**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**VISMAY PAWAR N(1BM22CS331)**

**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**

**Dec 2023- March 2024**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

**(Affiliated To Visvesvaraya Technological University, Belgaum)**

**Department of Computer Science and Engineering**

****

This is to certify that the Lab work entitled **“DATA STRUCTURES”** carried out by VISMAY PAWAR N **(1BM22CS331)**, 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 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST )**work prescribed for the said degree.

**Prof. Lakshmi Neelima** **Dr. Jyothi S Nayak**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 1 | Working of stack using an array | 4-8 |
| 2 | WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression | 9-11 |
| 3 | WAP to simulate the working of a queue of integers using an array.  WAP to simulate the working of a circular queue of integers using an array | 12-23 |
| 4 | WAP to Implement Singly Linked List (Insertion) | 24-28 |
| 5 | WAP to Implement Singly Linked List (Deletion) | 29-34 |
| 6 | WAP to Implement Single Link List (Sorting,Reversing and Concatenation).  WAP to implement Stack & Queues using Linked Representation . | 35-47 |
| 7 | WAP to Implement doubly link list with primitive operations | 48-54 |
| 8 | Constructing Binary Search Tree(BST) | 55-62 |
| 9 | BFS and DFS | 63-70 |
| 10 | Hash Function | 71-73 |

**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)]);

(\*top)--;

}

}

void display(int st[], int \*top) {

int i;

if (\*top == -1) {

printf("Stack is empty\n");

return;

}

for (i = 0; i <= \*top; i++)

printf("%d\t", st[i]);

}

int main() {

int st[STACK\_SIZE], top = -1, c;

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);

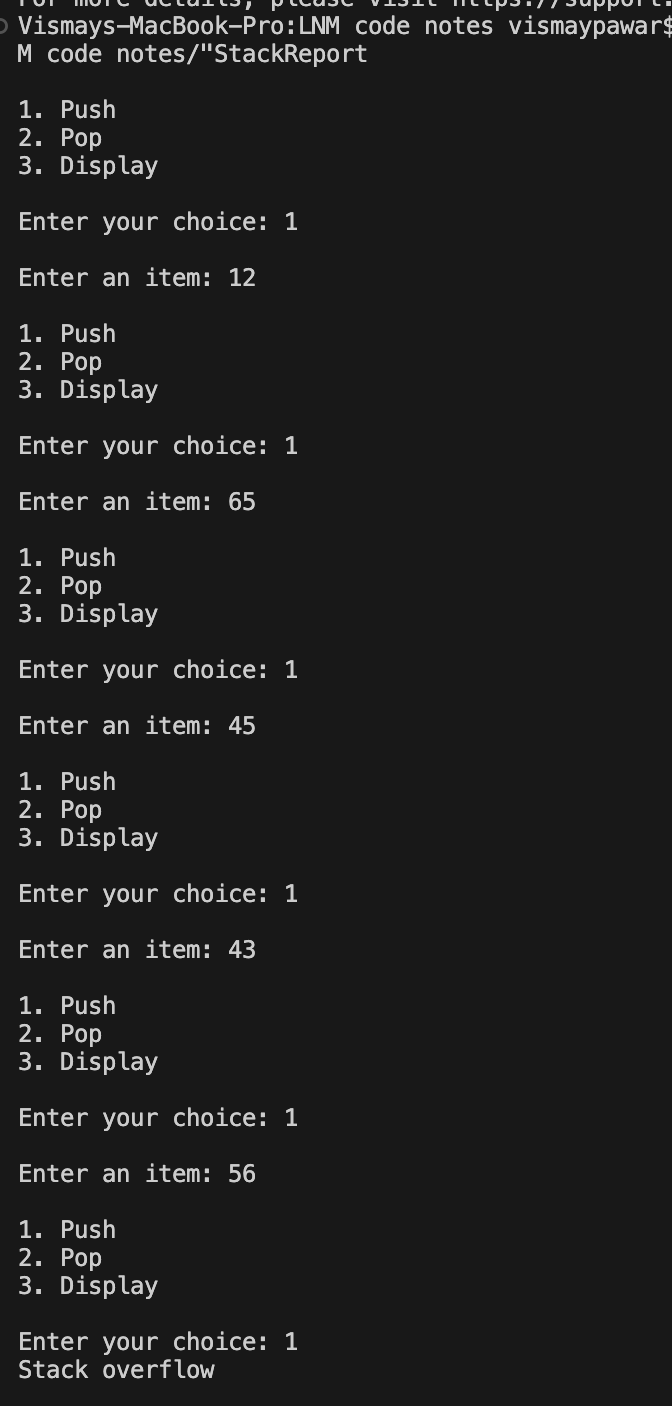
}

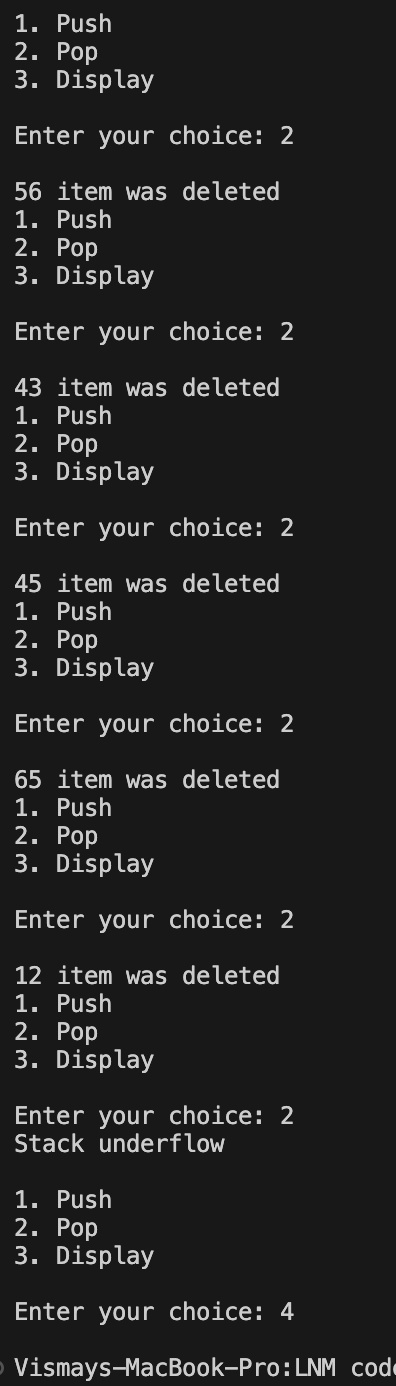
}

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) .**

#include <stdio.h>

#include <ctype.h>

#define SIZE 50

char stack[SIZE];

int top = -1;

void push(char elem) {

stack[++top] = elem;

}

char pop() {

return stack[top--];

}

int pr(char symbol) {

if (symbol == '^')

return 3;

else if (symbol == '\*' || symbol == '/')

return 2;

else if (symbol == '+' || symbol == '-')

return 1;

else

return 0;

}

int main() {

char infix[50], postfix[50], ch, elem;

int i = 0, k = 0;

printf("Enter Infix Expression: ");

scanf("%s", infix);

push('#');

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

if (ch == '(')

push(ch);

else if (isalnum(ch))

postfix[k++] = ch;

else if (ch == ')') {

while (stack[top] != '(')

postfix[k++] = pop();

elem = pop();

