**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**



**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**EASHAN JAIN V (1BM23CS098)**

**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 EASHAN JAIN V**(1BM23CS098)**, 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** **Dr. Kavitha Sooda**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | EXPERIMENT TITLE | **Page No.** |
| 1 | Lab program 1 | 4-6 |
| 2 | Lab program 2 | 6-9 |
| 3 | Lab program 3 | 9-16 |
| 4 | Lab program 4 | 17-20 |
| 5 | Lab program 5 | 20-25 |
| 6 | Lab program 6 | 26-35 |
| 7 | Lab program 7 | 35-40 |
| 8 | Lab program 8 | 40-43 |
| 9 | Lab program 9 | 43-47 |
| 10 | Lab program 10 | 47-49 |

**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**

1. **Pop**
2. **Display**

**The program should print appropriate messages for stack overflow, stack underflow.**

#include <stdio.h>

#include <stdlib.h>

#define STACK\_SIZE 5

int stack[STACK\_SIZE]; int top = -1;

void push(int item) { if (top == STACK\_SIZE - 1) { printf("Stack overflow\n");

} else { stack[++top] = item; printf("Item %d pushed to stack\n", item);

}

}

int pop() { if (top == -1) { printf("Stack underflow\n");

return -1; } else { printf("Item %d popped from stack\n", stack[top]);

return stack[top--];

}

}

void display() { if (top == 1) { printf("Stack is empty\n");

} else { printf("Stack contents:\n"); for (int i = 0; i

<= top; i++) { printf("%d ", stack[i]);

}

printf("\n");

}

}

int main() {

int choice, item;

while (1) {

printf("\n1: Push\n2: Pop\n3: Display\n4: Exit\n"); printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) { case 1:

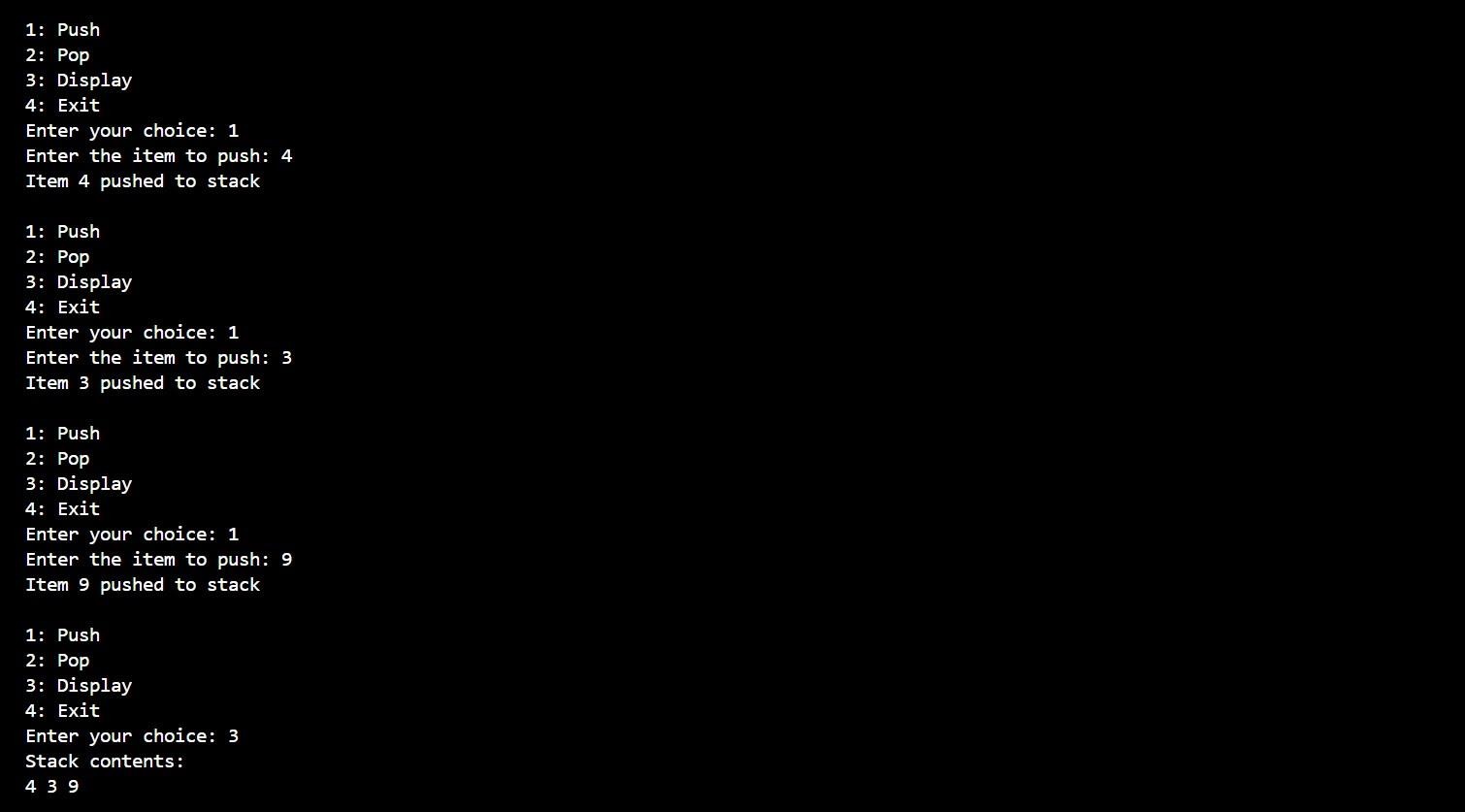
printf("Enter the item to push: ");

scanf("%d", &item); push(item); break; case 2: pop(); break; case 3: display(); break; case 4: exit(0); default: printf("Invalid choice\n");

}

}

return 0; } **Output:**



**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)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> char stack[100]; int top=-1; void push(char ele) { top++; stack[top]=ele;

} char pop()

