VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Crevan Neil Fernandes (1BM23CS082)

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 Crevan Neil Fernandes **(1BM23CS082)**, 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.

Dr. Selva kumar S
Associate 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>
#include<stdlib.h>
```

```
#define STACK_SIZE 5
void push(int st[],int *top)
{
	int item;
	if(*top==STACK_SIZE-1)
		printf("Stack overflow\n");
	else
	{
		printf("\nEnter an item :");
		scanf("%d",&item);
		(*top)++;
		st[*top]=item;
	}
}
void pop(int st[],int *top)
{
	if(*top==-1)
		printf("Stack underflow\n");
	else
	{
		printf("\n%d item was deleted",st[(*top)--]);
	}
}
void display(int st[],int *top)
{
	int i;
	if(*top==-1)
		printf("Stack is empty\n");
	for(i=0;i<=*top;i++)
		printf("%d\t",st[i]);
}
void main()
{
	int st[10],top=-1, c,val_del;
	while(1)
	{
		printf("\n1. Push\n2. Pop\n3. Display\n");
		printf("\nEnter your choice :");
		scanf("%d",&c);
		switch(c)
		{
			case 1: push(st,&top);
				break;
			case 2: pop(st,&top);
				break;
			case 3: display(st,&top);
				break;
			default: printf("\nInvalid choice!!!");
				exit(0);
		}
```

```
        }
}
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS D:\jyothika\DST> cd "d:\jyothika\DST\" ; if ($?) { gcc 1.c -o 1 } ; if ($?) { .\1 }

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :12

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :65

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :45

1. Push
2. Pop
3. Display

Enter your choice :1
Stack overflow
```

```
1. Push
2. Pop
3. Display

Enter your choice :2

45 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2

65 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :3
12
1. Push
2. Pop
3. Display

Enter your choice :2

12 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2
Stack underflow

1. Push
2. Pop
3. Display

Enter your choice :4

Invalid choice!!!
```

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<stdlib.h>
#include<ctype.h>
#define max 1000
char stack[max];
int top=-1;
void push(char x){
   if(top==max-1){
      printf("stack full\n");
      return;
   }
   stack[++top]=x;
}
char pop(){
   if(top==-1){
      printf("stack empty\n");
      return -1;
   }
   return stack[top--];
}
int precedence(char x){
   if(x=='+'||x=='-'){
      return 1;
   }
   else if(x=='*'||x=='/'){
      return 2;
```

```
    }
    else if(x=='^'){
        return 3;
    }
    else{
        return 0;
    }
}
int isop(char x){
    return (x=='+'||x=='-'||x=='*'||x=='/'||x=='^');
}
void infixtopostfix(char *exp){
    char postfix[max];
    int i=0;
    char *ptr = exp;
    while(*ptr!='\0'){
        if(isalpha(*ptr)){
            postfix[i++] = *ptr;
        }
        if(*ptr=='('){
            push(*ptr);
        }
        if(*ptr==')'){
            while(stack[top]!='('){
                postfix[i++]=pop();
            }
            pop();
        }
        if(isop(*ptr)){
```

```
            while(precedence(stack[top])>=precedence(*ptr)){
                postfix[i++]=pop();
            }
            push(*ptr);
        }
        ptr++;
    }
    while(top!=-1){
        postfix[i++]=pop();
    }
    postfix[i]='\0';
    printf("Postfix:%s",postfix);
}
int main(){
    char exp[max];
    printf("enter expression:");
    scanf("%s",exp);
    infixtopostfix(exp);
    return 0;
}
```

Output:

```
enter expression:A*B+C*D-E
Postfix:AB*CD*+E-
Process returned 0 (0x0)   execution time : 86.993 s
Press any key to continue.
```

Lab Program 3a):

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

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

```
#define Max 6
int queue[Max];
int front = -1, rear = -1;

void insert() {
   int a;
   printf("Enter the element to be entered: ");
   scanf("%d", &a);
   if (rear == Max - 1) {
      printf("The queue is full\n");
   } else if (front == -1) {
      front = 0;
      rear = 0;
      queue[rear] = a;
   } else {
      rear = rear + 1;
      queue[rear] = a;
   }
}

void del() {
   if (front == -1 || front > rear) {
      printf("The queue is empty\n");
   } else {
      printf("%d has been deleted\n", queue[front]);
      front = front + 1;
   }
}
```

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

int main() {
    int choice, i = 1;
    do {
        printf("1: Insert \n");
        printf("2: Delete\n");
        printf("3: Display\n");
        printf("4: Exit\n");
        printf("Enter the choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                insert();
                break;
            case 2:
                del();
                break;
            case 3:
                display();
```

```
            break;
        case 4:
            i = 0;
            break;
        default:
            printf("Invalid entry\n");
      }
   } while (i == 1);
   return 0;
}
```

Output:

```
Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Choose an option: 1
Enter a value to enqueue: 4
Enqueued: 4

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Choose an option: 1
Enter a value to enqueue: 5
Enqueued: 5

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Choose an option: 2
Dequeued: 4

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Choose an option: 3
Queue contents: 5

