**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**Chethana C (1BM23CS077)**

**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 Chethana C**(1BM23CS077)**, 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**

Associate Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 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 |  |
| 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) |  |
| 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  3b ) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete &amp; Display  The program should print appropriate messages for queue empty and queue overflow conditions |  |
| 4 | WAP to Implement Singly Linked List with following operations  a) Create a linked list.  b) Insertion of a node at first position, at any position and at  end of list.  Display the contents of the linked list. |  |
| 5 | WAP to Implement Singly Linked List with following operations  a) Create a linked list.  b) Deletion of first element, specified element and last  element in the list.  c) Display the contents of the linked list. |  |
| 6 | 6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.  6b) WAP to Implement Single Link List to simulate Stack &amp; Queue Operations. |  |
| 7 | WAP to Implement doubly link list with primitive operations  a) Create a doubly linked list.  b) Insert a new node to the left of the node.  c) Delete the node based on a specific value  d) Display the contents of the list |  |
| 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  c) To display the elements in the tree. |  |
| 9 | 9a) Write a program to traverse a graph using BFS method.  9b) Write a program to check whether given graph is connected or not  using DFS method. |  |
| 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 -&gt; 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. |  |

**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!!!");

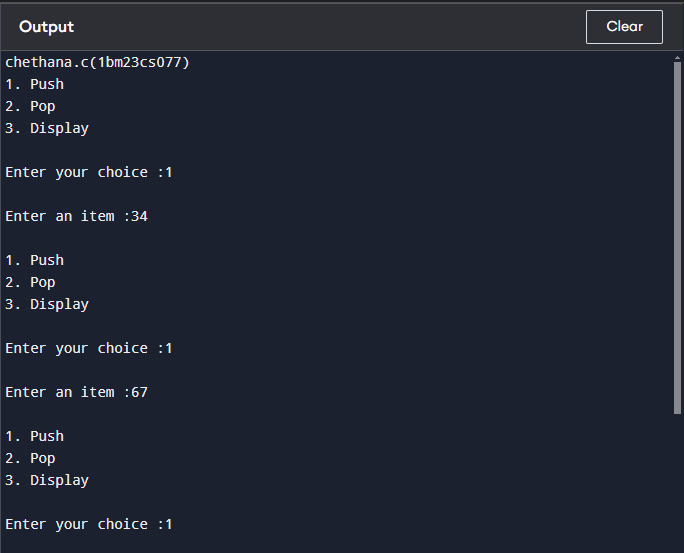
exit(0);

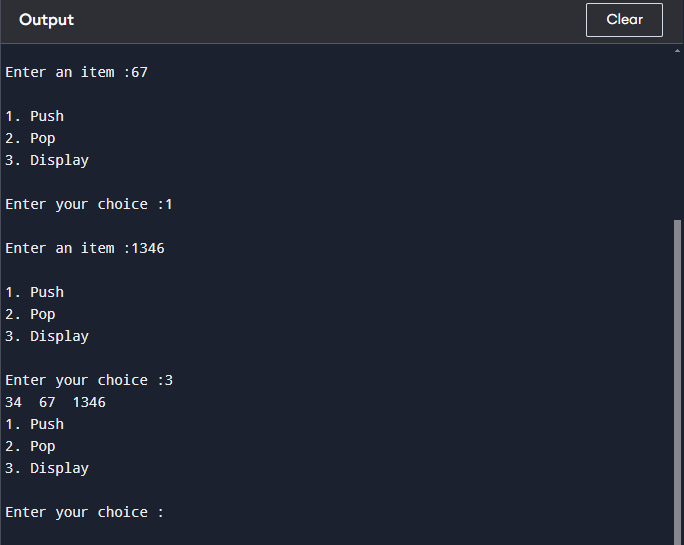
}

}

}

**Output:**





**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>

#include <string.h>

#define MAX 100

typedef struct {

char data[MAX];

int top;

} Stack;

void push(Stack \*stack, char value);

char pop(Stack \*stack);

char peek(Stack \*stack);

int isEmpty(Stack \*stack);

int precedence(char operator);

int isOperator(char symbol);

void infixToPostfix(const char \*infix, char \*postfix);

int main() {

char infix[MAX], postfix[MAX];

printf("Enter a valid parenthesized infix expression: ");

scanf("%s", infix);

infixToPostfix(infix, postfix);

printf("Postfix expression: %s\n", postfix);

return 0;

}

void push(Stack \*stack, char value) {

if (stack->top == MAX - 1) {

printf("Stack overflow\n");

exit(1);

}

stack->data[++stack->top] = value;

}

char pop(Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack underflow\n");

exit(1);

}

return stack->data[stack->top--];

}

char peek(Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack is empty\n");

exit(1);

}

return stack->data[stack->top];

}

int isEmpty(Stack \*stack) {

return stack->top == -1;

}

int precedence(char operator) {

switch (operator) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

default:

return 0;

}

}

int isOperator(char symbol) {

return symbol == '+' || symbol == '-' || symbol == '\*' || symbol == '/';

}

void infixToPostfix(const char \*infix, char \*postfix) {

Stack stack;

stack.top = -1;

int i = 0, j = 0;

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

if (isalnum(infix[i])) {

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

} else if (infix[i] == '(') {

push(&stack, infix[i]);

} else if (infix[i] == ')') {

while (!isEmpty(&stack) && peek(&stack) != '(') {

postfix[j++] = pop(&stack);

}

pop(&stack);

} else if (isOperator(infix[i])) {

while (!isEmpty(&stack) && precedence(peek(&stack)) >= precedence(infix[i])) {

postfix[j++] = pop(&stack);

}

push(&stack, infix[i]);

}

i++;