{ return stack[(top)--]; } int pr(char op) { switch(op)

{ case '#':return 0; break; case '(':return 1; break; case '+':return 2; break; case '':return 2; break; case '\*':return 2; break; case '/':return 2; break; default:return 0; break;

}

} void main()

{ char infix[100], postfix[100]; int i=0,count=0; char ch; //clrscr(); printf("Enter your infix expression:"); scanf("%s",infix); push('#'); while

(infix[i]!='\0')

{ if (isalpha(infix[i]))

{

postfix[count]=infix[i]; count++;

} else if(infix[i]

== '(') push(infix[i]); else if(infix[i] == ')') { while (stack[top] != '(')

{ ch=pop();

postfix[count]=ch; count++;

}

pop(); /\* Removing the ( \*/

} else

{

while ((stack[top]!='#') && (pr(infix[i])<=pr(stack[top])))

{ ch=pop();

postfix[count]=ch; count++;

} push(infix[i]);

} i++;

} for(i=top;i!=0;i--)

{ if (stack[i] == '(') printf ("\n There was an issue with the expression..."); ch=pop(); postfix[count]=ch; count++;

}

for(i=0;i<count;i++)

{ printf("%c",postfix[i]);

}

}



**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>

void insert(int q[], int \*rear, int item, int QSIZE) { if (\*rear == QSIZE - 1) { printf("Queue overflow\n");

} else {

(\*rear)++;

q[\*rear] = item;

}

}

void delete(int \*front, int \*rear, int q[]) { if (\*front > \*rear) { printf("Queue underflow\n");

} else { printf("Deleted item: %d\n", q[\*front]);

(\*front)++;

}

}

void display(int \*front, int \*rear, int q[]) { if (\*front == \*rear) { printf("Queue is empty\n");

} else { printf("Queue elements: "); for (int i = \*front; i <= \*rear; i++) { printf("%d ", q[i]);

}

printf("\n");

}

}

int main() { int QSIZE = 3; int q[QSIZE]; int choice, item; int front = 0; int rear = -1; while (1) { printf("Enter your

choice: ");

scanf("%d",

&choice);

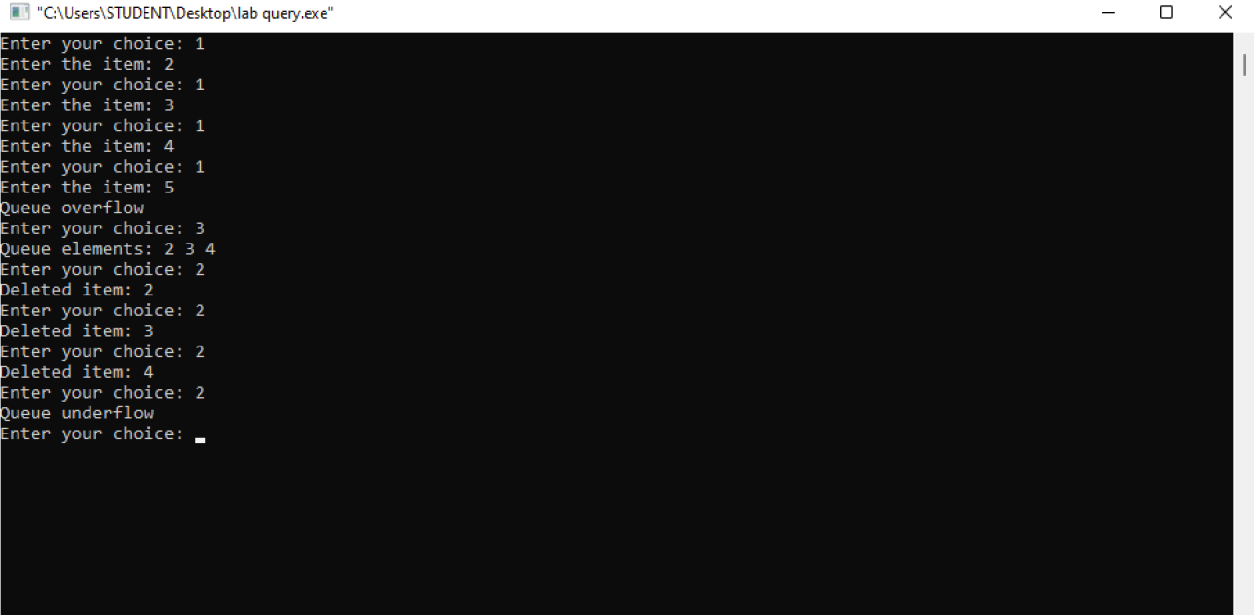
switch (choice) { case 1:

printf("Enter the item: "); scanf("%d", &item); insert(q, &rear, item, QSIZE); break; case 2: delete(&front, &rear,

q); break; case 3:

display(&front, &rear, q); break; default:

printf("Invalid choice\n");



} } return 0;

}

**LEETCODE PROBLEM:**

**3b) Remove all adjacent duplicates in a string** #include <stdlib.h>

#include <string.h>

char\* removeDuplicates(char\* s) {

int len = strlen(s); char\* stack = (char\*)malloc(len + 1); int top = -1;