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Choose an option: 4
Exiting...
```

3 b)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 <stdio.h>



#define MAX 5
int queue[MAX];
int front = -1;
int rear = -1;



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



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



void enqueue(int value) {
    if (isFull()) {
        printf("Queue is full. Cannot enqueue %d\n", value);
        return;
    }
    if (isEmpty()) {
        front = 0;
```

```
    }
    rear = (rear + 1) % MAX;
    queue[rear] = value;
    printf("Enqueued: %d\n", value);
}


int dequeue() {
    if (isEmpty()) {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }
    int item = queue[front];
    if (front == rear) {

        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % MAX;
    }
    printf("Dequeued: %d\n", item);
    return item;
}


void display() {
    if (isEmpty()) {
        printf("Queue is empty.\n");
        return;
```

```
    }
    printf("Queue contents: ");
    int i = front;
    while (1) {
        printf("%d ", queue[i]);
        if (i == rear) break;
        i = (i + 1) % MAX;
    }
    printf("\n");
}


int main() {
    int choice, value,i=1;

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

        switch (choice) {
            case 1:
                printf("Enter a value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
```

```
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting...\n");
            i=0;
        default:
            printf("Invalid option. Please try again.\n");
      }
   }

   return 0;
}
```

Output:

```
1: Insert
2: Delete
3: Display
4: Exit
Enter the choice: 1
Enter the element to be entered: 6
1: Insert
2: Delete
3: Display
4: Exit
Enter the choice: 1
Enter the element to be entered: 8
1: Insert
2: Delete
3: Display
4: Exit
Enter the choice: 2
6 has been deleted
1: Insert
2: Delete
3: Display
4: Exit
Enter the choice: 3
8
1: Insert
2: Delete
3: Display
4: Exit
Enter the choice: 4

Process returned 0 (0x0)   execution time : 28.000 s
Press any key to continue.
```

Write a program to implement Queues using Stacks

```
typedef struct {
   int ar[100];
   int head;
   int tail;
   int cnt;
} MyQueue;
```

```
MyQueue* myQueueCreate() {
    MyQueue* obj = malloc(sizeof(MyQueue));
    obj->head = 0;
    obj->tail = 0;
    obj->cnt = 0;
    return obj;
}

void myQueuePush(MyQueue* obj, int x) {
    if(obj == NULL) return;

    obj->cnt++;
    obj->ar[obj->tail] = x;
    obj->tail = (obj->tail + 1)%100;
}

int myQueuePop(MyQueue* obj) {
    if(obj == NULL) return NULL;

    obj->cnt--;
    obj->head = (obj->head + 1)%100;
    return (obj->ar[(obj->head-1)%100]);
}
```

```
int myQueuePeek(MyQueue* obj) {
    if(obj == NULL) return NULL;

    return obj->ar[obj->head];
}

bool myQueueEmpty(MyQueue* obj) {
    if(obj == NULL) return false;

    return (obj->cnt?false:true);
}

void myQueueFree(MyQueue* obj) {
    if(obj == NULL) return;
    free(obj);
}
```

Output:

Input

```
["MyQueue","push","push","peek","pop","empty"]
```

```
[[],[1],[2],[],[],[]]
```

Output

```
[null,null,null,1,1,false]
```

Expected

```
[null,null,null,1,1,false]
```

Lab Program 4a):

Write a program to Implement Singly Linked List with following operations

a) Create a linked list.

b) Insertion of a node at first position, at any position and at end of list.

Display the contents of the linked list.

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

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



struct node* create(int data){
   struct node* newnode= (struct node*) malloc(sizeof(struct node));
   newnode->data = data;
   newnode->next = NULL;
   return newnode;
}

 void insert_at_beginning(struct node** head , int data){
     struct node* newnode = create(data);
     newnode->next = *head;
     *head = newnode;
```

```
 }



void insert_at_end(struct node** head , int data){
    struct node * newnode = create(data);
    if (*head == NULL){
        *head = newnode;
        return;
    }
    struct node* temp = *head;
    while(temp->next!= NULL)temp = temp->next;
    temp->next = newnode;
}



 void insert_at_position(struct node** head , int data , int pos){
     struct node* newnode = create(data);
     if (pos <1){
         printf("Invalid location\n");
         return;
     }

     if (pos ==1){
         newnode->next = *head;
```

```
        *head = newnode;
        return;
    }

    struct node* temp = *head;
    for (int i=0; temp != NULL && i<pos-1; ++i){
        temp= temp->next;
    }

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

    newnode->next = temp->next;
    temp->next = newnode;
}
void display(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");



}

int main(){
    struct node* head1 = NULL;
    int data , position ,x,y,z;

    while (1){
        printf("Enter choice: \n");
        printf("1. Insert at beginning  \n");
        printf("2.Insert at end \n");
        printf("3. Insert at any position \n");
        printf("4. Display list\n")
        printf("5 .Exit\n");
        scanf("%d" , &x);
        switch (x)
        {
        case 1:
            printf("Enter the value to be inserted\n");
            scanf("%d" ,&y);
      insert_at_beginning(&head1,y);
                    break;
```

```
        case 2:
              printf("Enter the value to be inserted\n");
            scanf("%d" ,&y);
       insert_at_end(&head1,y);

            break;

        case 3:
              printf("Enter the value to be inserted\n");
            scanf("%d" ,&y);
            printf(“Enter the position:”);
            scanf(“%d”,&position);
       insert_at_position(&head1,y,position);
        case4;
        display(&head1);
            break;
     case 5:
            return 0;

        default:
            printf("Invalid entry");
            break;
        }
    }
    return 0;
}
```

Output:

```
Enter choice:
1. Insert at beginning
2.Insert at end
3. Insert at any position
4. Display list
5 .Exit
1
Enter the value to be inserted
2
Enter choice:
1. Insert at beginning
2.Insert at end
3. Insert at any position
4. Display list
5 .Exit
2
Enter the value to be inserted
5
Enter choice:
1. Insert at beginning
2.Insert at end
3. Insert at any position
4. Display list
5 .Exit
3
Enter the value to be inserted
9
Enter the position:2
2 ->5 ->9 ->NULL
Enter choice:
1. Insert at beginning
2.Insert at end
3. Insert at any position
4. Display list
5 .Exit
4
2 ->5 ->9 ->NULL
Enter choice:
1. Insert at beginning
2.Insert at end
3. Insert at any position
```

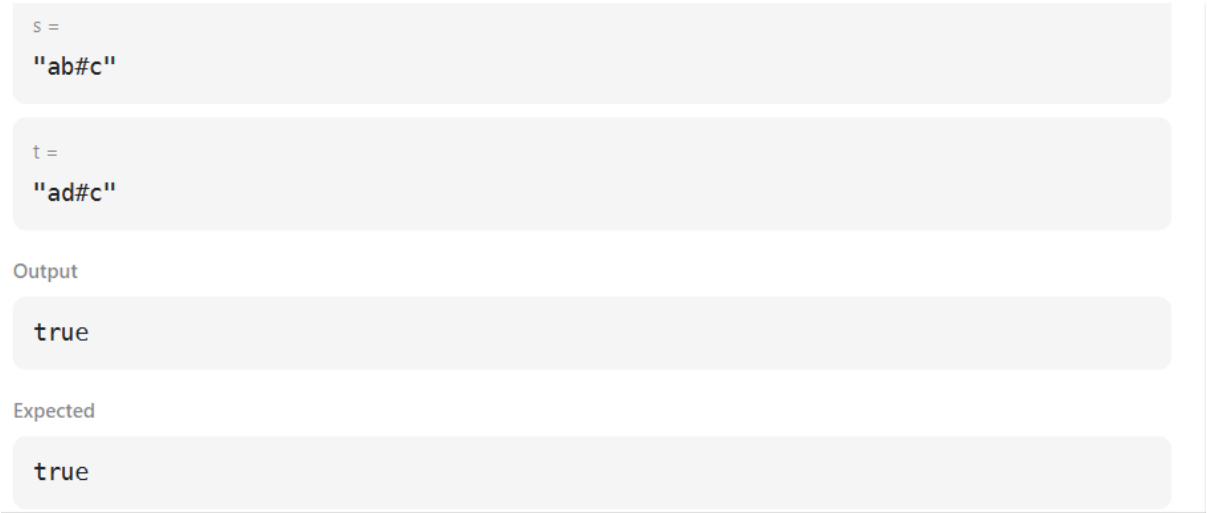
4b)

Given two strings s and t, return true *if they are equal when both are typed into empty text editors*. '#' means a backspace character.

```
int backspaceCompare(char* s, char* t) {
    char ss[1000];
    char tt[1000];
    int i, j;



    i = 0;
    for (int k = 0; s[k] != '\0'; ++k) {
        if (s[k] == '#') {
            if (i > 0) {
                --i;
            }

        } else {
            ss[i++] = s[k];
        }
    }
    ss[i] = '\0';

     j = 0;
    for (int k = 0; t[k] != '\0'; ++k) {
        if (t[k] == '#') {
            if (j > 0) {
                --j;
```

```
            }
        } else {
            tt[j++] = t[k];
        }
    }
    tt[j] = '\0';

    return strcmp(ss, tt) == 0;
}
```

Output:



Lab Program 5:

Write a program to Implement Singly Linked List with following operations

a) Create a linked list.

b) Deletion of first element, specified element and last element in the list.

c) Display the contents of the linked list.

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

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

struct Node* head = NULL;

void insert_at_beginning(int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = data;
    if (head == NULL) {
        head = new_node;
        new_node->next = NULL;
        return;
    }
    new_node->next = head;
    head = new_node;
}

void insert_at_end(int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* ptr = head;
    new_node->data = data;

    if (head == NULL) {
        head = new_node;
```

```
        new_node->next = NULL;
        return;
    }

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

void delete_from_start() {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
    struct Node* ptr = head;
    head = head->next;
    free(ptr);
}

void delete_from_end() {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
```

```
    struct Node* ptr = head;
    struct Node* prev = NULL;

    if (ptr->next == NULL) { // If there is only one node
        free(ptr);
        head = NULL;
        return;
    }

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

    prev->next = NULL;
    free(ptr);
}

void delete_from_pos(int pos) {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
    if (pos < 1) {
        printf("Invalid position.\n");
```