} else {

while (pr(stack[top]) >= pr(ch))

postfix[k++] = pop();

push(ch);

}

}

while (stack[top] != '#')

postfix[k++] = pop();

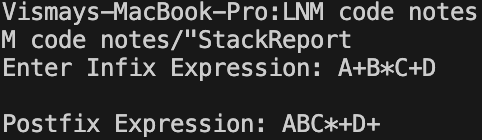
postfix[k] = '\0';

printf("\nPostfix Expression: %s\n", postfix);

return 0;

}

**Output:**

****

**Lab program 3a:**

**WAP to simulate the working of a queue of integers using an array. Provide the following operations**

**a) Insert**

**b) Delete**

**c) Display**

**The program should print appropriate messages for queue empty and queue overflow conditions.**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 5

typedef struct {

int queue[MAX\_SIZE];

int front, rear;

int size;

} Queue;

void initQueue(Queue \*q) {

q->front = 0;

q->rear = -1;

q->size = 0;

}

bool isEmpty(Queue \*q) {

return q->size == 0;

}

bool isFull(Queue \*q) {

return q->size == MAX\_SIZE;

}

void enqueue(Queue \*q, int item) {

if (isFull(q)) {

printf("Queue Overflow! Cannot insert element.\n");

return;

}

q->rear = (q->rear + 1) % MAX\_SIZE;

q->queue[q->rear] = item;

q->size++;

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

}

int dequeue(Queue \*q) {

if (isEmpty(q)) {

printf("Queue Underflow! Cannot delete element.\n");

return -1;

}

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

q->front++;

q->size--;

printf("Deleted %d from the queue.\n", item);

return item;

}

void display(Queue \*q) {

if (isEmpty(q)) {

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

return;

}

printf("Queue elements: ");

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

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

}

printf("\n");

}

int main() {

Queue q;

initQueue(&q);

int choice, item;

do {

printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

if (isFull(&q)) {

printf("Queue Overflow. Cannot enqueue.\n");

} else {

printf("Enter element to enqueue: ");

scanf("%d", &item);

enqueue(&q, item);

}

break;

case 2:

dequeue(&q);

break;

case 3:

display(&q);

break;

case 4:

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

break;

default:

printf("Invalid choice! Please enter a valid option.\n");

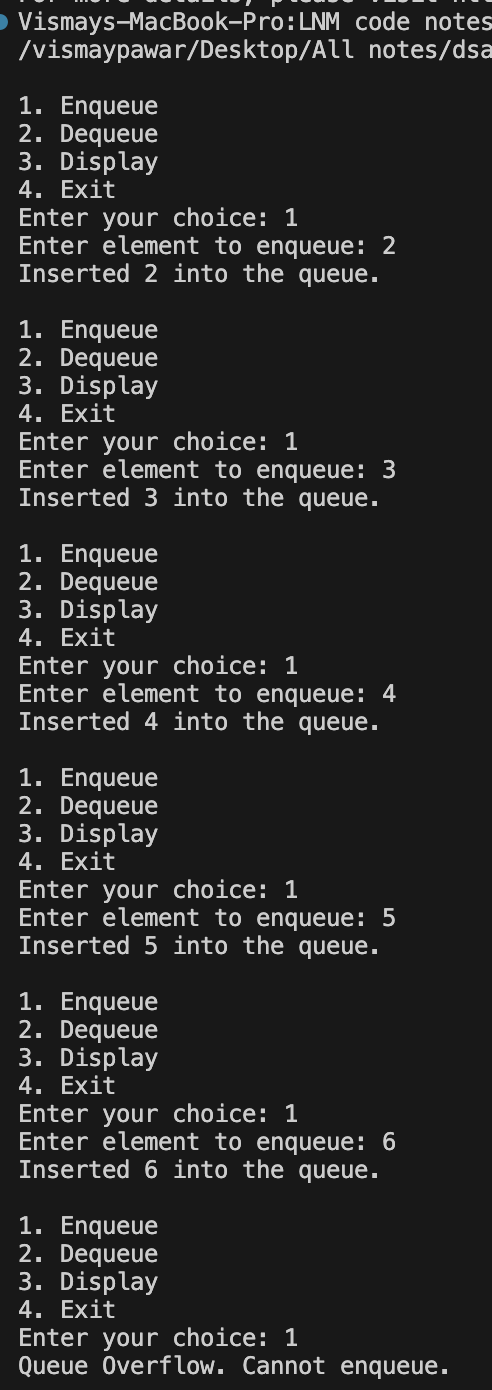
}

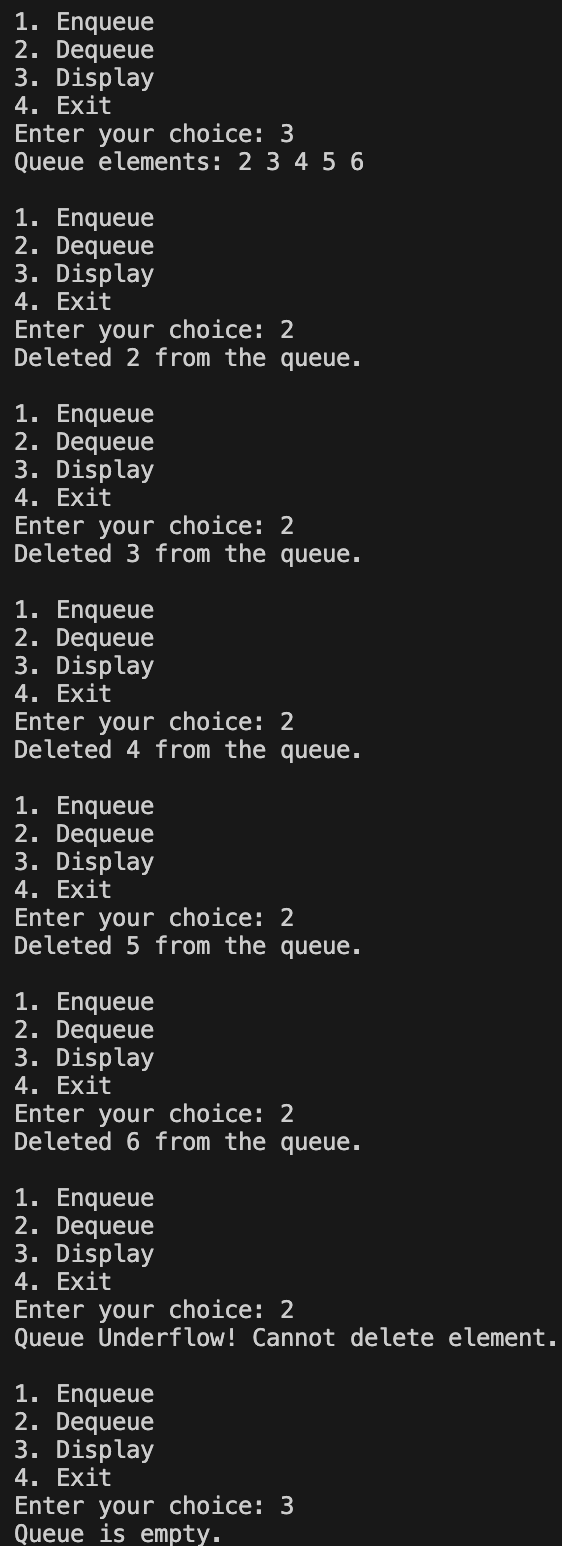
} while (choice != 4);