for (int i = 0; i < len; i++) { if (top >= 0 && stack[top] == s[i]) {

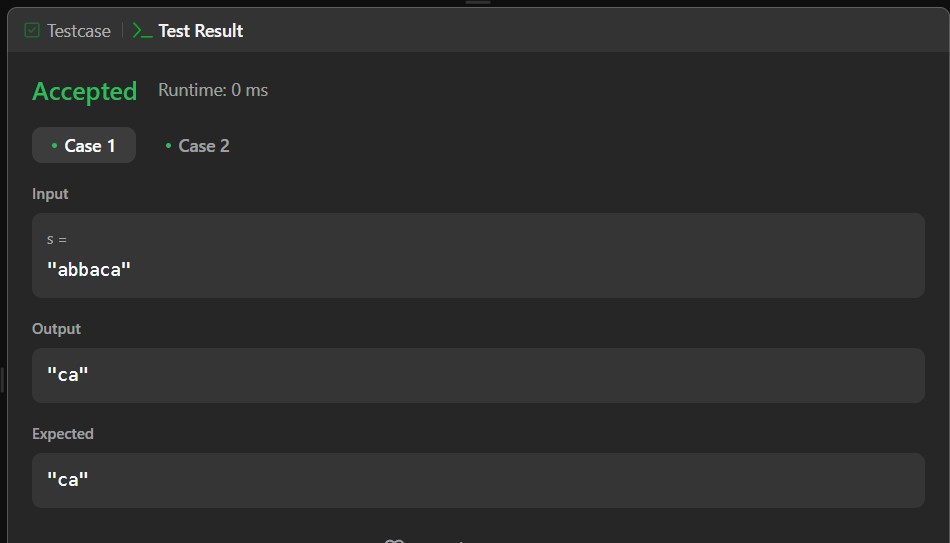
top--; } else { stack[++top] = s[i];

}

}

stack[top + 1] = '\0'; return stack;

}



**3b)WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & DisplayThe program should print appropriate messages for queue empty and queue overflow conditions. The program should be done using pass by reference only.**

#include <stdio.h>

#include <stdlib.h>

void insert(int q[], int \*rear, int \*count, int item, int QSIZE) { if (\*count >= QSIZE) { printf("Queue overflow\n");

} else {

\*rear = (\*rear + 1) % QSIZE; q[\*rear] = item;

(\*count)++;

}

}

int delete(int q[], int \*front, int \*count, int QSIZE) { int deleted\_item; if (\*count == 0) { printf("Queue underflow\n"); return -1; } else { deleted\_item = q[\*front];

\*front = (\*front + 1) % QSIZE;

(\*count)--; return deleted\_item;

}

}

void display(int q[], int \*front, int \*count, int QSIZE) {

int i;

if (\*count == 0) { printf("Queue is empty\n");

} else { printf("Queue elements: "); for (i = \*front; i < \*front + \*count; i++) { printf("%d ", q[i % QSIZE]);

} printf("\n");

}

}

int main() { int count = 0; int QSIZE = 3; int q[QSIZE]; int choice, deleted\_item, item; int front = 0; int rear = -1;

while (1) { printf("Enter your choice (1: Insert, 2: Delete, 3: Display): "); scanf("%d", &choice);

switch (choice) { case 1:

printf("Enter the item: "); scanf("%d", &item); insert(q, &rear, &count, item, QSIZE);

break; case 2:

deleted\_item = delete(q, &front, &count, QSIZE);

if (deleted\_item != -1) { printf("Deleted item is %d\n", deleted\_item);

} break; case 3:

display(q, &front, &count, QSIZE);

break; default:

printf("Invalid choice\n");

}

}

return 0;

}

**LeetCode Program- Remove Digit from Number to Maximize Result**

#include <string.h>

#include <stdlib.h>

char\* removeDigit(char\* number, char digit) { int len = strlen(number); char\* result = (char\*)malloc(len); int maxIndex = -1;

for (int i = 0; i < len; i++) { if (number[i] == digit) { if (i + 1 < len && number[i] < number[i + 1]) { maxIndex = i; break;

}

maxIndex = i;

}

}

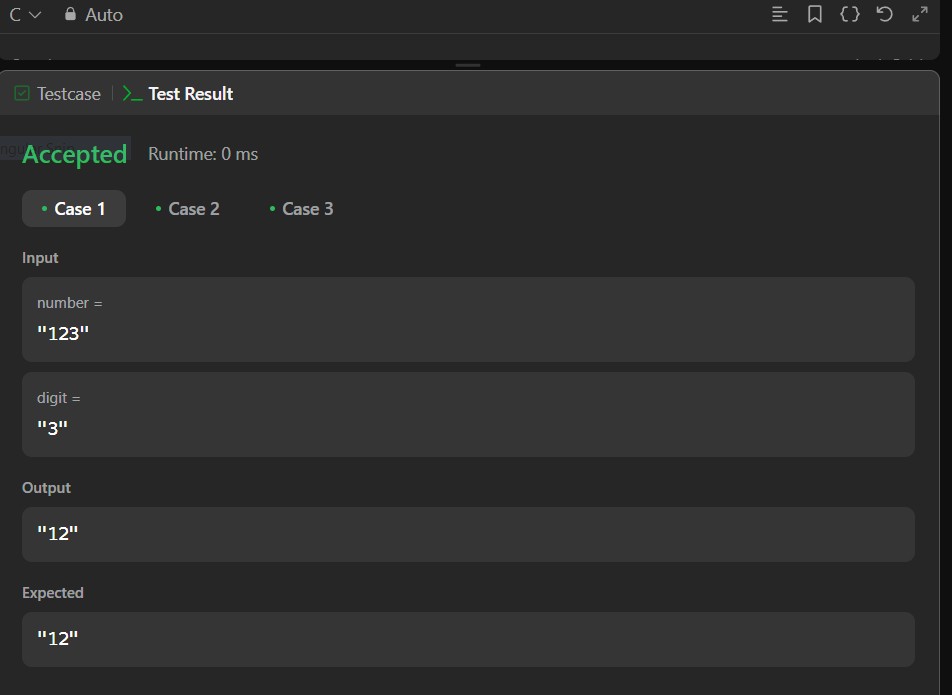
for (int i = 0, j = 0; i < len; i++) { if (i != maxIndex) { result[j++] = number[i];

}

}

result[len - 1] = '\0'; return result;

}



**Lab Program-4:**

**WAP to Implement Singly Linked List with following operations**

1. **Create a linked list.**
2. **Insertion of a node at first position, at any position and at end of list.**
3. **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 createLinkedList(int data[], int n) { for (int i = 0; i < n; i++) { int value = data[i]; struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); newNode->data = value; newNode->next = NULL;

if (head == NULL) {

head = newNode;