}

while (!isEmpty(&stack)) {

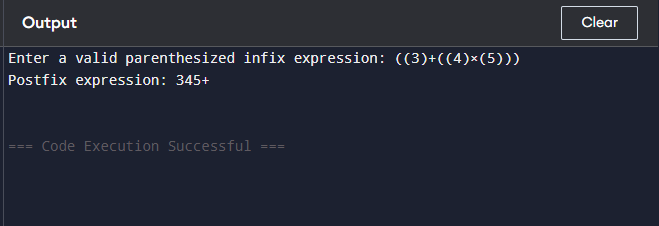
postfix[j++] = pop(&stack);

}

postfix[j] = '\0';

}

**Output:**



**Lab program**  **3a.**

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

#include <stdio.h>

#define MAX 5

int queue[MAX];

int front = -1, rear = -1;

void insert(int value) {

if (rear == MAX - 1) {

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

return;

}

if (front == -1) front = 0;

queue[++rear] = value;

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

}

void delete() {

if (front == -1 || front > rear) {

printf("Queue Underflow! The queue is empty.\n");

return;

}

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

front++;

if (front > rear) {

front = rear = -1;

}

}

void display() {

if (front == -1 || front > rear) {

printf("The queue is empty.\n");

return;

}

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 Operations:\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Display\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);

insert(value);

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

printf("Exiting...\n");

return 0;

default:

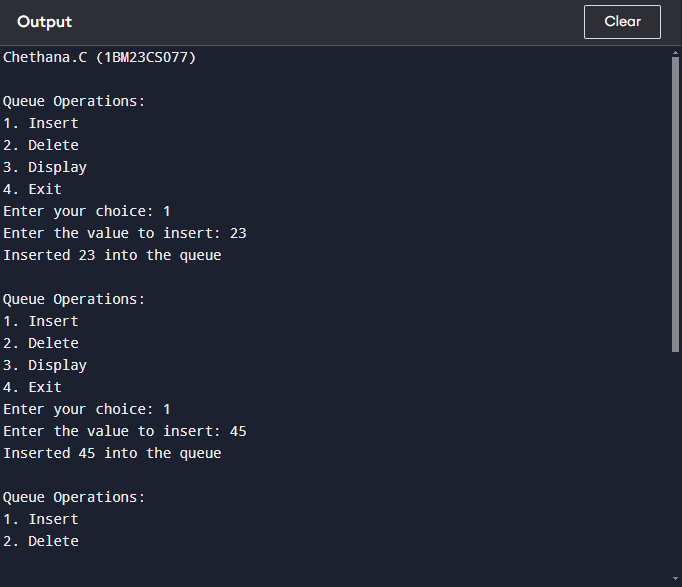
printf("Invalid choice! Please try again.\n");

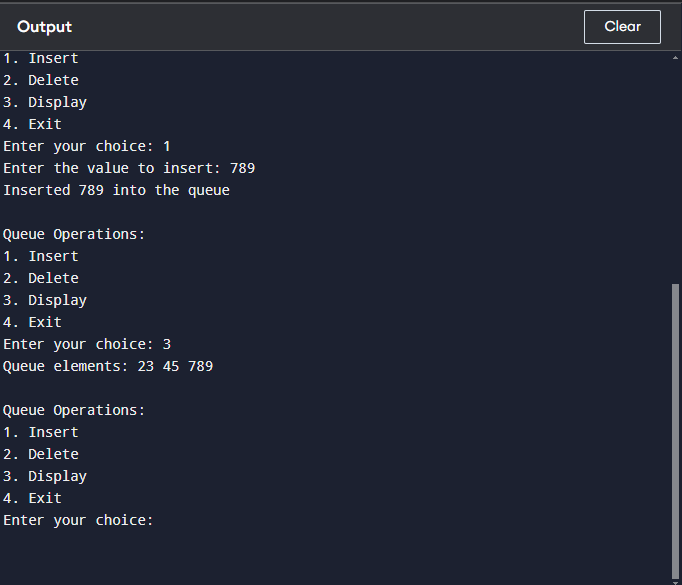
}

}

}

**Output:**





**Lab program**  **3b.**

**WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete &amp; 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, rear = -1;

void insert(int value) {

if ((front == 0 && rear == MAX - 1) || (rear == (front - 1) % (MAX - 1))) {

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

return;

}

if (front == -1) {

front = rear = 0;

} else if (rear == MAX - 1 && front != 0) {

rear = 0;

} else {

rear++;

}

queue[rear] = value;

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

}

void delete() {

if (front == -1) {

printf("Queue Underflow! The queue is empty.\n");

return;

}

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

if (front == rear) {

front = rear = -1;

} else if (front == MAX - 1) {

front = 0;

} else {

front++;

}

}

void display() {

if (front == -1) {

printf("The queue is empty.\n");

return;

}

printf("Queue elements: ");

if (rear >= front) {

for (int i = front; i <= rear; i++) {

printf("%d ", queue[i]);

}

} else {

for (int i = front; i < MAX; i++) {

printf("%d ", queue[i]);

}

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

printf("%d ", queue[i]);

}

}

printf("\n");

}

int main() {

printf("Chethana.C (1BM23CS077)\n");

int choice, value;

while (1) {

printf("\nCircular Queue Operations:\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Display\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);

insert(value);

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

printf("Exiting...\n");

return 0;

default:

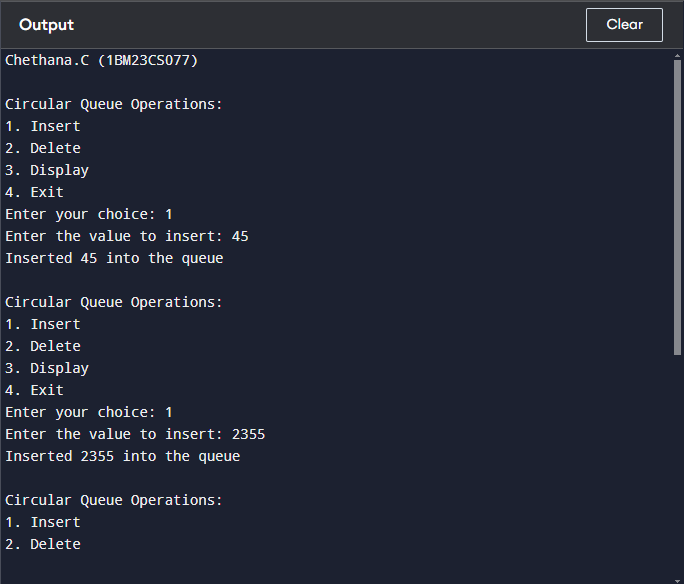
printf("Invalid choice! Please try again.\n");

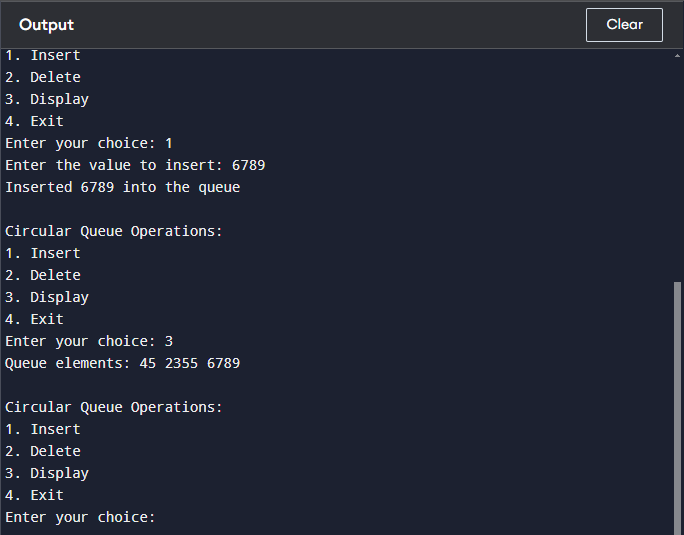
}

}

}

**Output:**





**Leet code challenge:**

[**Implement Queue using Stacks**](https://leetcode.com/problems/implement-queue-using-stacks/)

class MyQueue:

def \_\_init\_\_(self):

self.stack\_in = []

self.stack\_out = []

def push(self, x):

self.stack\_in.append(x)

def pop(self):

self.\_transfer()

return self.stack\_out.pop()

def peek(self):

self.\_transfer()

return self.stack\_out[-1]

def empty(self):

return not self.stack\_in and not self.stack\_out

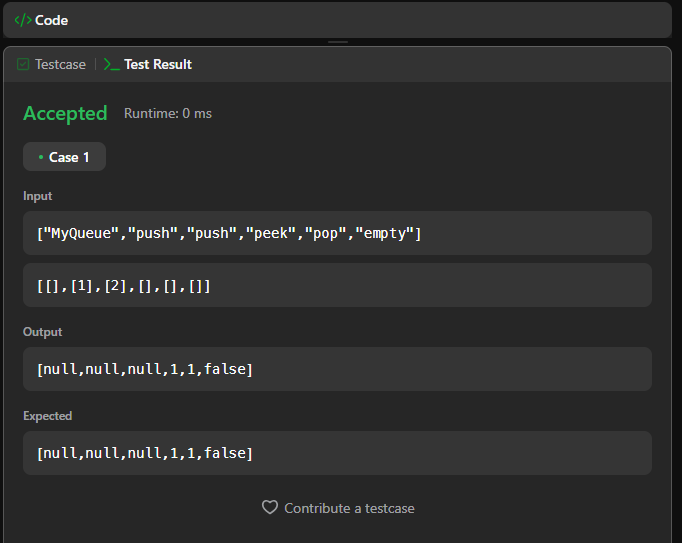
def \_transfer(self):

if not self.stack\_out:

while self.stack\_in:

self.stack\_out.append(self.stack\_in.pop())

**Output:**

****

**Lab program**  **4**

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

**a) Create a linked list.**

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

**Display the contents of the linked list.**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

Node\* createList() {

return NULL;

}

void insertAtFirst(Node\*\* head, int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = data;

newNode->next = \*head;

\*head = newNode;

}

void insertAtEnd(Node\*\* head, int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) {

\*head = newNode;

return;

}

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

void insertAtPosition(Node\*\* head, int data, int position) {

if (position <= 0) {

printf("Invalid position.\n");

return;

}

if (position == 1) {

insertAtFirst(head, data);

return;

}

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = data;

Node\* temp = \*head;

for (int i = 1; i < position - 1 && temp != NULL; i++) {

temp = temp->next;

}

if (temp == NULL) {

printf("Position out of range.\n");

free(newNode);

return;

}

newNode->next = temp->next;

temp->next = newNode;

}

void displayList(Node\* head) {

if (head == NULL) {

printf("List is empty.\n");

return;

}

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

Node\* head = createList();

printf("Chethana.C(1BM23CS077)\n");

insertAtFirst(&head, 7);

insertAtFirst(&head, 7);

insertAtEnd(&head, 1);

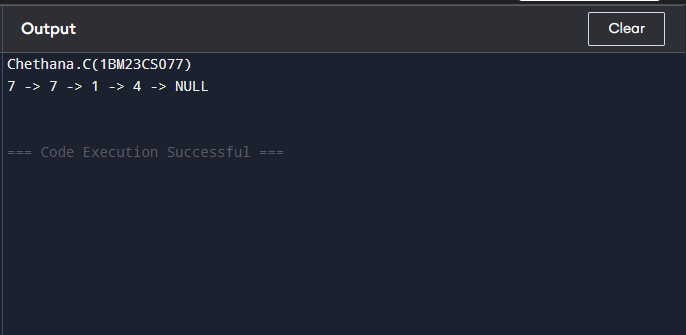
insertAtEnd(&head,4);

displayList(head);

return 0;