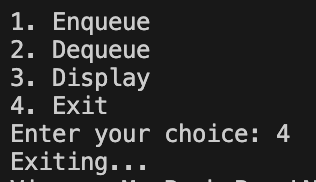
return 0;

}

**Output:**

****

****

****

**Lab program 3b:**

**WAP to simulate the working of a circular queue of integers using an array. Provide the following operations.**

**a) Insert**

**b) Delete**

**c) Display**

**The program should print appropriate messages for queue empty and queue overflow conditions.**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 5

typedef struct {

int queue[MAX\_SIZE];

int front, rear;

int size;

} CircularQueue;

void initQueue(CircularQueue \*cq) {

cq->front = 0;

cq->rear = -1;

cq->size = 0;

}

bool isEmpty(CircularQueue \*cq) {

return cq->size == 0;

}

bool isFull(CircularQueue \*cq) {

return cq->size == MAX\_SIZE;

}

void enqueue(CircularQueue \*cq, int item) {

if (isFull(cq)) {

printf("Queue Overflow! Cannot insert element.\n");

return;

}

cq->rear = (cq->rear + 1) % MAX\_SIZE;

cq->queue[cq->rear] = item;

cq->size++;

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

}

int dequeue(CircularQueue \*cq) {

if (isEmpty(cq)) {

printf("Queue Underflow! Cannot delete element.\n");

return -1;

}

int item = cq->queue[cq->front];

cq->front = (cq->front + 1) % MAX\_SIZE;

cq->size--;

printf("Deleted %d from the queue.\n", item);

return item;

}

void display(CircularQueue \*cq) {

if (isEmpty(cq)) {

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

return;

}

printf("Queue elements: ");

int i, count;

for (count = 0, i = cq->front; count < cq->size; count++, i = (i + 1) % MAX\_SIZE) {

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

}

printf("\n");

}

int main() {

CircularQueue cq;

initQueue(&cq);

int choice, item;

do {

printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter element to enqueue: ");

scanf("%d", &item);

enqueue(&cq, item);

break;

case 2:

dequeue(&cq);

break;

case 3:

display(&cq);

break;

case 4:

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

break;

default:

printf("Invalid choice! Please enter a valid option.\n");

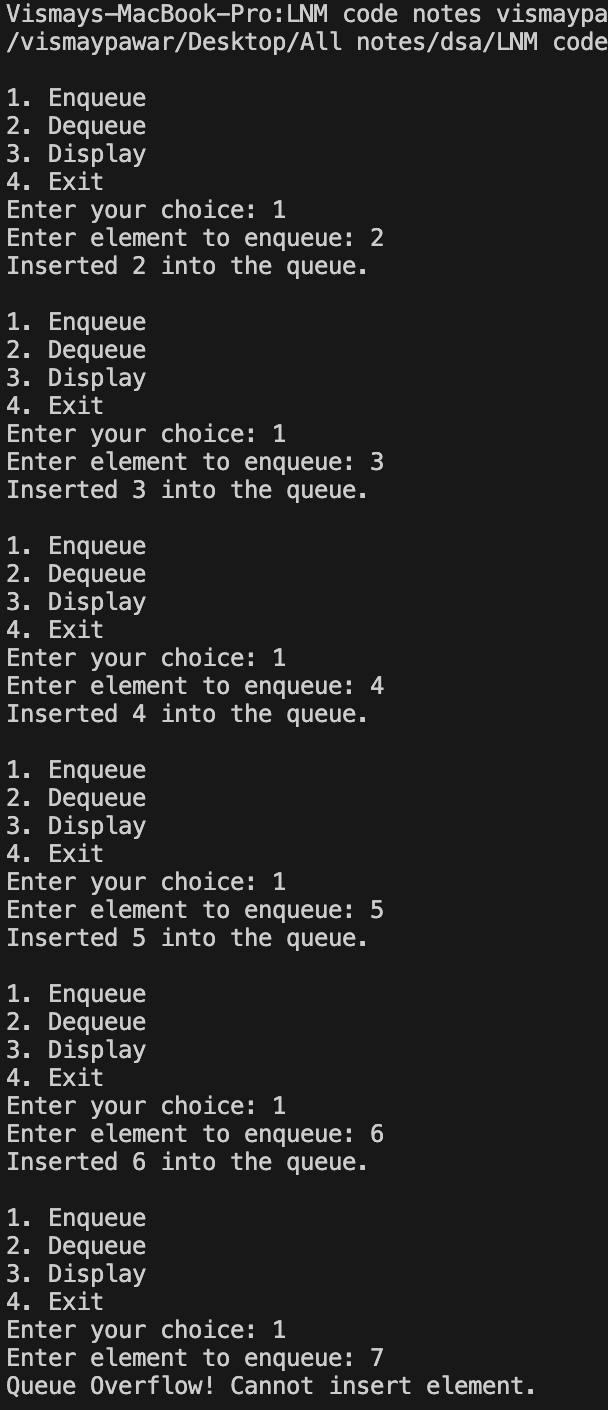
}

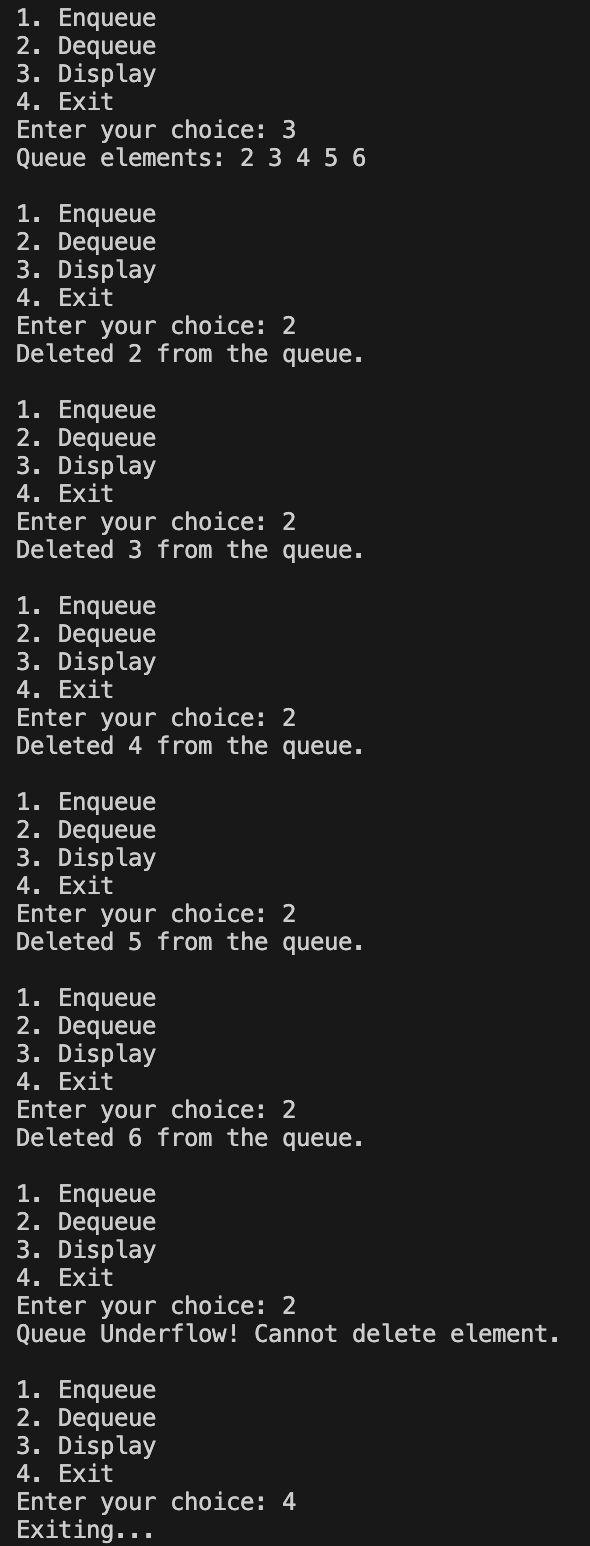
} while (choice != 4);

return 0;