} else { struct Node\* temp

= head; while (temp->next !=

NULL) { temp = temp-

>next;

}

temp->next = newNode;

}

}

}

void insertAtBeginning(int data) { struct Node\* newNode = (struct

Node\*)malloc(sizeof(struct Node)); newNode->data = data; newNode-

>next = head; head = newNode;

}

void insertAtEnd(int data) { struct Node\* newNode = (struct

Node\*)malloc(sizeof(struct Node)); newNode->data = data; newNode-

>next = NULL;

if (head == NULL) { head = newNode;

} else { struct Node\* temp = head; while (temp->next !=

NULL) { temp = temp-

>next;

}

temp->next = newNode;

}

}

void insertAtPosition(int data, int position) { struct Node\* newNode =

(struct Node\*)malloc(sizeof(struct Node)); newNode->data = data;

if (position == 0) { newNode->next = head; head = newNode;

return;

}

struct Node\* temp = head; for (int i = 0; temp !=

NULL && i < position - 1; i++) { temp = temp-

>next;

}

if (temp == NULL) { printf("Position out of bounds\n"); free(newNode);

} else { newNode->next = temp->next; temp->next = newNode;

}

}

void displayList() { struct

Node\* temp = head; while

(temp != NULL) { printf("%d -> ", temp->data); temp = temp->next;

}

printf("NULL\n");

}

int main() { int data[] = {10, 20, 30};

int n = sizeof(data) / sizeof(data[0]);

createLinkedList(data, n);

displayList();

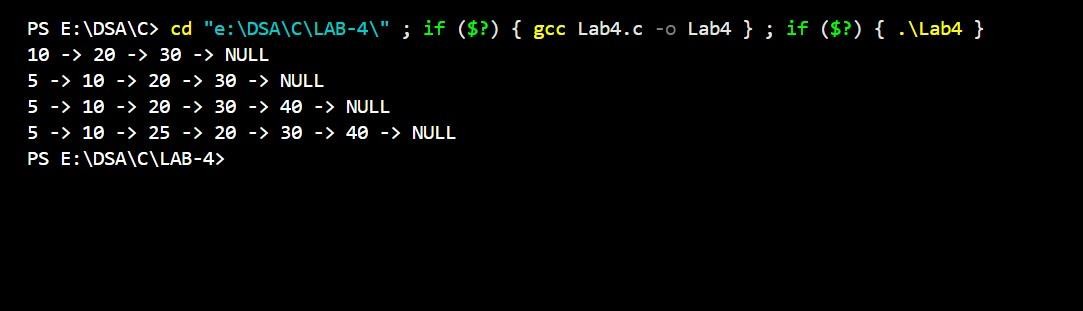
insertAtBeginning(5); displayList(); insertAtEnd(40); displayList();

insertAtPosition(25, 2);

displayList();

return 0;

}



**Lab Program-5:**

**WAP to Implement Singly Linked List with following operations**

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

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data; struct

Node\* next;

};

struct Node\* createLinkedList(); void deleteFirst(struct Node\*\* head); void deleteSpecified(struct Node\*\* head, int value); void deleteLast(struct Node\*\* head); void displayLinkedList(struct Node\* head);

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

while (1) { printf("\n--- Singly Linked List Operations ---\n"); printf("1. Create Linked List\n"); printf("2. Delete First Element\n"); printf("3. Delete Specified Element\n"); printf("4.

Delete Last Element\n"); printf("5. Display Linked List\n"); printf("6. Exit\n"); printf("Enter your choice: "); scanf("%d", &choice);

switch (choice) { case 1:

head = createLinkedList();

break; case 2:

deleteFirst(&head); break; case 3: printf("Enter the value to delete: "); scanf("%d", &value); deleteSpecified(&head, value);

break; case 4:

deleteLast(&head); break; case 5: displayLinkedList(head);

break; case 6:

printf("Exiting program.\n");

exit(0);

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

}

}

return 0;

}

struct Node\* createLinkedList() { struct Node \*head = NULL,

\*temp = NULL, \*newNode = NULL;

int data;

printf("Enter elements of the list (-1 to stop):\n"); while (1) { printf("Enter data: "); scanf("%d",

&data); if (data == -1) break;

newNode = (struct Node\*)malloc(sizeof(struct Node)); newNode->data

= data;

newNode->next = NULL;

if (head == NULL) { head = newNode;

} else { temp->next = newNode;

}

temp = newNode;

} return head;

}

void deleteFirst(struct Node\*\* head) { if (\*head == NULL) { printf("List is empty. Nothing to delete.\n"); return;

}

struct Node\* temp = \*head; \*head = (\*head)->next; free(temp); printf("First element deleted.\n");

}

void deleteSpecified(struct Node\*\* head, int value) { if (\*head == NULL) { printf("List is empty. Nothing to delete.\n");

return;

}

struct Node \*temp = \*head, \*prev = NULL;

if (temp != NULL && temp->data == value) {

\*head = temp->next; free(temp); printf("Element %d deleted.\n", value);

return;

}

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

}

if (temp == NULL) { printf("Element %d not found in the list.\n", value); return;

}

prev->next = temp->next; free(temp); printf("Element %d deleted.\n", value);

}

void deleteLast(struct Node\*\* head) { if (\*head == NULL) { printf("List is empty. Nothing to delete.\n"); return;

}

struct Node \*temp = \*head, \*prev = NULL;

if (temp->next == NULL) {

\*head = NULL; free(temp);

printf("Last element deleted.\n");

return;

}

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

>next;

}

prev->next = NULL; free(temp); printf("Last element deleted.\n");