}

**Output:**

****

**Leetcode Challenge:  
Remove digit from number to maximize result**

#include <stdio.h>

#include <string.h>

char\* removeDigit(char\* number, char digit) {

int len = strlen(number);

for (int i = 0; i < len - 1; i++) {

if (number[i] == digit && number[i] < number[i + 1]) {

memmove(&number[i], &number[i + 1], len - i);

return number;

}

}

for (int i = len - 1; i >= 0; i--) {

if (number[i] == digit) {

memmove(&number[i], &number[i + 1], len - i);

return number;

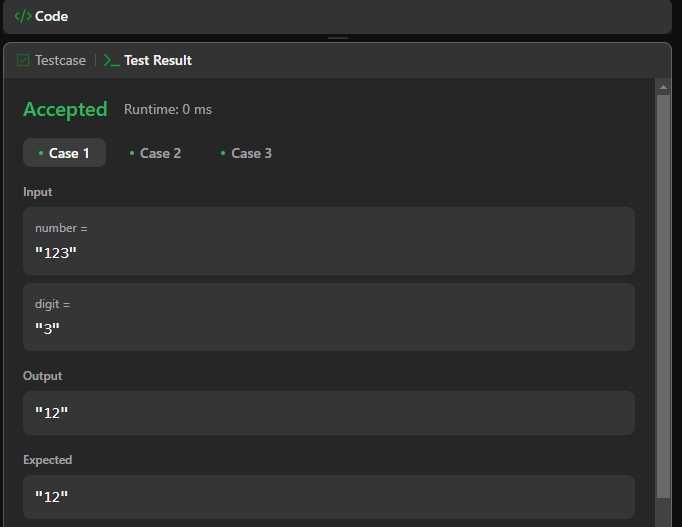
}

}

return number;

}

**Output:**



**Lab Program 5:**

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

**a) Create a linked list.**

**b) Deletion of first element, specified element and last**

**element in the list.**

**c) Display the contents of the linked list.**

#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 addNode(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;

}

void deleteFirst(struct Node\*\* head) {

if (\*head == NULL) {

printf("List is empty.\n");

return;

}

struct Node\* temp = \*head;

\*head = (\*head)->next;

free(temp);

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

}

void deleteLast(struct Node\*\* head) {

if (\*head == NULL) {

printf("List is empty.\n");

return;

}

if ((\*head)->next == NULL) {

free(\*head);

\*head = NULL;

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

return;

}

struct Node\* temp = \*head;

while (temp->next && temp->next->next != NULL) {

temp = temp->next;

}

free(temp->next);

temp->next = NULL;

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

}

void deleteSpecific(struct Node\*\* head, int value) {

if (\*head == NULL) {

printf("List is empty.\n");

return;

}

struct Node\* temp = \*head;

if (temp->data == value) {

\*head = temp->next;

free(temp);

printf("Element %d deleted.\n", value);

return;

}

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

temp = temp->next;

}

if (temp->next) {

struct Node\* toDelete = temp->next;

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

free(toDelete);

printf("Element %d deleted.\n", value);

} else {

printf("Element %d not found in the list.\n", value);

}

}

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 choice, value;

printf("Chethana.C(1BM23CS077)\n");

while (1) {

printf("\nMenu:\n");

printf("1. Add a node\n");

printf("2. Delete first element\n");

printf("3. Delete last element\n");

printf("4. Delete a specific element\n");

printf("5. Display list\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

if (choice == 1) {

printf("Enter the value to add: ");

scanf("%d", &value);

addNode(&head, value);

}

else if (choice == 2) {

deleteFirst(&head);

}

else if (choice == 3) {

deleteLast(&head);

}

else if (choice == 4) {

printf("Enter the value to delete: ");

scanf("%d", &value);

deleteSpecific(&head, value);

}

else if (choice == 5) {

display(head);

}

else if (choice == 6) {

printf("Exiting...\n");

return 0;

}

else {

printf("Invalid choice. Try again.\n");

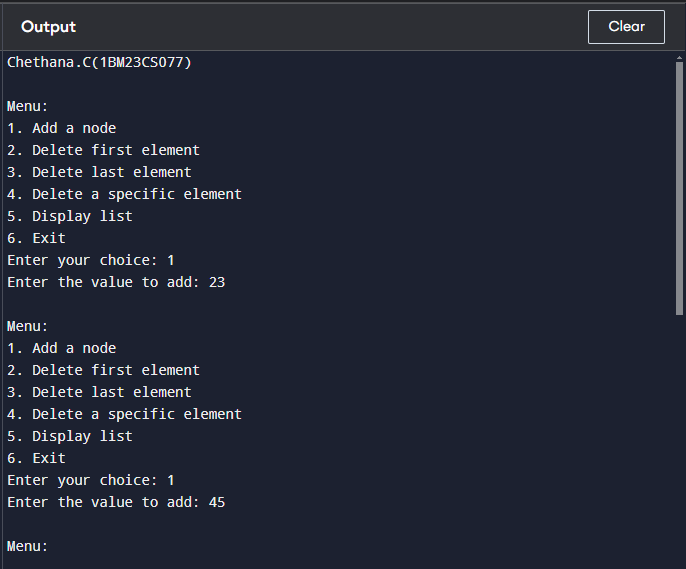
}

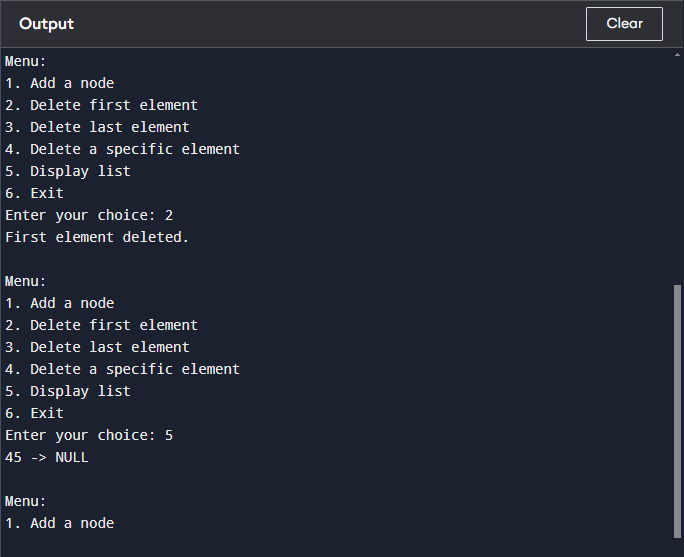
}

return 0;