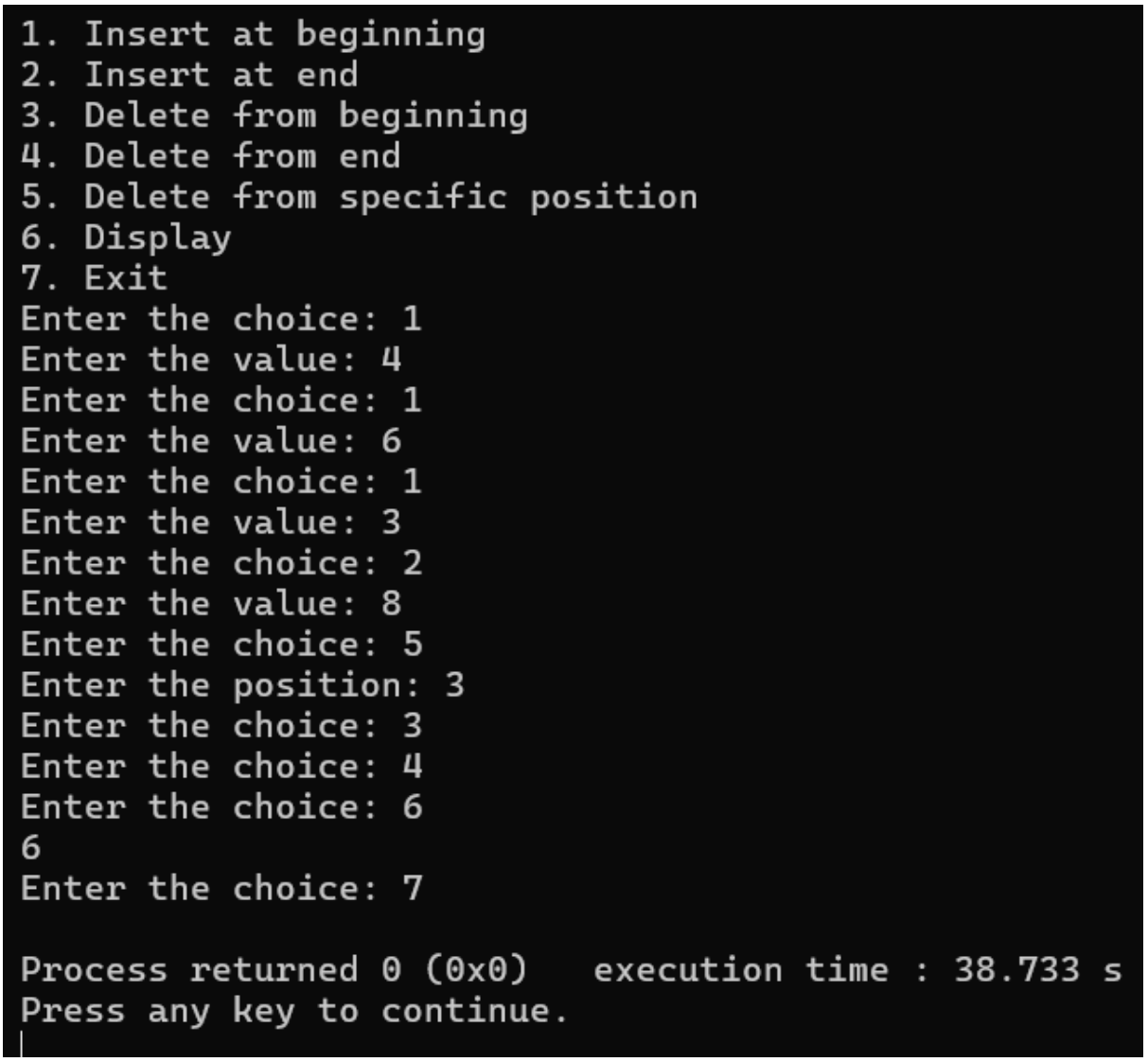
```
        return;
    }

    struct Node* ptr = head;
    struct Node* prev = NULL;

    if (pos == 1) {
        head = ptr->next;
        free(ptr);
        return;
    }

    for (int i = 1; ptr != NULL && i < pos; i++) {
        prev = ptr;
        ptr = ptr->next;
    }

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

    prev->next = ptr->next;
    free(ptr);
}
```

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

int main() {
    int choice, data, i = 1;
    printf("1. Insert at beginning\n");
    printf("2. Insert at end\n");
    printf("3. Delete from beginning\n");
    printf("4. Delete from end\n");
    printf("5. Delete from specific position\n");
    printf("6. Display\n");
    printf("7. Exit\n");

    while (i == 1) {
        printf("Enter the choice: ");
        scanf("%d", &choice);
```

```
switch (choice) {
    case 1:
        printf("Enter the value: ");
        scanf("%d", &data);
        insert_at_beginning(data);
        break;
    case 2:
        printf("Enter the value: ");
        scanf("%d", &data);
        insert_at_end(data);
        break;
    case 3:
        delete_from_start();
        break;
    case 4:
        delete_from_end();
        break;
    case 5:
        printf("Enter the position: ");
        scanf("%d", &data);
        delete_from_pos(data);
        break;
    case 6:
        display();
        break;
    case 7:
```

```
            i = 0;
            break;
        default:
            printf("Invalid choice, please try again.\n");
    }
  }
  return 0;
}
```

Output:



5b) Write a program to delete duplicates from a sorted linked list

```
struct ListNode* deleteDuplicates(struct ListNode* head) {
 struct ListNode*k=head;
struct ListNode*z;
if(head==NULL)
return head;
int i;
for(i=0;i<5;i++) {
while(k->next!=NULL) {
z=k;
k=k->next;
if(z->val==k->val){
z->next=k->next;
}
}
k=head;
}
return head;
}
```

Output:

Input

head =

[1,1,2]

Output

[1,2]

Expected

[1,2]

Lab Program 6a):

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

```
#include <stdio.h>

#include <stdlib.h>


struct node

{

   int data;

   struct node * next;

};



struct node* create(int data){

   struct node* newnode= (struct node*) malloc(sizeof(struct node));

   newnode->data = data;

   newnode->next = NULL;

   return newnode;

}


// void insert_at_beginning(struct node** head , int data){

//     struct node* newnode = create(data);

//     newnode->next = *head;

//     *head = newnode;

// }
```

```
void insert_at_end(struct node** head , int data){
    struct node * newnode = create(data);
    if (*head == NULL){
        *head = newnode;
        return;
    }
    struct node* temp = *head;
    while(temp->next!= NULL)temp = temp->next;
    temp->next = newnode;
}



// void insert_at_position(struct node** head , int data , int pos){
//     struct node* newnode = create(data);
//     if (pos <1){
//         printf("Invalid location\n");
//         return;
//     }

//     if (pos ==1){
//         newnode->next = *head;
//         *head = newnode;
//         return;
//     }
```