}

**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.**

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

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node \*next;

} Node;

Node\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

Node\* insertAtBeginning(Node \*head, int data) {

Node \*newNode = createNode(data);

newNode->next = head;

return newNode;

}

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

if (position < 1) {

printf("Invalid position!\n");

return head;

}

Node \*newNode = createNode(data);

if (position == 1 || head == NULL) {

newNode->next = head;

return newNode;

}

Node \*current = head;

int count = 1;

while (count < position - 1 && current != NULL) {

current = current->next;

count++;

}

if (current == NULL) {

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

return head;

}

newNode->next = current->next;

current->next = newNode;

return head;

}

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

Node \*newNode = createNode(data);

if (head == NULL) {

return newNode;

}

Node \*current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

return head;

}

void displayList(Node \*head) {

if (head == NULL) {

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

return;

}

Node \*current = head;

printf("List elements: ");

while (current != NULL) {

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

current = current->next;

}

printf("\n");

}

void freeList(Node \*head) {

Node \*current = head;

Node \*temp;

while (current != NULL) {

temp = current;

current = current->next;

free(temp);

}

}

int main() {

Node \*head = NULL;

int choice, data, position;

do {

printf("\n1. Insert at beginning\n2. Insert at position\n3. Insert at end\n4. Display\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at beginning: ");

scanf("%d", &data);

head = insertAtBeginning(head, data);

break;

case 2:

printf("Enter data to insert: ");

scanf("%d", &data);

printf("Enter position to insert at: ");

scanf("%d", &position);

head = insertAtPosition(head, data, position);

break;

case 3:

printf("Enter data to insert at end: ");

scanf("%d", &data);

head = insertAtEnd(head, data);

break;

case 4:

displayList(head);

break;

case 5:

freeList(head);

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

break;

default:

printf("Invalid choice! Please enter a valid option.\n");

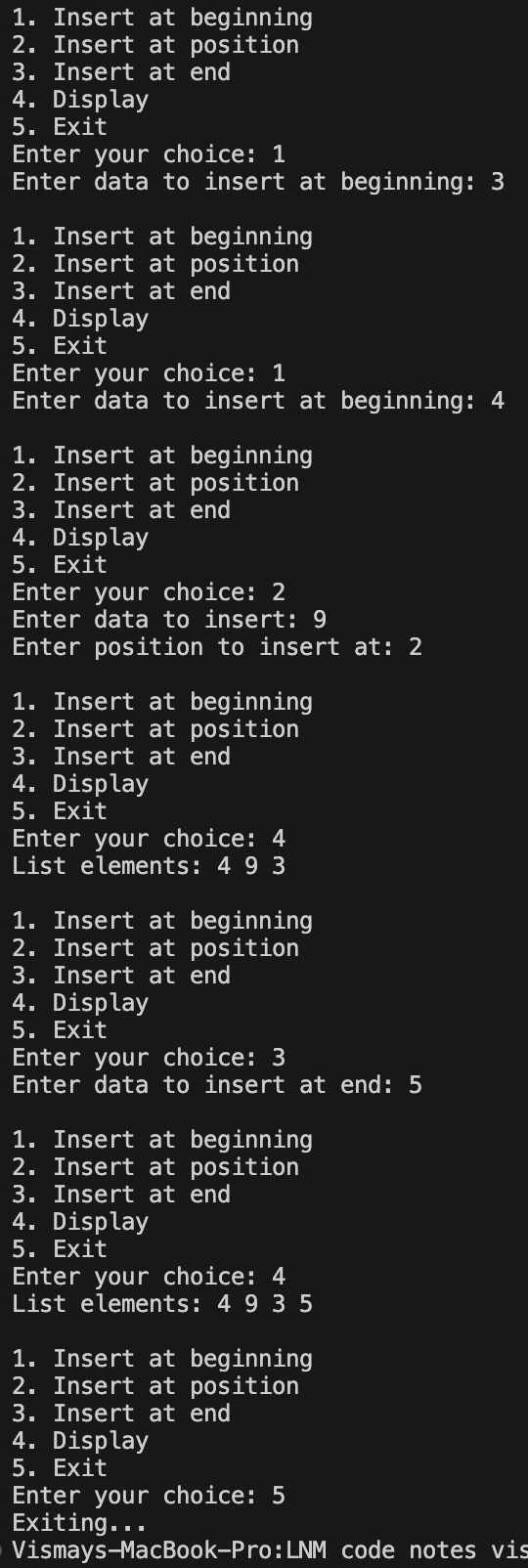
}

} while (choice != 5);

return 0;

}

**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>

typedef struct Node {

int data;

struct Node \*next;

} Node;

Node\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

Node\* insertAtBeginning(Node \*head, int data) {

Node \*newNode = createNode(data);

newNode->next = head;

return newNode;

}

Node\* deleteFirstNode(Node \*head) {

if (head == NULL) {

printf("List is empty. Nothing to delete.\n");

return NULL;

}

Node \*temp = head;

head = head->next;

free(temp);

printf("Deleted the first node from the list.\n");

return head;

}

Node\* deleteSpecifiedNode(Node \*head, int key) {

Node \*current = head;

Node \*prev = NULL;

if (current != NULL && current->data == key) {

head = head->next;

free(current);

printf("Deleted node with key %d from the list.\n", key);

return head;

}

while (current != NULL && current->data != key) {

prev = current;

current = current->next;

}

if (current == NULL) {

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

return head;

}

prev->next = current->next;

free(current);

printf("Deleted node with key %d from the list.\n", key);

return head;

}

Node\* deleteLastNode(Node \*head) {

if (head == NULL) {

printf("List is empty. Nothing to delete.\n");

return NULL;

}

if (head->next == NULL) {

free(head);

printf("Deleted the last node from the list.\n");

return NULL;

}

Node \*prev = NULL;

Node \*current = head;

while (current->next != NULL) {

prev = current;

current = current->next;

}

prev->next = NULL;

free(current);

printf("Deleted the last node from the list.\n");

return head;

}

void displayList(Node \*head) {

if (head == NULL) {

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

return;

}

Node \*current = head;

printf("List elements: ");

while (current != NULL) {

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

current = current->next;

}

printf("\n");

}

void freeList(Node \*head) {

Node \*current = head;

Node \*temp;

while (current != NULL) {

temp = current;

current = current->next;

free(temp);