}

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

}

printf("Linked List: "); struct

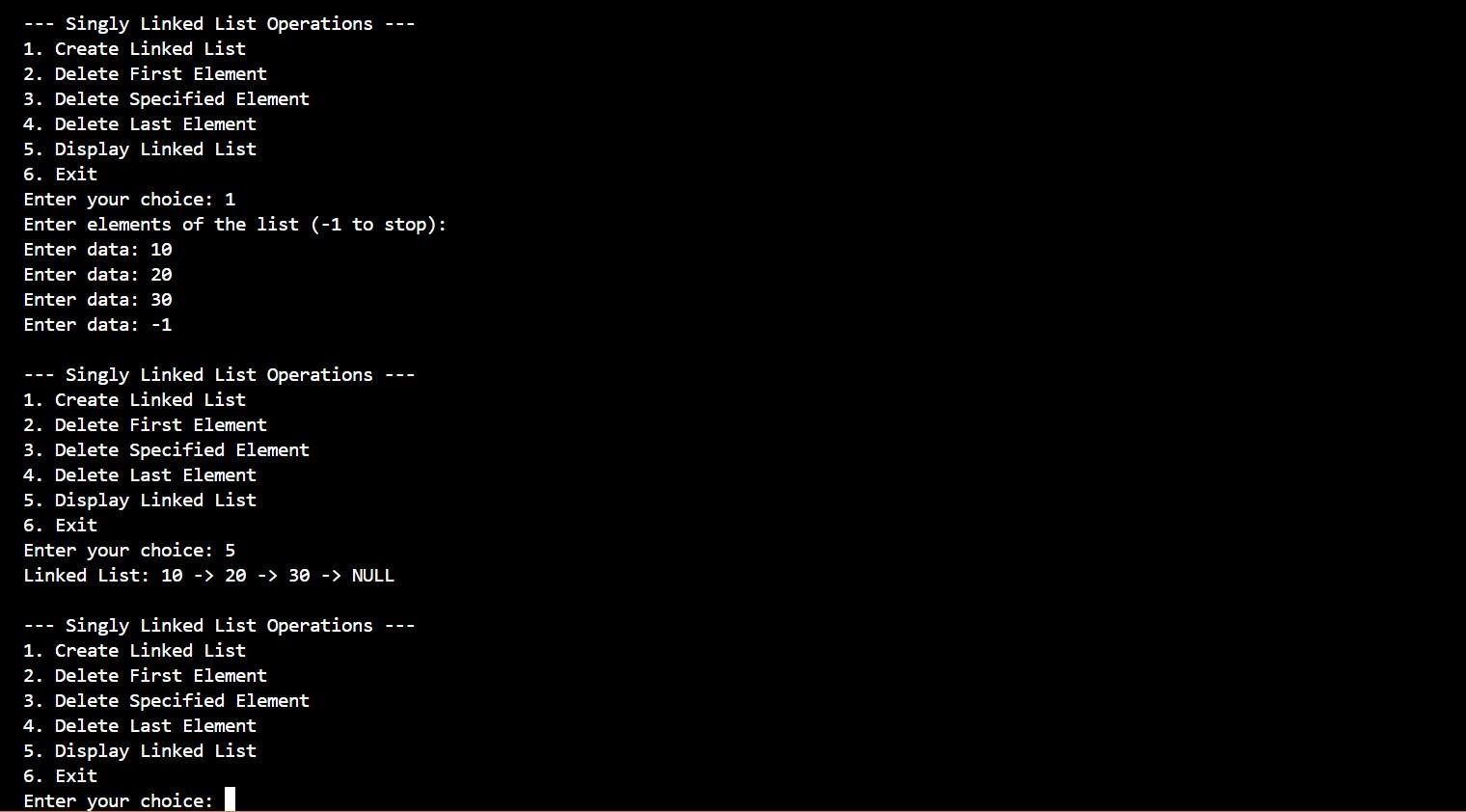
Node\* temp = head; while

(temp != NULL) { printf("%d -> ", temp->data); temp = temp->next;

}

printf("NULL\n");

}



**Lab Program-6:**

**6a) 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>

struct node

{ int data; struct node

\*next;

};

typedef struct node\* NODE;

NODE getnode()

{

NODE ptr; ptr=(NODE)malloc(sizeof(struct node)); if(ptr==NULL)

{ printf("node not created");

return NULL;

} return ptr;

}

NODE insert\_beg(NODE first,int item)

{

NODE new\_node; new\_node=getnode(); new\_node->data=item; new\_node->next=NULL;

if(first==NULL) return new\_node; new\_node-

>next=first; return new\_node;

}

void display(NODE first)

{

NODE temp; if(first==NULL) {

printf("Linked list is empty\n");

} temp=first;

while(temp!=NULL)

{ printf("%d ",temp->data);

temp=temp->next;

}

}

NODE reverse(NODE first)

{

NODE current,temp; current=NULL; if(first==NULL) return NULL; while(first!=NULL)

{ temp=first; first=first-

>next; temp->next=current; current=temp;

} return current;

}

void sort(NODE first)

{

NODE temp1,temp2; temp1=first;

//temp2=first->next; while(temp1-

>next!=NULL)

{

temp2=temp1->next;

while(temp2!=NULL)

{

if(temp1->data>=temp2->data)

{ int x=temp1->data; temp1>data=temp2->data; temp2->data=x;

}

temp2=temp2->next;

}

temp1=temp1->next;

}

}

NODE concatenate(NODE first1,NODE first2)

{

NODE last1; if(first1==NULL

&& first2==NULL) return NULL; if(first1==NULL) return first2; if(first2==NULL) return first1; last1=first1; while(last1->next!=NULL) last1=last1->next; last1>next=first2; return first1;

}

void main()

{

NODE first1=NULL; NODE first2=NULL; int choice,item,pos,value; while(1)

{

printf("\nEnter your choice\n 1.insert\n 2.reverse\n 3.sort\n 4.concatenate\n 5.display\n");

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

{ case 1: { printf("Enter the item:"); scanf("%d",&item); first1=insert\_beg(first1,item); first2=insert\_beg(first2,item); break;