```
//     struct node* temp = *head;
//     for (int i=0; temp != NULL && i<pos-1; ++i){
//         temp= temp->next;
//     }

//     if (temp == NULL) {
//         printf("Position out of range\n");
//         return;
//     }

//     newnode->next = temp->next;
//     temp->next = newnode;
// }




void display(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");


}



void sort (struct node** head){
    if (*head==NULL)return;

    struct node * i = *head;
    struct node* j;
    int temp;


    while (i!= NULL){
        j= i->next;
        while (j->next!= NULL){
            if (i->data > j->data){
                temp =  i->data;
                i->data = j->data;
                j->data = temp;
            }
            j = j->next;
```

```
        }
        i = i->next;
    }
}



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



void reverse (struct node** head){
    struct node* prev = NULL , *current = *head, *next = NULL;

    while (current!= NULL){
        next = current->next;
        current->next = prev;
```

```
        prev = current;
        current = next;
    }
    *head = prev;
}

int main(){
    struct node* head1 = NULL;
    struct node* head2 = NULL;
    int data , position ,x,y,z;

    while (1){
        printf("Enter choice: \n");
        printf("1. Insert in list \n");
        printf("2. Sort list \n");
        printf("3. Reverse list \n");
        printf("4. display list\n");
        printf("5. Concatenate\n");
        printf("6 .Exit\n");
        scanf("%d" , &x);
        switch (x)
        {
        case 1:
            printf("Enter the value to be inserted\n");
            scanf("%d" ,&y);
            printf("Which list?\n");
```

```
    scanf("%d", &z);
    if (z == 1)insert_at_end(&head1,y);
    else insert_at_end(&head2,y);
    break;

case 2:
    printf("Which list?\n");
    scanf("%d", &z);
    if (z == 1)sort(&head1);
    else sort(&head2);
    printf("List sorted\n");
    break;

case 3:
    printf("Which list?\n");
    scanf("%d", &z);
    if (z == 1)reverse(&head1);
    else reverse(&head2);
    printf("List reversed\n");
    break;

case 4:
    printf("Which list?\n");
    scanf("%d", &z);
    if (z == 1)display(&head1);
    else display(&head2);
```

```
            break;



        case 5:
            concatenate(&head1, &head2);
            printf("Lists concatenated\n");
            break;

        case 6:
            return 0;

        default:
            printf("Invalid entry");
            break;
        }
    }
    return 0;
}
```

Output:

```
Enter choice:
1. Insert in list
2. Sort list
3. Reverse list
4. display list
5. Concatenate
6 .Exit
1
Enter the value to be inserted
5
Which list?
1
Enter choice:
1. Insert in list
2. Sort list
3. Reverse list
4. display list
5. Concatenate
6 .Exit
1
Enter the value to be inserted
3
Which list?
1
Enter choice:
1. Insert in list
2. Sort list
3. Reverse list
4. display list
5. Concatenate
6 .Exit
1
Enter the value to be inserted
9
Which list?
2
Enter choice:
1. Insert in list
2. Sort list
3. Reverse list
4. display list
```

```
2. Sort list
3. Reverse list
4. display list
5. Concatenate
6 .Exit
3
Which list?
1
List reversed
Enter choice:
1. Insert in list
2. Sort list
3. Reverse list
4. display list
5. Concatenate
6 .Exit
5
Lists concatenated
Enter choice:
1. Insert in list
2. Sort list
3. Reverse list
4. display list
5. Concatenate
6 .Exit
4
Which list?
1
3 ->5 ->9 ->NULL
Enter choice:
1. Insert in list
2. Sort list
3. Reverse list
4. display list
5. Concatenate
6 .Exit
6

Process returned 0 (0x0)   execution time : 91.347 s
Press any key to continue.
```

6b) Write a program to Implement Single Link List to simulate Stack & Queue Operations.

```
#include <stdio.h>

#include <stdlib.h>


typedef struct s_node {
```

```
    int data;
    struct s_node* next;
} s_node;

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

s_node* push(s_node **last, int data, s_node **head) {
    if (*head == NULL) {

        *head = init(data);
        *last = *head;
        return *head;
    }

    s_node *l = *last;
    l->next = init(data);
    *last = l->next;
```

```
    return l->next;
}

s_node* pop(s_node **head) {
    if (*head == NULL) {
        printf("List is empty, nothing to pop.\n");
        return NULL;
    }

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


    while (h->next->next != NULL) {
        h = h->next;
    }

    s_node *temp = h->next;
    free(temp);
    h->next = NULL;
    return h;
}
```

```
void print_list(s_node *head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

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

int main() {
    s_node *head = NULL;
    s_node *last = NULL;

    int choice, data;
    while (1) {
        printf("\nMenu:\n");
        printf("1. Push a node\n");
        printf("2. Pop a node\n");
        printf("3. Print the list\n");
```

```
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter the data to push: ");
        scanf("%d", &data);
        last = push(&last, data, &head);
        break;

    case 2:
        last = pop(&head);
        break;

    case 3:
        print_list(head);
        break;

    case 4:

        while (head != NULL) {
            head = pop(&head);
        }
        printf("Exiting...\n");
        return 0;
```

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

Output:

```
Menu:
1. Push a node
2. Pop a node
3. Print the list
4. Exit
Enter your choice: 1
Enter the data to push: 5

Menu:
1. Push a node
2. Pop a node
3. Print the list
4. Exit
Enter your choice: 1
Enter the data to push: 3

Menu:
1. Push a node
2. Pop a node
3. Print the list
4. Exit
Enter your choice: 2

Menu:
1. Push a node
2. Pop a node
3. Print the list
4. Exit
Enter your choice: 3
5 -> NULL

Menu:
1. Push a node
2. Pop a node
3. Print the list
4. Exit
Enter your choice: 4
Exiting...

