Queue));

    queue->capacity = capacity;

    queue->front = -1;

    queue->rear = -1;

    queue->arr = (int*)malloc(capacity * sizeof(int)); return

    queue;

}


bool isEmpty(struct Queue *queue) { return

    queue->front == -1;

}


void enqueue(struct Queue *queue, int value) { if

    (queue->rear == queue->capacity - 1) {

        printf("Queue overflow\n");

        return;
```

```
    }
    if (queue->front == -1) {
        queue->front = 0;
    }
    queue->rear++;
    queue->arr[queue->rear] = value;
}

int dequeue(struct Queue *queue) { if
    (isEmpty(queue)) {
        printf("Queue underflow\n");
        return -1;
    }
    int value = queue->arr[queue->front];
    queue->front++;
    if (queue->front > queue->rear) {
        queue->front = queue->rear = -1;
    }
    return value;
}

void bfs(int graph[][5], int start, int n) {
    bool visited[n];
    for (int i = 0; i < n; i++) {
        visited[i] = false;
    }
    struct Queue *queue = createQueue(n);

    visited[start] = true;
```

```
    enqueue(queue, start);


    printf("BFS Traversal: ");
    while (!isEmpty(queue)) {
        int node = dequeue(queue); printf("%d
        ", node);


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


int main() {
    int n;
    printf("Enter the value for n \n");
    scanf("%d", &n);
    int graph[n][n];
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            printf("Enter value for graph element\n"); scanf("\n%d",
            &graph[i][j]);
        }
    }
    bfs(graph, 0, n);
```

```
    return 0;

}
```

Output-

```
Enter the value for n
4
Enter value for graph element
1
Enter value for graph element
1
Enter value for graph element
1
Enter value for graph element
0
Enter value for graph element
0
Enter value for graph element
0
Enter value for graph element
0
Enter value for graph element
1
Enter value for graph element
1
Enter value for graph element
0
Enter value for graph element
1
Enter value for graph element
0
Enter value for graph element
0
Enter value for graph element
1
Enter value for graph element
1
Enter value for graph element
1
BFS Traversal: 0 1 2 3

Process returned 0 (0x0)   execution time : 110.497 s
Press any key to continue.
```

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

```
#include <stdio.h>

#include <stdbool.h>


void dfs(int graph[][5], int node, bool visited[], int n) { visited[node]

    = true;

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

        if (graph[node][i] == 1 && !visited[i]) {

            dfs(graph, i, visited, n);

        }

    }

}

bool isConnected(int graph[][5], int n) {

    bool visited[n];

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

        visited[i] = false;

    }

    dfs(graph, 0, visited, n);



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

        if (!visited[i]) {

            return false;

        }

    }

    return true;

}

int main() {

    int n;
```

```
    printf("Enter the value for n \n");
    scanf("%d", &n);
    int graph[n][n];
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            printf("Enter value for graph element\n"); scanf("\n%d",
            &graph[i][j]);
        }
    }
    if (isConnected(graph, n)) { printf("The
        graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }

    return 0;
}
```

Output-

```
Enter the value for n
4
Enter value for graph element
1
Enter value for graph element
1
Enter value for graph element
1
Enter value for graph element
0
Enter value for graph element
0
Enter value for graph element
1
Enter value for graph element
1
Enter value for graph element
0
Enter value for graph element
0
Enter value for graph element
0
Enter value for graph element
0
Enter value for graph element
1
Enter value for graph element
1
Enter value for graph element
1
Enter value for graph element
1
Enter value for graph element
0
The graph is connected.

Process returned 0 (0x0)   execution time : 18.001 s
Press any key to continue.
```