}

}

int main() {

Node \*head = NULL;

int choice, data, key;

do {

printf("\n1. Insert at beginning\n2. Delete first node\n3. Delete specified node\n4. Delete last node\n5. Display\n6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at beginning: ");

scanf("%d", &data);

head = insertAtBeginning(head, data);

break;

case 2:

head = deleteFirstNode(head);

break;

case 3:

printf("Enter the key of node to delete: ");

scanf("%d", &key);

head = deleteSpecifiedNode(head, key);

break;

case 4:

head = deleteLastNode(head);

break;

case 5:

displayList(head);

break;

case 6:

freeList(head);

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

break;

default:

printf("Invalid choice! Please enter a valid option.\n");

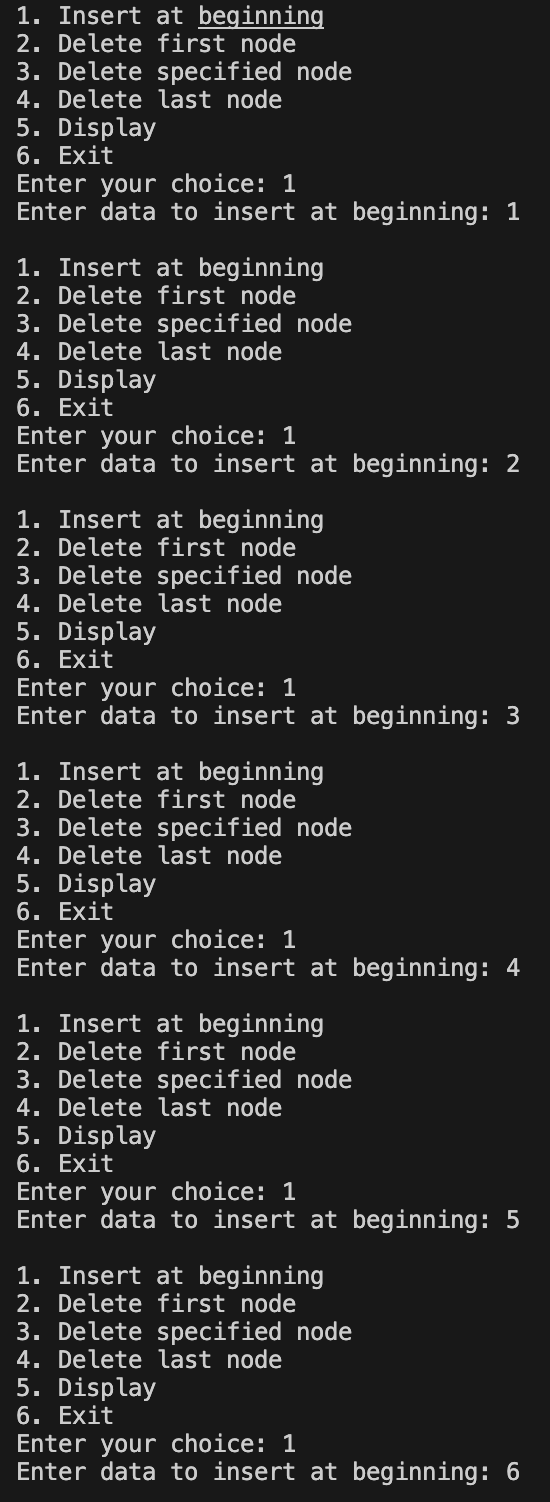
}

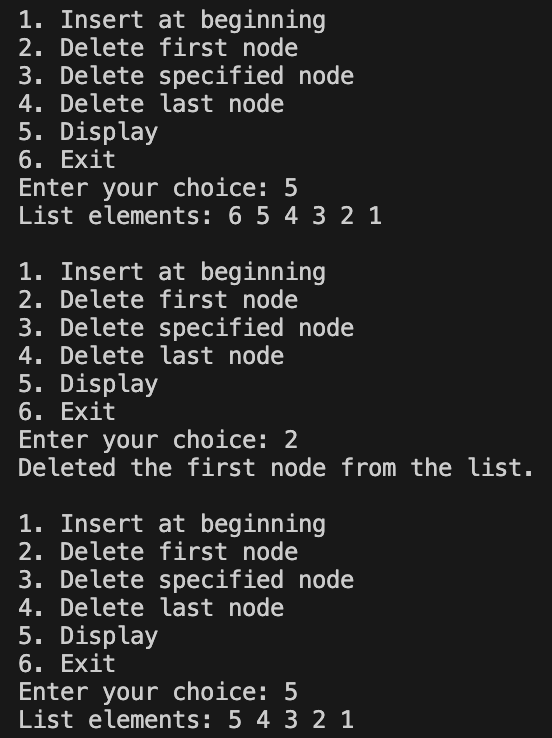
} while (choice != 6);

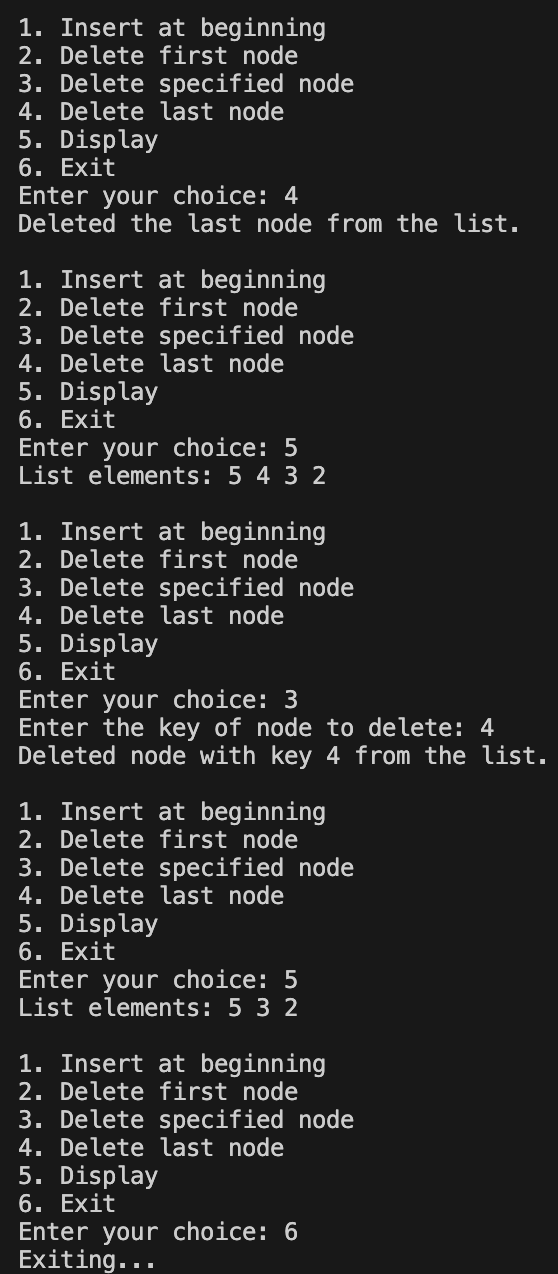
return 0;

}

**Output:**

****

****

****

**Lab program 6a:**

**WAP to Implement Single Link List with following operations**

**a) Sort the linked list.**

**b) Reverse the linked list.**

**c) Concatenation of two linked lists**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node \*next;

} Node;

Node\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

Node\* insertAtBeginning(Node \*head, int data) {

Node \*newNode = createNode(data);

newNode->next = head;

return newNode;