Process returned 0 (0x0)   execution time : 42.539 s
```

```
#include <stdio.h>

#include <stdlib.h>

typedef struct p_node {

    int data;
```

```
    struct p_node* next;
} p_node;
p_node* init(int data) {
    p_node *n = (p_node*)malloc(sizeof(p_node));
    if (n == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    n->data = data;
    n->next = NULL;
    return n;
}
p_node* enquee(p_node **last, int data, p_node **head) {
    if (*head == NULL) {
        *head = init(data);
        *last = *head;
        return *head;
    }
    p_node *l = *last;
    l->next = init(data);
    *last = l->next;
    return l->next;
}
int dequee(p_node **head) {
    if (*head == NULL) {
        printf("Queue is empty, nothing to dequeue.\n");
```

```
        return -1;
    }
    p_node *h = *head;
    int r = h->data;
    *head = h->next;
    free(h);
    return r;
}
void print_list(p_node *head) {
    if (head == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    p_node *current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
int main() {
    p_node *head = NULL;
    p_node *last = NULL;
    int choice, data;
    while (1) {
        printf("\nMenu:\n");
```

```
printf("1. Enqueue a node\n");
printf("2. Dequeue a node\n");
printf("3. Print the queue\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
    case 1:
        printf("Enter the data to enqueue: ");
        scanf("%d", &data);
        last = enquee(&last, data, &head);
        break;
    case 2:
        data = dequee(&head);
        if (data != -1) {
            printf("Dequeued: %d\n", data);
        }
        break;
    case 3:
        print_list(head);
        break;
    case 4:
        while (head != NULL) {
            dequee(&head);
        }
        printf("Exiting...\n");
```

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

Output:

```
Menu:
1. Enqueue a node
2. Dequeue a node
3. Print the queue
4. Exit
Enter your choice: 1
Enter the data to enqueue: 3

Menu:
1. Enqueue a node
2. Dequeue a node
3. Print the queue
4. Exit
Enter your choice: 1
Enter the data to enqueue: 7

Menu:
1. Enqueue a node
2. Dequeue a node
3. Print the queue
4. Exit
Enter your choice: 2
Dequeued: 3

Menu:
1. Enqueue a node
2. Dequeue a node
3. Print the queue
4. Exit
Enter your choice: 3
7 -> NULL

Menu:
1. Enqueue a node
2. Dequeue a node
3. Print the queue
4. Exit
Enter your choice: 4
Exiting...
```

Lab Program 7:

Write a program to Implement doubly link list with primitive operations

a) Create a doubly linked list.

b) Insert a new node to the left of the node.

c) Delete the node based on a specific value

d) Display the contents of the list

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

struct node {
    int data;
    struct node *next;
    struct node *prev;
};

struct node* create(int data) {
    struct node* newnode = (struct node*) malloc(sizeof(struct node));
    newnode->data = data;
    newnode->next = NULL;
    newnode->prev = NULL;
    return newnode;
}

void insert_at_beginning(struct node** head, int data) {
    struct node* newnode = create(data);
    newnode->next = *head;
```

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

void delete_from_val(struct node **head, int data) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct node *ptr = *head;


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


    if (ptr == NULL) {
        printf("Element %d does not exist in list\n", data);
        return;
    }
```

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

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



    free(ptr);
}

void display(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");
}

int main() {
    struct node* head = NULL;
    int x, y;

    while (1) {
        printf("Enter choice: \n");
        printf("1. Insert in list \n");
        printf("2. Delete from list \n");
        printf("3. Display \n");
        printf("4. Exit\n");
        scanf("%d", &x);

        switch (x) {
            case 1:
                printf("Enter the value to be inserted: ");
                scanf("%d", &y);
                insert_at_beginning(&head, y);
                break;

            case 2:
                printf("Enter value to be deleted: ");
                scanf("%d", &y);
```

```
            delete_from_val(&head, y);
            break;

        case 3:
            display(&head);
            break;

        case 4:
            return 0;

        default:
            printf("Invalid entry\n");
            break;
      }
   }

   return 0;
}
```

Output:

```
Enter choice:
1. Insert in list
2. Delete from list
3. Display
4. Exit
1
Enter the value to be inserted: 7
Enter choice:
1. Insert in list
2. Delete from list
3. Display
4. Exit
1
Enter the value to be inserted: 4
Enter choice:
1. Insert in list
2. Delete from list
3. Display
4. Exit
2
Enter value to be deleted: 4
Enter choice:
1. Insert in list
2. Delete from list
3. Display
4. Exit
3
7 ->NULL
Enter choice:
1. Insert in list
2. Delete from list
3. Display
4. Exit
4

Process returned 0 (0x0)   execution time : 32.825 s
Press any key to continue.
```

Lab program 8:

Write a program

a) To construct a binary Search tree.

b) To traverse the tree using all the methods i.e., in-order, preorder and post order to display the elements in the tree.

```
#include <stdio.h>

#include <stdlib.h>
```

```
typedef struct BST {
    int data;
    struct BST *left;
    struct BST *right;
} node;

node *create() {
    node *temp;
    printf("\nEnter data: ");
    temp = (node*)malloc(sizeof(node));
    scanf("%d", &temp->data);
    temp->left = temp->right = NULL;
    return temp;
}

void insert(node *root, node *temp) {
    if (temp->data < root->data) {
        if (root->left != NULL)
            insert(root->left, temp);
        else
            root->left = temp;
    }
    if (temp->data > root->data) {
        if (root->right != NULL)
            insert(root->right, temp);
        else
```

```
            root->right = temp;
        }
    }

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

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

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