}

**Output:**





**Lab Program 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;

};

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void addNode(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;

}

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 sortList(struct Node\* head) {

struct Node \*i, \*j;

int temp;

for (i = head; i != NULL; i = i->next) {

for (j = i->next; j != NULL; j = j->next) {

if (i->data > j->data) {

temp = i->data;

i->data = j->data;

j->data = temp;

}

}

}

}

void reverseList(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;

}

void concatenateLists(struct Node\*\* head1, struct Node\* head2) {

if (\*head1 == NULL) {

\*head1 = head2;

return;

}

struct Node\* temp = \*head1;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = head2;

}

int main() {

struct Node\* list1 = NULL;

struct Node\* list2 = NULL;

int choice, value;

printf("Chethana.C(1BM23CS077)\n");

while (1) {

printf("\nOptions:\n");

printf("1. Insert in List 1\n");

printf("2. Insert in List 2\n");

printf("3. Sort List 1\n");

printf("4. Reverse List 1\n");

printf("5. Concatenate List 2 to List 1\n");

printf("6. Print List 1\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

if (choice == 1) {

printf("Enter value to insert in List 1: ");

scanf("%d", &value);

addNode(&list1, value);

}

else if (choice == 2) {

printf("Enter value to insert in List 2: ");

scanf("%d", &value);

addNode(&list2, value);

}

else if (choice == 3) {

sortList(list1);

printf("List 1 sorted.\n");

}

else if (choice == 4) {

reverseList(&list1);

printf("List 1 reversed.\n");

}

else if (choice == 5) {

concatenateLists(&list1, list2);

printf("List 2 concatenated to List 1.\n");

}

else if (choice == 6) {

printf("List 1: ");

display(list1);

}

else if (choice == 7) {

printf("Exiting...\n");

return 0;

}

else {

printf("Invalid choice. Try again.\n");

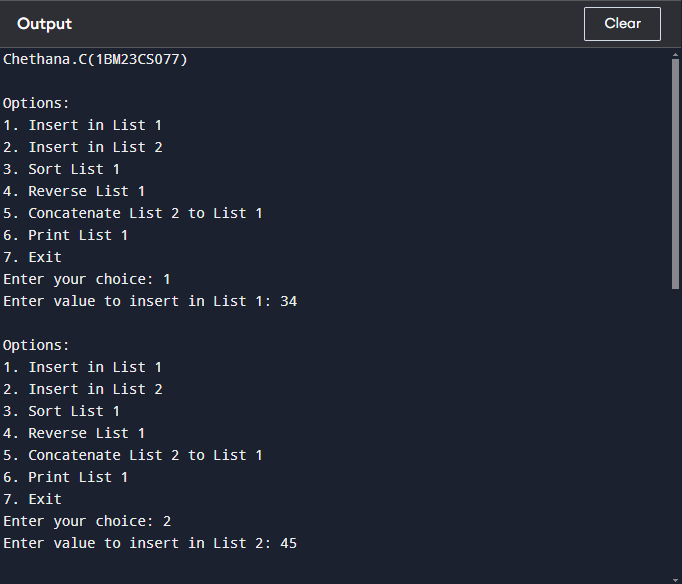
}

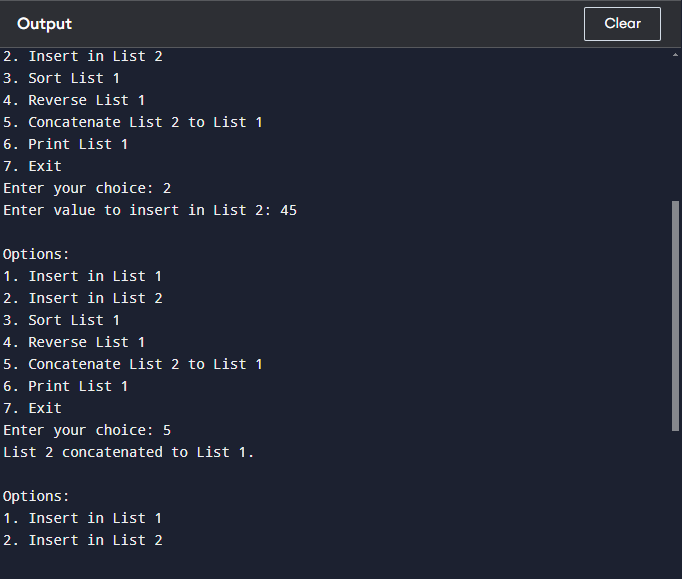
}

return 0;

}

**Output:**





**Lab program 6b)**

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

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void push(struct Node\*\* top, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = \*top;

\*top = newNode;

}

int pop(struct Node\*\* top) {

if (\*top == NULL) {

printf("Stack Underflow\n");

return -1;

}

int value = (\*top)->data;

struct Node\* temp = \*top;

\*top = (\*top)->next;

free(temp);

return value;

}

void displayStack(struct Node\* top) {

if (top == NULL) {

printf("Stack is empty.\n");

return;