}

void displayList(Node \*head) {

if (head == NULL) {

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

return;

}

Node \*current = head;

printf("List elements: ");

while (current != NULL) {

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

current = current->next;

}

printf("\n");

}

Node\* sortLinkedList(Node \*head) {

if (head == NULL || head->next == NULL)

return head;

Node \*prev = head;

Node \*current = head->next;

while (current != NULL) {

Node \*innerPrev = NULL;

Node \*innerCurrent = head;

while (innerCurrent != current) {

if (innerCurrent->data > current->data) {

prev->next = current->next;

current->next = innerCurrent;

if (innerPrev == NULL)

head = current;

else

innerPrev->next = current;

current = prev->next;

break;

}

innerPrev = innerCurrent;

innerCurrent = innerCurrent->next;

}

if (innerCurrent == current) {

prev = current;

current = current->next;

}

}

return head;

}

Node\* reverseLinkedList(Node \*head) {

Node \*prev = NULL;

Node \*current = head;

Node \*next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

return head;

}

Node\* concatenateLinkedLists(Node \*list1, Node \*list2) {

if (list1 == NULL)

return list2;

if (list2 == NULL)

return list1;

Node \*current = list1;

while (current->next != NULL) {

current = current->next;

}

current->next = list2;

return list1;

}

int main() {

Node \*list1 = NULL;

Node \*list2 = NULL;

list1 = insertAtBeginning(list1, 30);

list1 = insertAtBeginning(list1, 20);

list1 = insertAtBeginning(list1, 10);

printf("List 1:\n");

displayList(list1);

list1 = sortLinkedList(list1);

printf("Sorted List 1:\n");

displayList(list1);

list1 = reverseLinkedList(list1);

printf("Reversed List 1:\n");

displayList(list1);

list2 = insertAtBeginning(list2, 60);

list2 = insertAtBeginning(list2, 50);

list2 = insertAtBeginning(list2, 40);

printf("List 2:\n");

displayList(list2);

list2 = sortLinkedList(list2);

printf("Sorted List 2:\n");

displayList(list2);

list2 = reverseLinkedList(list2);

printf("Reversed List 2:\n");

displayList(list2);

Node \*concatenatedList = concatenateLinkedLists(list1, list2);

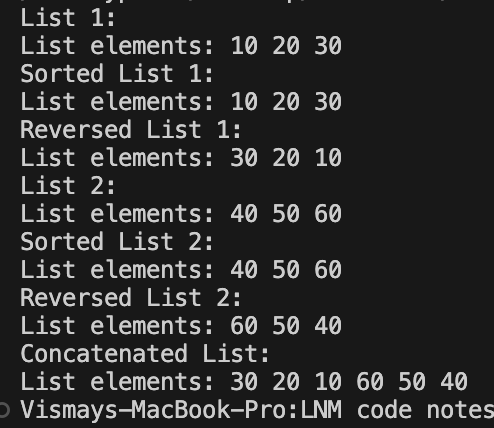
printf("Concatenated List:\n");

displayList(concatenatedList);

return 0;

}

**Output:**

****

**Lab program 6b:**

**WAP to implement Stack & Queues using Linked Representation**

#include <stdio.h>

#include <stdlib.h>

typedef struct StackNode {

int data;

struct StackNode\* next;

} StackNode;

StackNode\* createStackNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

int isEmpty(StackNode\* root) {

return (root == NULL);

}

void push(StackNode\*\* root, int data) {

StackNode\* newNode = createStackNode(data);

newNode->next = \*root;

\*root = newNode;

printf("Pushed %d onto the stack.\n", data);

}

int pop(StackNode\*\* root) {

if (isEmpty(\*root)) {

printf("Stack Underflow! Cannot pop element.\n");

return -1;

}

int popped = (\*root)->data;

StackNode\* temp = \*root;

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

free(temp);

return popped;

}

int peek(StackNode\* root) {

if (isEmpty(root)) {

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

return -1;

}

return root->data;

}

void displayStack(StackNode\* root) {

if (isEmpty(root)) {

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

return;

}

printf("Stack elements: ");

while (root != NULL) {

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

root = root->next;

}

printf("\n");

}

int main() {

StackNode\* stack = NULL;

int choice, data;

do {

printf("\n1. Push\n2. Pop\n3. Peek\n4. Display\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to push onto the stack: ");

scanf("%d", &data);

push(&stack, data);

break;

case 2:

printf("Popped %d from the stack.\n", pop(&stack));

break;

case 3:

printf("Top element of the stack: %d\n", peek(stack));

break;

case 4:

displayStack(stack);

break;

case 5:

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

break;

default:

printf("Invalid choice! Please enter a valid option.\n");