```
}

int main() {
    char ch;
    node *root = NULL, *temp;

    do {
        temp = create();
        if (root == NULL)
            root = temp;
        else
            insert(root, temp);

        printf("\nDo you want to enter more(y/n)? ");
        getchar();  // To consume the newline character from previous input
        scanf("%c", &ch);
    } while (ch == 'y' || ch == 'Y');

    printf("\nPreorder Traversal: ");
    preorder(root);

    printf("\nInorder Traversal: ");
    inorder(root);

    printf("\nPostorder Traversal: ");
    postorder(root);
```

```
    return 0;
}
```

Output:

```
Enter data: 5

Do you want to enter more(y/n)? y

Enter data: 6

Do you want to enter more(y/n)? y

Enter data: 1

Do you want to enter more(y/n)? n

Preorder Traversal: 5 1 6
Inorder Traversal: 1 5 6
Postorder Traversal: 1 6 5
Process returned 0 (0x0)   execution time : 18.950 s
Press any key to continue.
```

Lab Program 9a):

Write a program to traverse a graph using BFS method.

#include<stdio.h> void bfs(int); int a[10][10],vis[10],n;

void main()

{ int i,j,src;

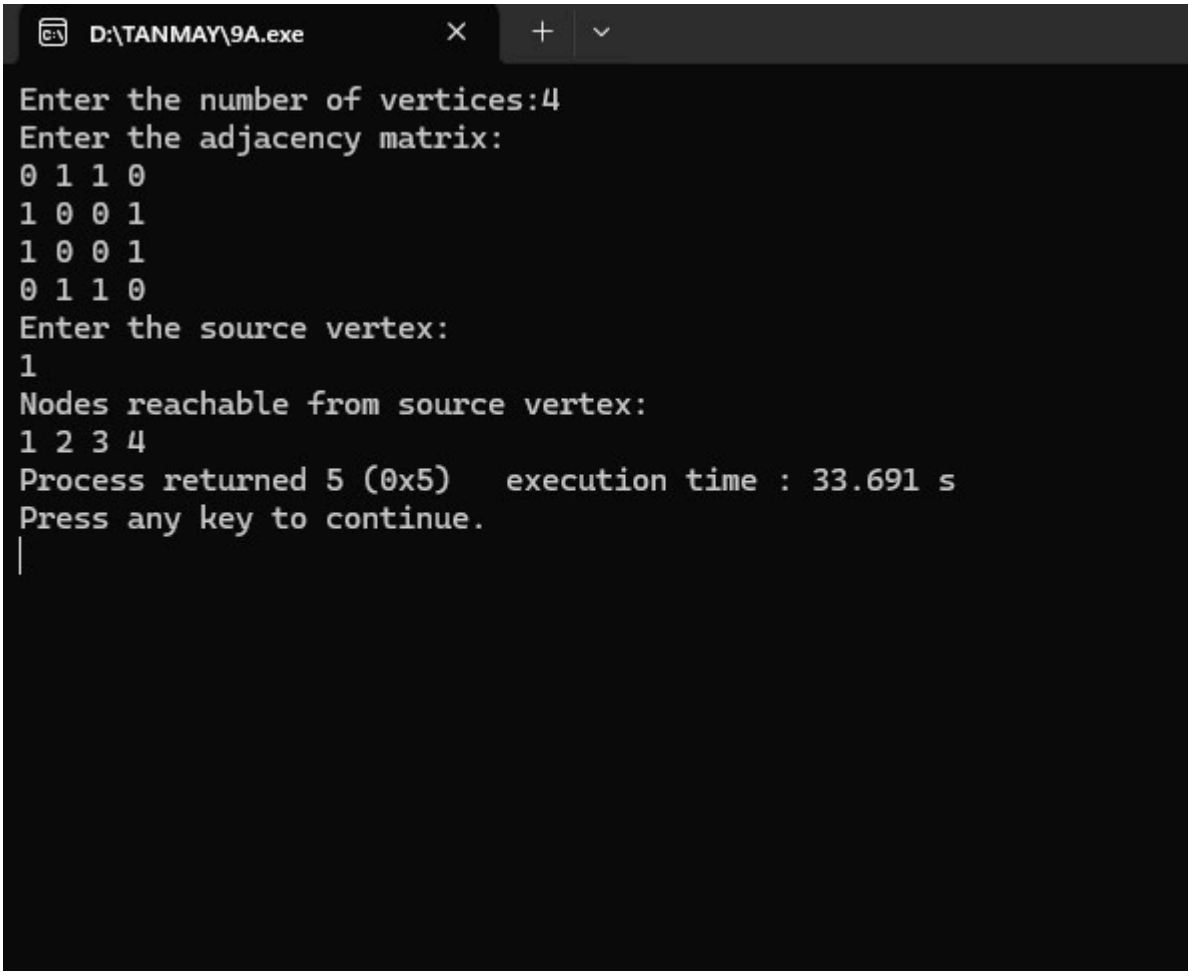
printf("enter the number of vertices\n"); scanf("%d",&n); printf("enter the adjacency matrix\n"); for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

```
    {
        scanf("%d",&a[i][j]);

    }

    vis[i]=0;
  }

  printf("enter the src vertex\n");  scanf("%d",&src);  printf("nodes reachable from src vertex\n");  bfs(src);