} case 2: { first1=reverse(first1);

break;

} case 3:

{ sort(first1);

break;

} case 4:

{

first1=concatenate(first1,first2); break;

} case 5: { display(first1);

break;

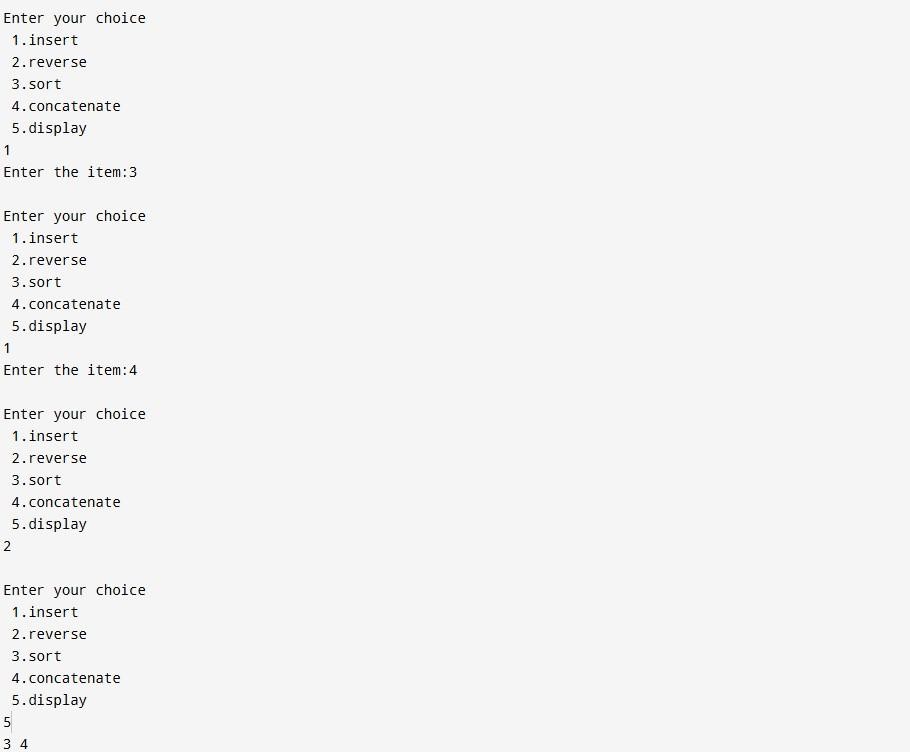
} default: { printf("exiting\n"); exit(0);

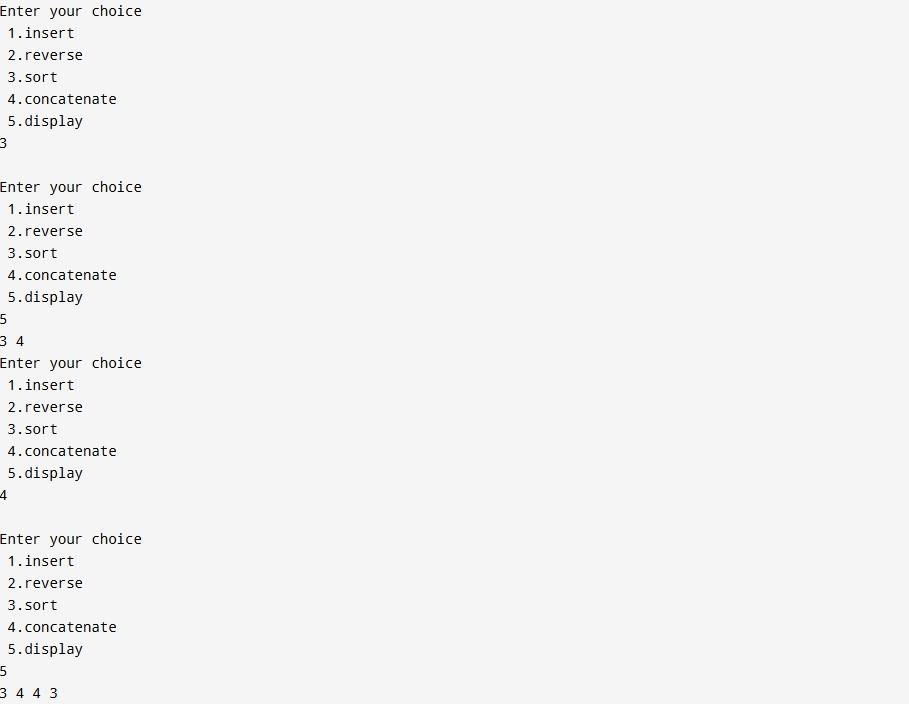
}

}

}

}





**6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.**

#include<stdio.h>

#include<stdlib.h>

struct node

{ int data; struct node \*next;

};

typedef struct node\* NODE;

NODE getnode()

{

NODE ptr; ptr=(NODE)malloc(sizeof(struct node)); if(ptr==NULL)

{ printf("node not created");

return NULL;

} return ptr;

}

NODE insert\_end(NODE first,int item)

{

NODE new\_end,current; new\_end=getnode(); new\_end->data=item; new\_end->next=NULL;

if(first==NULL) return new\_end; current=first; while(current->next!=NULL) current=current->next; current->next=new\_end;

return first;

}

NODE delete\_end(NODE first)

{

NODE prev,last; if(first==NULL)

{ printf("Linked list is empty\n");

return NULL;

}

prev=NULL;

last=first; while(last-

>next!=NULL)

{ prev=last; last=last-

>next;

}

prev->next=NULL;

free(last); return first;

}

void display(NODE first)

{

NODE temp; if(first==NULL)

{ printf("Linked list is empty\n");

} temp=first;

while(temp!=NULL)

{

printf(" %d ",temp->data); temp=temp->next;

}

}

void main()

{

NODE first=NULL; int choice,item,pos,value; while(1)