}

struct Node\* temp = top;

printf("Stack: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

void enqueue(struct Node\*\* front, struct Node\*\* rear, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

if (\*rear == NULL) {

\*front = \*rear = newNode;

return;

}

(\*rear)->next = newNode;

\*rear = newNode;

}

int dequeue(struct Node\*\* front) {

if (\*front == NULL) {

printf("Queue Underflow\n");

return -1;

}

int value = (\*front)->data;

struct Node\* temp = \*front;

\*front = (\*front)->next;

free(temp);

return value;

}

void displayQueue(struct Node\* front) {

if (front == NULL) {

printf("Queue is empty.\n");

return;

}

struct Node\* temp = front;

printf("Queue: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

int main() {

struct Node\* stackTop = NULL;

struct Node\* queueFront = NULL;

struct Node\* queueRear = NULL;

int choice, value;

while (1) {

printf("\nMenu:\n");

printf("1. Push to Stack\n");

printf("2. Pop from Stack\n");

printf("3. Enqueue to Queue\n");

printf("4. Dequeue from Queue\n");

printf("5. Display Stack\n");

printf("6. Display Queue\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

if (choice == 1) {

printf("Enter value to push to stack: ");

scanf("%d", &value);

push(&stackTop, value);

}

else if (choice == 2) {

value = pop(&stackTop);

if (value != -1) {

printf("Popped from stack: %d\n", value);

}

}

else if (choice == 3) {

printf("Enter value to enqueue to queue: ");

scanf("%d", &value);

enqueue(&queueFront, &queueRear, value);

}

else if (choice == 4) {

value = dequeue(&queueFront);

if (value != -1) {

printf("Dequeued from queue: %d\n", value);

}

}

else if (choice == 5) {

displayStack(stackTop);

}

else if (choice == 6) {

displayQueue(queueFront);

}

else if (choice == 7) {

printf("Exiting...\n");

return 0;

}

else {

printf("Invalid choice. Try again.\n");

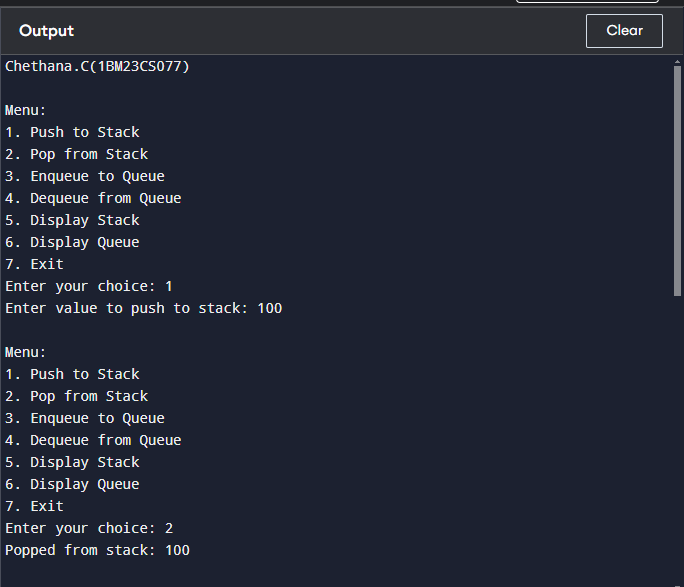
}

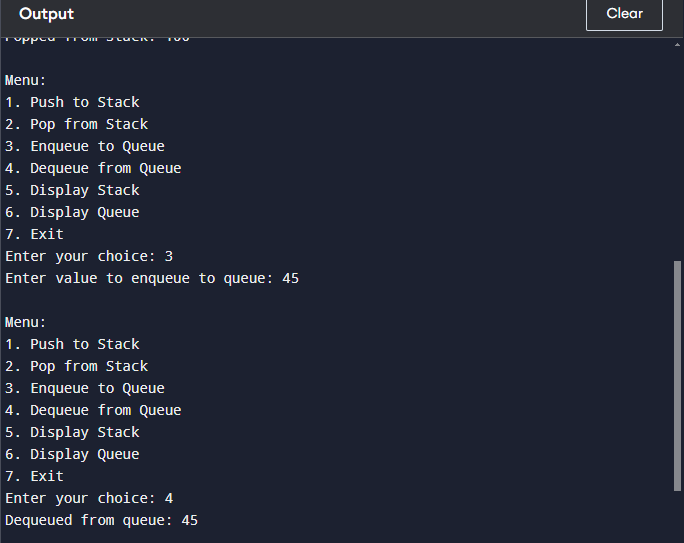
}

return 0;

}

**Output:**





**Lab program 7:  
WAP to Implement doubly link list with primitive operations**

**a) Create a doubly linked list.**

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

**c) Delete the node based on a specific value**

**d) Display the contents of the list**

#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 insertLeft(struct Node\*\* head, int value, int newData) {

struct Node\* newNode = createNode(newData);

struct Node\* temp = \*head;

if (\*head == NULL) {

\*head = newNode;

return;

}

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

temp = temp->next;

}

if (temp != NULL) {

newNode->next = temp;

newNode->prev = temp->prev;

if (temp->prev != NULL) {

temp->prev->next = newNode;

} else {

\*head = newNode;

}

temp->prev = newNode;

}

}

void deleteNode(struct Node\*\* head, int value) {

struct Node\* temp = \*head;

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

temp = temp->next;

}

if (temp != NULL) {

if (temp->prev != NULL) {

temp->prev->next = temp->next;

} else {

\*head = temp->next;

}

if (temp->next != NULL) {

temp->next->prev = temp->prev;

}

free(temp);

}

}

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");