}

void bfs(int v)
{   int q[10],f=1,r=1,u,i;   q[r]=v;   vis[v]=1;   while(f<=r)
  {     u=q[f];     printf("%d ",u);     for(i=1;i<=n;i++)
    {
      if(a[v][i]==1 && vis[i]==0)
      {
        vis[i]=1;       r=r+1;       q[r]=i;
      }
    }
    f=f+1;
  }
}
```

Output:



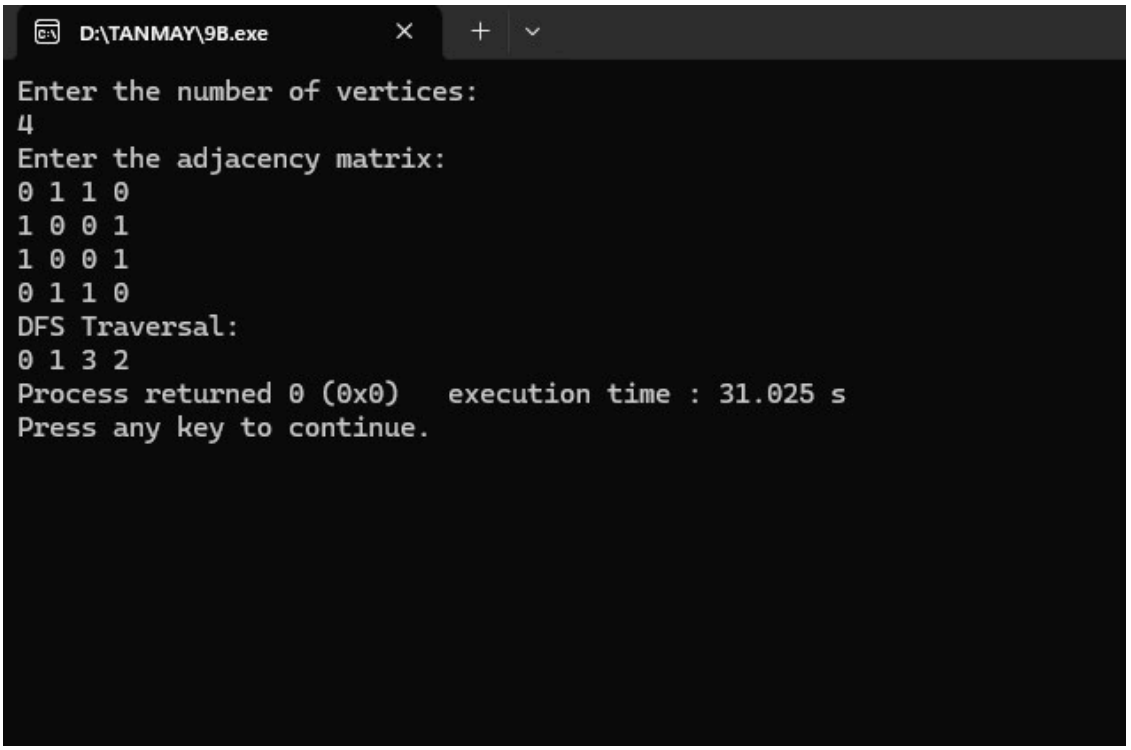
Lab Program 9b)

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

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

int i,j,n,a[10][10],vis[10]; void dfs(int v)

{    vis[v]=1;    printf("%d ",v);    for(j=1;j<=n;j++)

  {

     if(a[v][j]==1&&vis[j]==0)

     {          dfs(j);

     }

  }

}
```

```
void main()
{
   printf("Enter the no of vertices:");    scanf("%d",&n);    printf("Enter the adjacency matrix");    for(i=1;i<=n;i++)
   {
      for(j=1;j<=n;j++)
      {
         scanf("%d",&a[i][j]);
      }       vis[i]=0;
   }
   printf("dfs traversal");    for(i=1;i<=n;i++)
   {       if(vis[i]==0)          dfs(i);
   }
   getch();
}
```

Output:



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>

#include <stdlib.h>


int key[20], n, m; int *ht, index; int count = 0;


void insert(int key) {     index = key % m;     while (ht[index] != -1) {         index = (index +
1) % m;

   }

   ht[index] = key;     count++;

}


void display() {     if (count == 0) {         printf("\nHash Table is empty");

       return;

   }

   printf("\nHash Table contents are:\n");     for (int i = 0; i < m; i++) {
printf("\nT[%d] --> %d", i, ht[i]);

   }

}


void main() {     printf("\nEnter the number of employee records (N): ");     scanf("%d",
&n);
```

```
    printf("\nEnter the two-digit memory locations (m) for hash table: ");     scanf("%d", &m);

    ht = (int *)malloc(m * sizeof(int));     for (int i = 0; i < m; i++)         ht[i] = -1;

    printf("\nEnter the four-digit key values (K) for %d Employee Records:\n", n);
    for (int i = 0; i < n; i++)         scanf("%d", &key[i]);

    for (int i = 0; i < n; i++) {        if (count == m) {            printf("\nHash table is full. Cannot insert record %d key", i + 1);
            break;
        }
        insert(key[i]);
    }

    display();     free(ht);
}
```

```
Enter the number of employee records (N): 5

Enter the two-digit memory locations (m) for hash table: 7

Enter the four-digit key values (K) for 5 Employee Records:
1234 5678 9201 4397 6130

Hash Table contents are:

T[0] --> -1
T[1] --> 5678
T[2] --> 1234
T[3] --> 9201
T[4] --> 4397
T[5] --> 6130
T[6] --> -1
PS E:\DSA\C\LAB-10>
```