{

printf("\n Enter your choice\n 1.insert\n 2.delete\n 0.display\n");

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

{ case 1: { printf("Enter the item:"); scanf("%d",&item); first=insert\_end(first,item); break;

} case 2: { first=delete\_end(first);

break;

} case 0:

{ display(first); break;

} default: { printf("exiting\n");

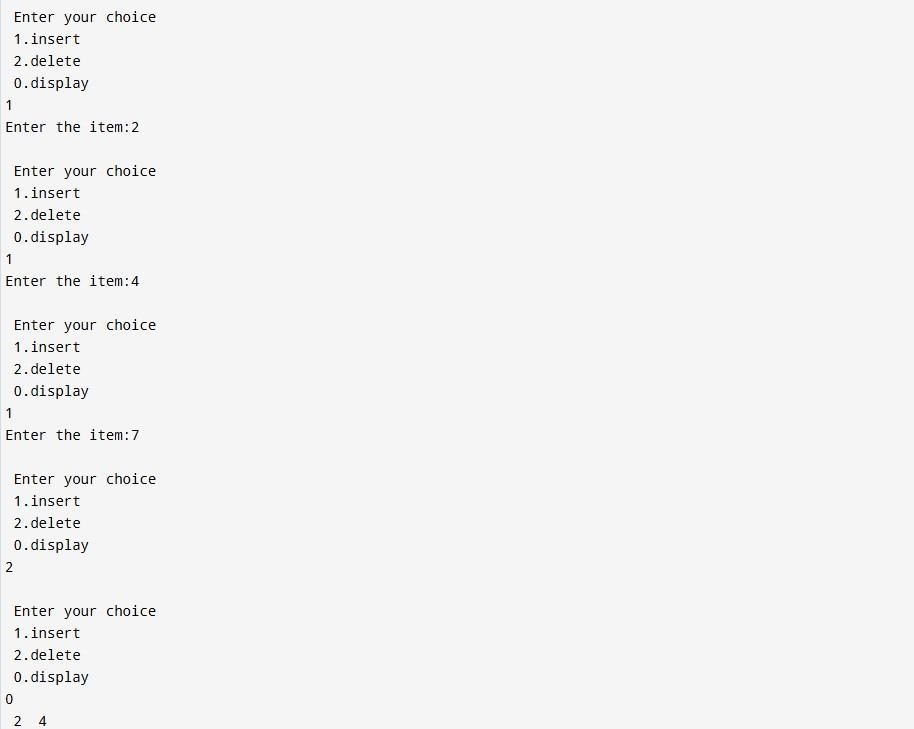
exit(0);

}

}

}

}



**Lab program-7:**

**WAP to Implement doubly link list with primitive operations**

1. **Create a doubly linked list.**
2. **Insert a new node to the left of the node.**
3. **Delete the node based on a specific value**
4. **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(int val) { struct Node\* n = (struct Node\*)malloc(sizeof(struct Node)); n->data = val; n->prev = n->next = NULL; return n;

}

void insertEnd(struct Node\*\* head, int val) { struct Node\* n = createNode(val);

if (!\*head) { \*head = n; return;

}

struct Node\* t = \*head; while (t->next) t = t->next; t->next = n; n->prev = t;

}

void insertLeft(struct Node\*\* head, int target, int val) { struct Node\* t = \*head; while (t && t->data != target) t

= t->next; if (!t) return;

struct Node\* n = createNode(val); n->next = t; n->prev = t->prev;

if (t->prev) t->prev->next = n; else \*head = n; t->prev = n;

}

void deleteNode(struct Node\*\* head, int val) { struct Node\* t = \*head; while (t && t->data != val) t = t->next; if (!t) return; if (t->prev) t>prev->next = t->next; else \*head = t->next; if (t->next) t->next->prev = t->prev; free(t);

}

void display(struct Node\* head) { while (head) { printf("%d <->

", head->data); head = head-

>next;

}

printf("NULL\n");

}

int main() { struct Node\* dll = NULL;

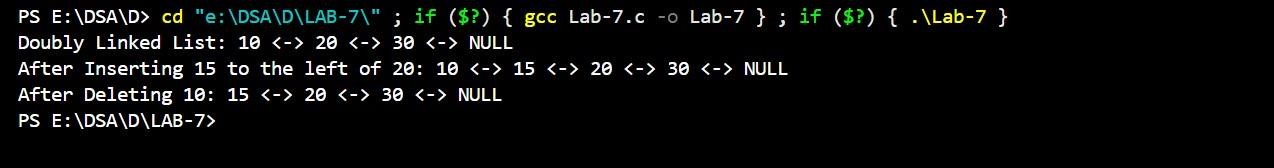
insertEnd(&dll, 10); insertEnd(&dll, 20); insertEnd(&dll, 30); printf("Doubly Linked List: "); display(dll);

insertLeft(&dll, 20, 15); printf("After

Inserting 15 to the left of 20: "); display(dll); deleteNode(&dll, 10); printf("After Deleting 10: "); display(dll);

return 0;

}



**Leetcode Program-**

#include <stdio.h>

#include <stdlib.h>

struct ListNode {

int val;

struct ListNode\*

next;