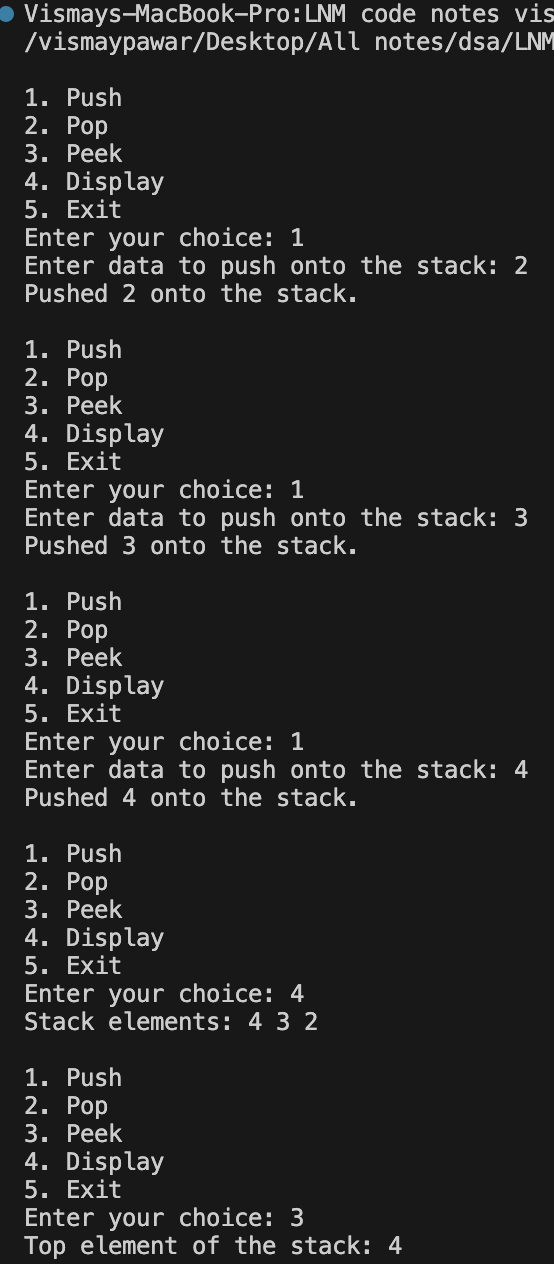
}

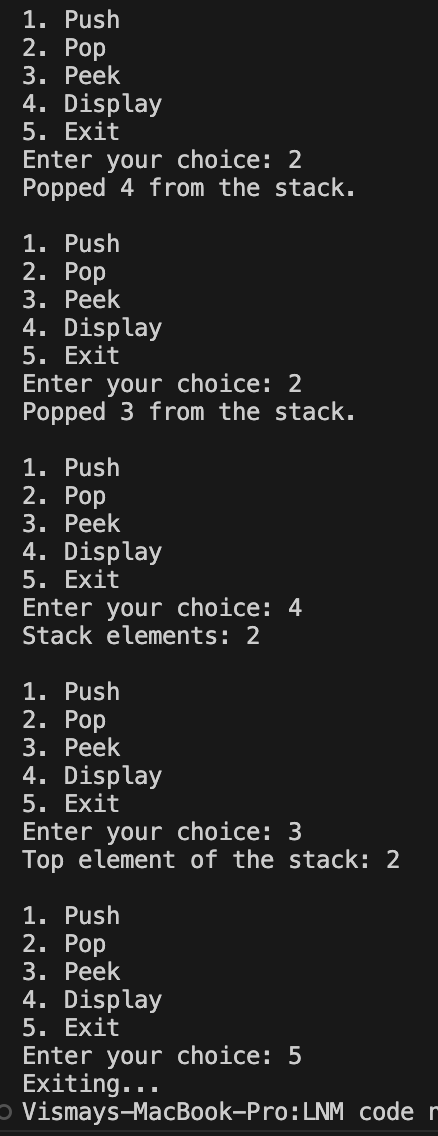
} while (choice != 5);

return 0;

}

**Output:**

****

****

**Queue:**

#include <stdio.h>

#include <stdlib.h>

typedef struct QueueNode {

int data;

struct QueueNode\* next;

} QueueNode;

typedef struct {

QueueNode\* front;

QueueNode\* rear;

} Queue;

QueueNode\* createQueueNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

Queue\* createQueue() {

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

if (queue == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

queue->front = queue->rear = NULL;

return queue;

}

int isEmpty(Queue\* queue) {

return (queue->front == NULL);

}

void enqueue(Queue\* queue, int data) {

QueueNode\* newNode = createQueueNode(data);

if (isEmpty(queue)) {

queue->front = queue->rear = newNode;

} else {

queue->rear->next = newNode;

queue->rear = newNode;

}

printf("Enqueued %d into the queue.\n", data);

}

int dequeue(Queue\* queue) {

if (isEmpty(queue)) {

printf("Queue Underflow! Cannot dequeue element.\n");

return -1;

}

int dequeued = queue->front->data;

QueueNode\* temp = queue->front;

queue->front = queue->front->next;

if (queue->front == NULL) {

queue->rear = NULL;

}

free(temp);

return dequeued;

}

int peek(Queue\* queue) {

if (isEmpty(queue)) {

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

return -1;

}

return queue->front->data;

}

void displayQueue(Queue\* queue) {

if (isEmpty(queue)) {

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

return;

}

printf("Queue elements: ");

QueueNode\* current = queue->front;

while (current != NULL) {

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

current = current->next;

}

printf("\n");

}

int main() {

Queue\* queue = createQueue();

int choice, data;

do {

printf("\n1. Enqueue\n2. Dequeue\n3. Peek\n4. Display\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to enqueue into the queue: ");

scanf("%d", &data);

enqueue(queue, data);

break;

case 2:

printf("Dequeued %d from the queue.\n", dequeue(queue));

break;

case 3:

printf("Front element of the queue: %d\n", peek(queue));

break;

case 4:

displayQueue(queue);

break;

case 5:

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

break;

default:

printf("Invalid choice! Please enter a valid option.\n");

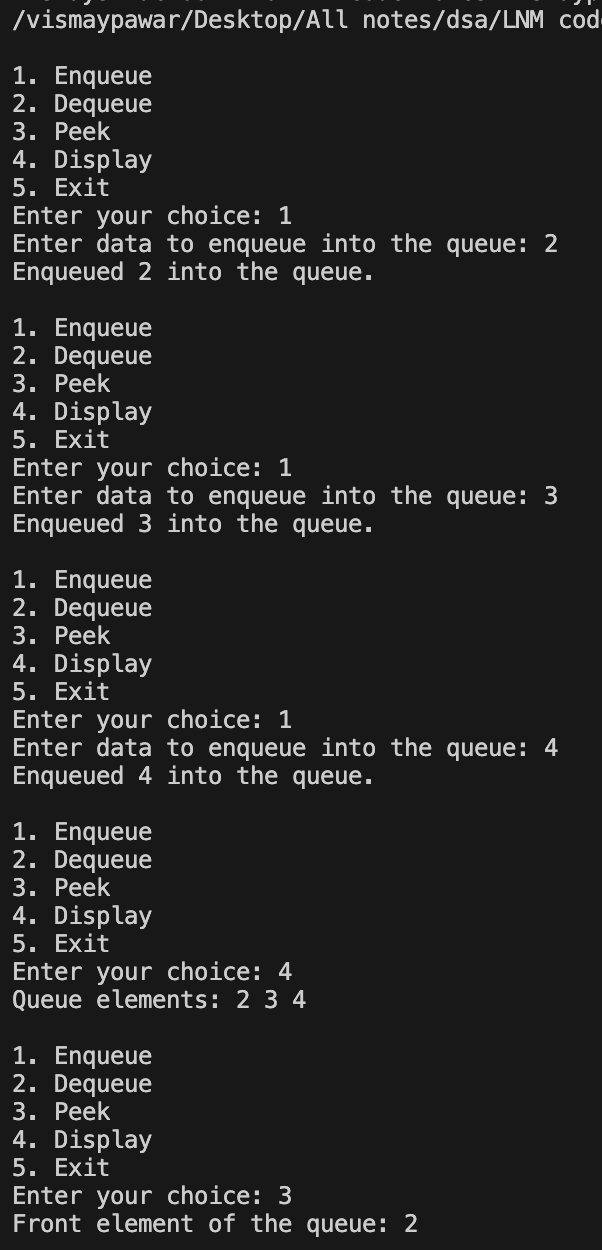
}

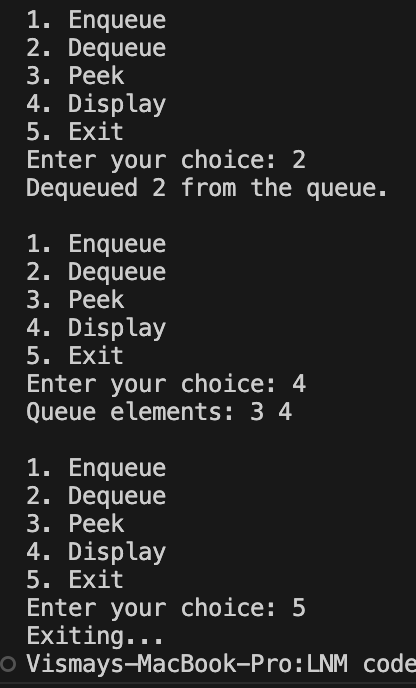
} while (choice != 5);

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>

typedef struct Node {

int data;

struct Node\* prev;

struct Node\* next;

} Node;

Node\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

void insertLeft(Node\*\* head, Node\* node, int data) {

Node\* newNode = createNode(data);

newNode->next = node;

newNode->prev = node->prev;

if (node->prev != NULL) {

node->prev->next = newNode;

} else {

\*head = newNode;

}

node->prev = newNode;

}

void deleteNode(Node\*\* head, int key) {

Node\* current = \*head;

while (current != NULL) {

if (current->data == key) {

if (current->prev != NULL) {

current->prev->next = current->next;