}

int main() {

struct Node\* head = NULL;

int choice, value, newData;

printf("Chethana.C(1BM23CS077)");

while (1) {

printf("\nMenu:\n");

printf("1. Create \n");

printf("2. Insertto left\n");

printf("3. Delete BY value\n");

printf("4. Display \n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

if (choice == 1) {

printf("Enter value to create a new node: ");

scanf("%d", &value);

struct Node\* newNode = createNode(value);

if (head == NULL) {

head = newNode;

} else {

struct Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

}

else if (choice == 2) {

printf("Enter the value of the node to insert before: ");

scanf("%d", &value);

printf("Enter the new node's data: ");

scanf("%d", &newData);

insertLeft(&head, value, newData);

}

else if (choice == 3) {

printf("Enter the value to delete: ");

scanf("%d", &value);

deleteNode(&head, value);

}

else if (choice == 4) {

displayList(head);

}

else if (choice == 5) {

printf("Exiting...\n");

return 0;

}

else {

printf("Invalid choice. Try again.\n");

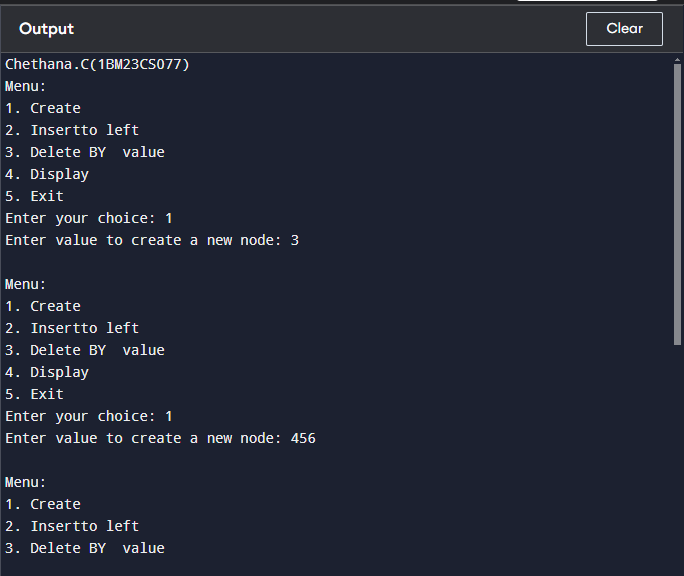
}

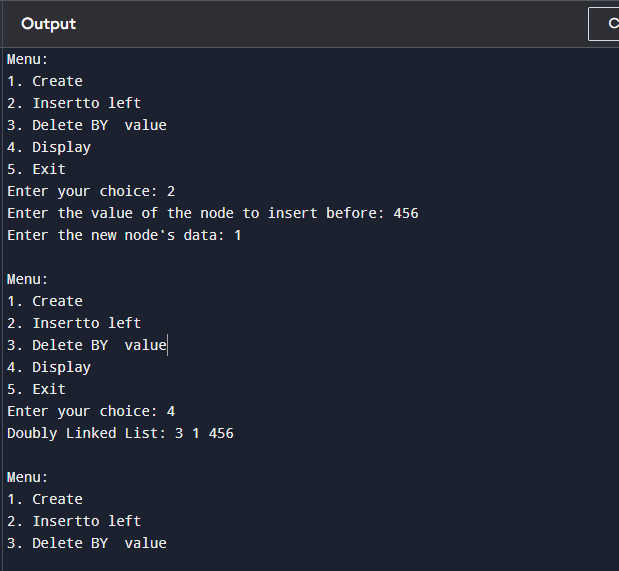
}

return 0;

}

**Output:**

****



**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**

**c) To display the elements in the 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 data) {

if (root == NULL) {

return createNode(data);

}

if (data < root->data) {

root->left = insert(root->left, data);

} else {

root->right = insert(root->right, data);

}

return root;

}

void inorder(struct Node\* root) {

if (root != NULL) {

inorder(root->left);

printf("%d ", root->data);

inorder(root->right);

}

}

void preorder(struct Node\* root) {

if (root != NULL) {

printf("%d ", root->data);

preorder(root->left);

preorder(root->right);

}

}

void postorder(struct Node\* root) {

if (root != NULL) {

postorder(root->left);

postorder(root->right);

printf("%d ", root->data);

}

}

void display(struct Node\* root) {

printf("In-order traversal: ");

inorder(root);

printf("\n");

printf("Pre-order traversal: ");

preorder(root);

printf("\n");

printf("Post-order traversal: ");

postorder(root);

printf("\n");

}

int main() {

struct Node\* root = NULL;

printf("Chethana,c (1bm23cs077)\n");

root = insert(root, 50);

insert(root, 30);

insert(root, 70);

insert(root, 20);

insert(root, 40);

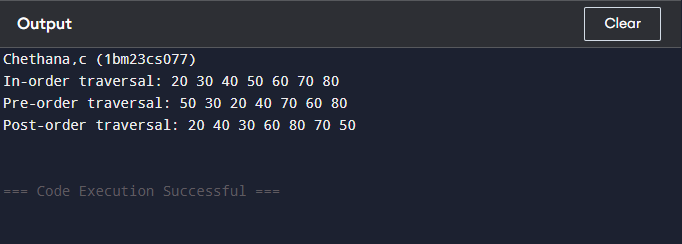
insert(root, 60);

insert(root, 80);

display(root);

return 0;

**Output:**

****

**Lab program 9a:**

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

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

struct Queue {

int items[MAX];

int front;

int rear;

};

void initQueue(struct Queue\* q) {

q->front = -1;

q->rear = -1;

}

int isEmpty(struct Queue\* q) {

return q->front == -1;

}

int isFull(struct Queue\* q) {

return q->rear == MAX - 1;

}

void enqueue(struct Queue\* q, int value) {

if (isFull(q)) {

printf("Queue is full\n");

return;

}

if (q->front == -1) {

q->front = 0;

}

q->rear++;

q->items[q->rear] = value;