};

struct ListNode\*

middleNode(struct ListNode\* head) { struct ListNode\*

slow = head; struct ListNode\* fast

= head;

while (fast != NULL

&& fast->next != NULL) { slow = slow-

>next; fast = fast->next-

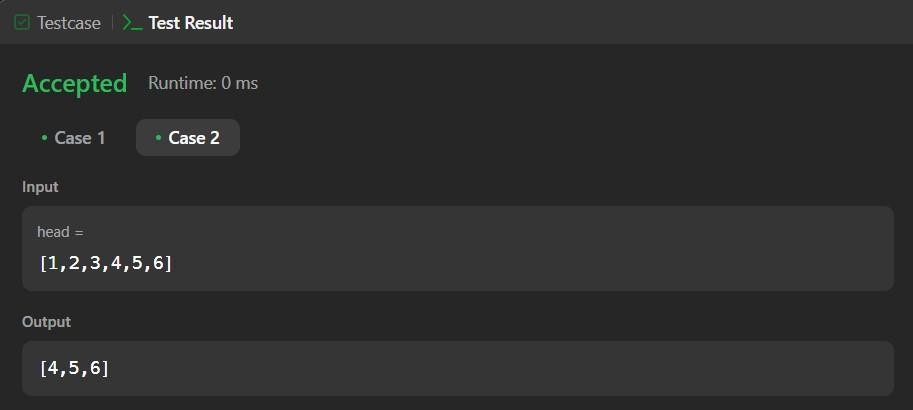
>next;

}

return slow;

}

`



**Lab program-8: Write a program**

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

#include<stdio.h>

#include<stdlib.h>

struct BST {

int data; struct

BST \*left; struct

BST \*right;

};

typedef struct BST\* NODE;

NODE create()

{

NODE temp; temp =

(NODE)malloc(sizeof(struct BST)); printf("Enter the item: "); scanf("%d",

&temp->data); temp->left = temp->right =

NULL; return temp;

}

void insert(NODE root, NODE temp)

{ if (root->data < temp->data)

{ if (root->right != NULL) insert(root->right, temp); else root->right = temp;

}

else { if (root->left != NULL) insert(root->left, temp); else root->left = temp;

}

}

void preorder(NODE root)

{ if (root != NULL)

{ printf("%d ", root->data); preorder(root->left); preorder(root->right);

}

}

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);

}

}

int main()

{

NODE root = create(); NODE temp;

int choice;

while (1)

{

printf("\nEnter your choice\n1. Insert\n2. Preorder\n3. Inorder\n4. Postorder\n5. Exit\n"); scanf("%d", &choice);

switch(choice)

{ case 1:

temp = create(); insert(root, temp); break; case 2: printf("Preorder traversal: "); preorder(root); printf("\n"); break; case 3: printf("Inorder traversal: "); inorder(root); printf("\n"); break;

case 4:

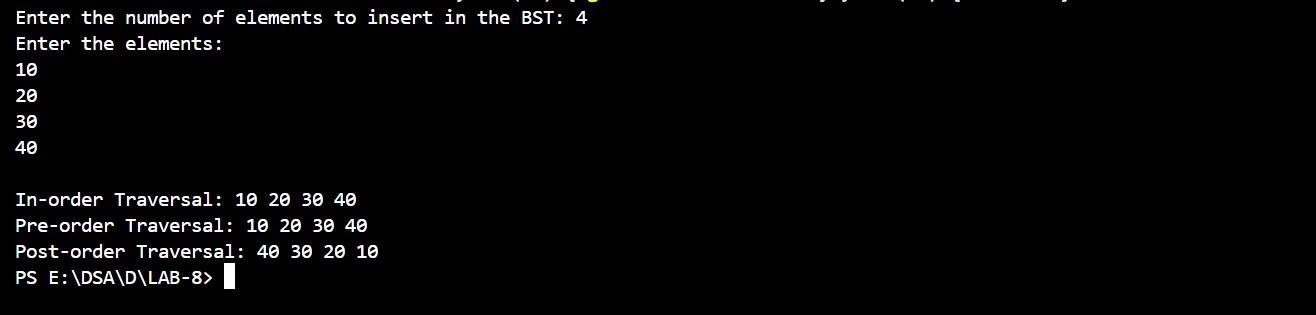
printf("Postorder traversal: "); postorder(root); printf("\n"); break; case 5: exit(0); default: printf("Invalid choice! Please try again.\n");

}

}

return 0;

}



**LAB PROGRAM 9-**

**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)

{

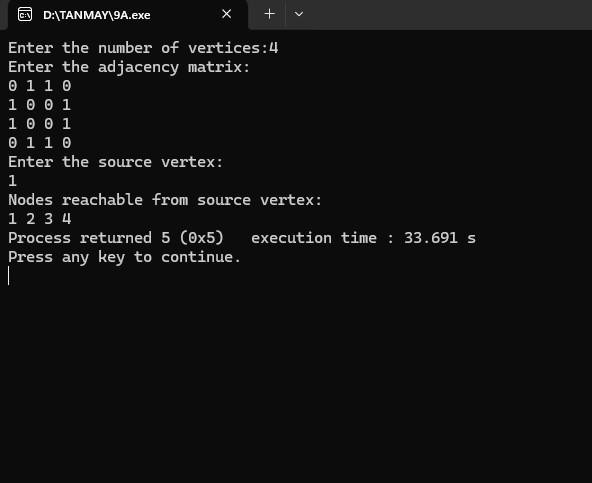
vis[i]=1; r=r+1;

q[r]=i;

}

} f=f+1; }

}



**9b) Write a program to check whether 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]);

}

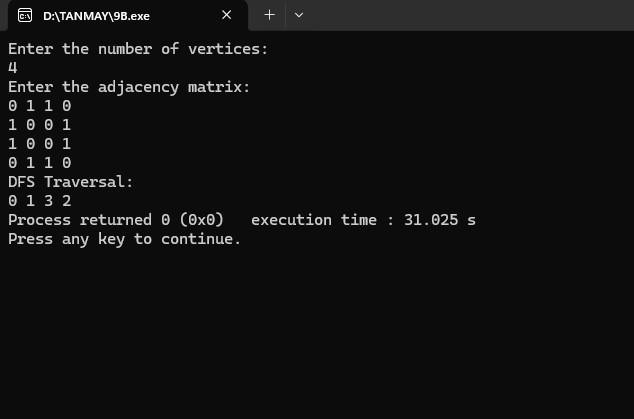
vis[i]=0;

} printf("dfs traversal");

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

{ if(vis[i]==0) dfs(i); } getch();

}



**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);

}