}

int dequeue(struct Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return -1;

}

int item = q->items[q->front];

q->front++;

if (q->front > q->rear) {

q->front = q->rear = -1;

}

return item;

}

void bfs(int graph[MAX][MAX], int visited[MAX], int startVertex, int vertices) {

struct Queue q;

initQueue(&q);

visited[startVertex] = 1;

enqueue(&q, startVertex);

while (!isEmpty(&q)) {

int currentVertex = dequeue(&q);

printf("%d ", currentVertex);

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

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

visited[i] = 1;

enqueue(&q, i);

}

}

}

}

int main() {

int vertices, startVertex;

int graph[MAX][MAX] = {0};

int visited[MAX] = {0};

printf("chethana,c(1bm23cs077)\n");

printf("Enter the number of vertices: ");

scanf("%d", &vertices);

printf("Enter the adjacency matrix:\n");

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

for (int j = 0; j < vertices; j++) {

scanf("%d", &graph[i][j]);

}

}

printf("Enter the starting vertex for BFS: ");

scanf("%d", &startVertex);

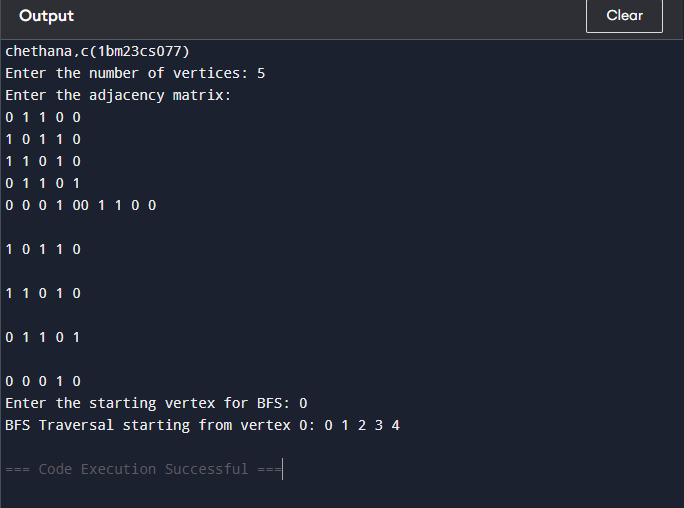
printf("BFS Traversal starting from vertex %d: ", startVertex);

bfs(graph, visited, startVertex, vertices);

return 0;

}

**Output:**



**Lab program 9b:**

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

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

void dfs(int graph[MAX][MAX], int visited[MAX], int vertex, int vertices) {

visited[vertex] = 1;

printf("%d ", vertex);

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

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

dfs(graph, visited, i, vertices);

}

}

}

int isConnected(int graph[MAX][MAX], int vertices) {

int visited[MAX] = {0};

// Start DFS from vertex 0

dfs(graph, visited, 0, vertices);

// Check if all vertices were visited

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

if (!visited[i]) {

return 0; // Graph is not connected

}

}

return 1; // Graph is connected

}

int main() {

int vertices;

int graph[MAX][MAX] = {0};

printf("chethana.c(1bm23cs077)\n");

printf("Enter the number of vertices: ");

scanf("%d", &vertices);

printf("Enter the adjacency matrix (0 for no edge, 1 for edge):\n");

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

for (int j = 0; j < vertices; j++) {

scanf("%d", &graph[i][j]);

}

}

if (isConnected(graph, vertices)) {

printf("The graph is connected.\n");

} else {

printf("The graph is not connected.\n");

}

return 0;

}

**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 -&gt; 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>

#define MAX\_KEYS 5

#define MAX\_HT\_SIZE 7

struct Employee {

int key;

int address;

};

struct HashTable {

struct Employee\* table[MAX\_HT\_SIZE];

};

int hashFunction(int key, int m) {

return key % m;

}

int linearProbing(struct HashTable\* ht, int key, int m) {

int index = hashFunction(key, m);

int originalIndex = index;

while (ht->table[index] != NULL && ht->table[index]->key != key) {

index = (index + 1) % m;

if (index == originalIndex) {

return -1;

}

}

return index;

}

void insert(struct HashTable\* ht, int key, int address, int m) {

int index = linearProbing(ht, key, m);

if (index == -1) {

printf("Hash table is full, unable to insert key: %d\n", key);

return;

}

struct Employee\* emp = (struct Employee\*)malloc(sizeof(struct Employee));

emp->key = key;

emp->address = address;

ht->table[index] = emp;

}

void display(struct HashTable\* ht, int m) {

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

if (ht->table[i] != NULL) {

printf("Index: %d, Key: %d\n", i, ht->table[i]->key);

}

}

}

int main() {

struct HashTable ht;

for (int i = 0; i < MAX\_HT\_SIZE; i++) {

ht.table[i] = NULL;

}

int m = MAX\_HT\_SIZE;

int keys[MAX\_KEYS] = {1001, 2045, 1003, 1023, 3050};

int addresses[MAX\_KEYS] = {10, 20, 30, 40, 60};

for (int i = 0; i < MAX\_KEYS; i++) {

insert(&ht, keys[i], addresses[i], m);

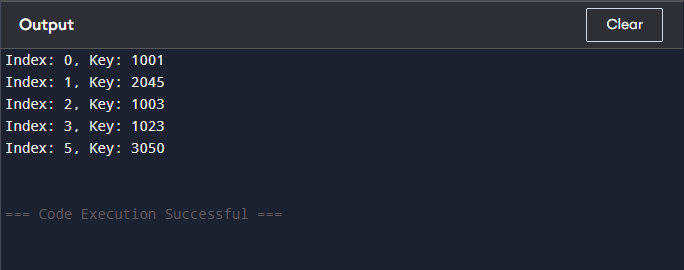
}

display(&ht, m);

return 0;

}

**Output:**

****