} else {

\*head = current->next;

}

if (current->next != NULL) {

current->next->prev = current->prev;

}

free(current);

return;

}

current = current->next;

}

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

}

void displayList(Node\* head) {

if (head == NULL) {

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

return;

}

printf("List elements: ");

while (head != NULL) {

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

head = head->next;

}

printf("\n");

}

void freeList(Node\* head) {

Node\* current = head;

Node\* temp;

while (current != NULL) {

temp = current;

current = current->next;

free(temp);

}

}

int main() {

Node\* head = NULL;

int choice, data, value;

do {

printf("\n1. Create a Doubly Linked List\n2. Insert a new node to the left of a node\n3. Delete a node based on a specific value\n4. Display the contents of the list\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the number of elements to create the list: ");

scanf("%d", &data);

printf("Enter the elements: ");

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

int value;

scanf("%d", &value);

if (head == NULL) {

head = createNode(value);

} else {

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

Node\* newNode = createNode(value);

temp->next = newNode;

newNode->prev = temp;

}

}

break;

case 2:

if (head == NULL) {

printf("List is empty. Create a list first.\n");

break;

}

printf("Enter the value of the node to the left of which you want to insert a new node: ");

scanf("%d", &value);

printf("Enter the data of the new node: ");

scanf("%d", &data);

Node\* current = head;

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

current = current->next;

}

if (current == NULL) {

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

} else {

insertLeft(&head, current, data);

}

break;

case 3:

if (head == NULL) {

printf("List is empty. Create a list first.\n");

break;

}

printf("Enter the value of the node you want to delete: ");

scanf("%d", &data);

deleteNode(&head, data);

break;

case 4:

displayList(head);

break;

case 5:

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

break;

default:

printf("Invalid choice! Please enter a valid option.\n");

}

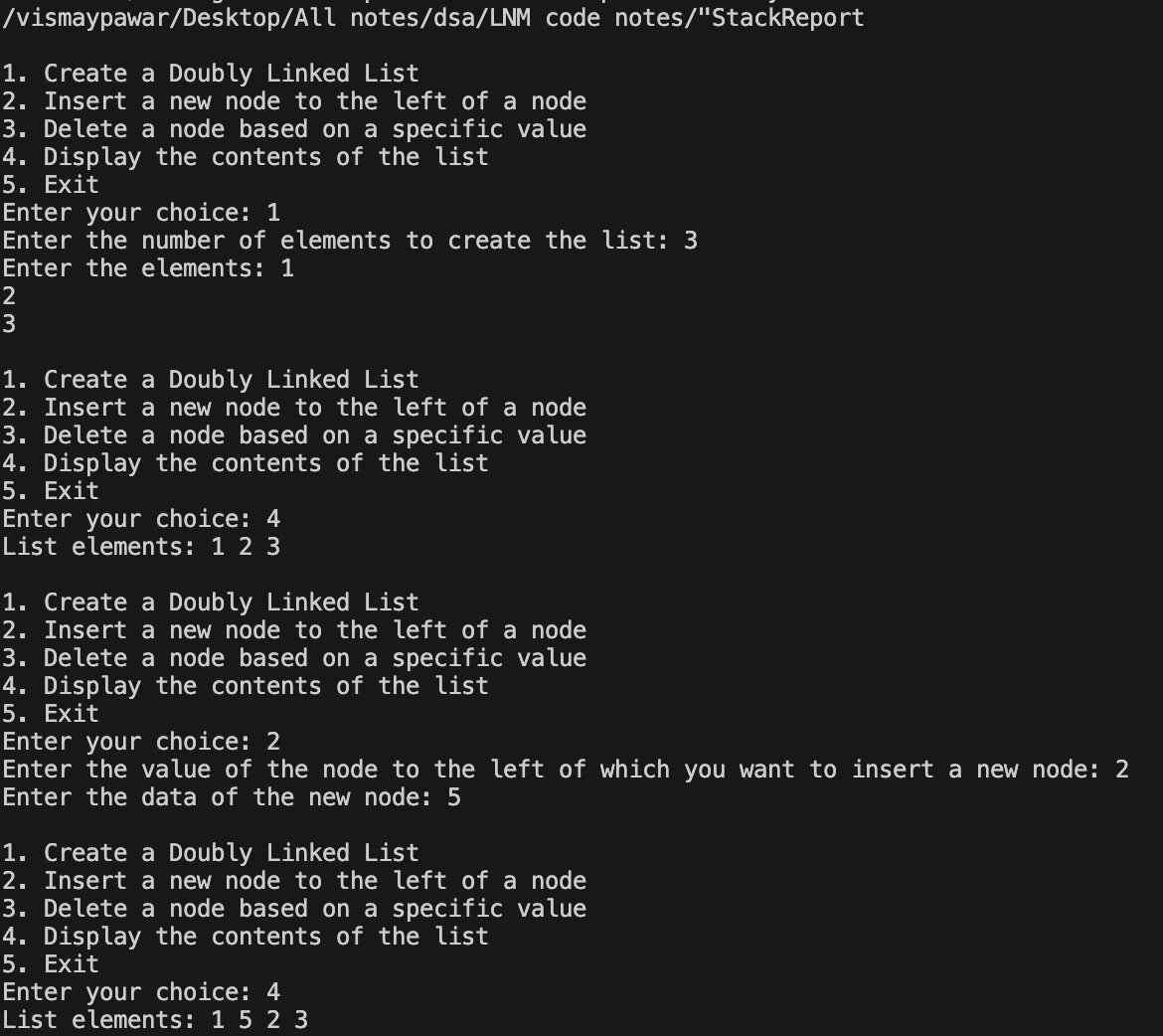
} while (choice != 5);

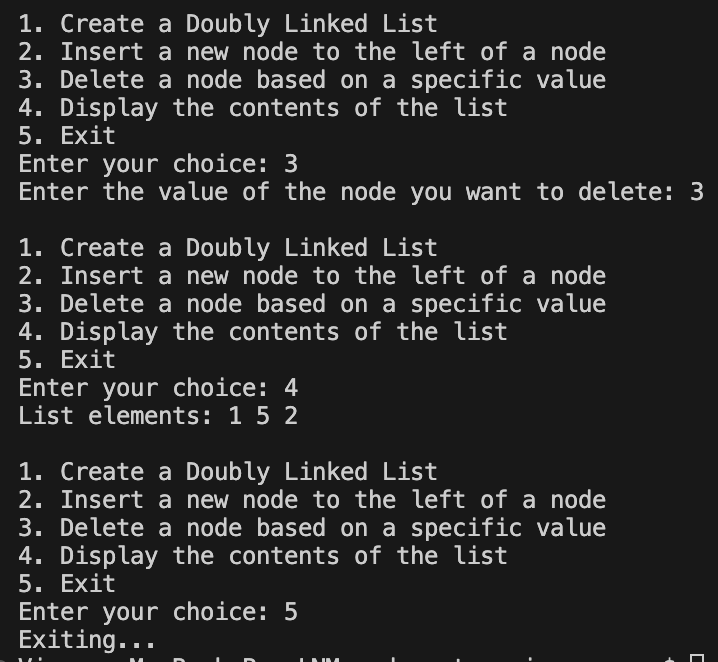
freeList(head);

return 0;

}

**Output:**

****

****

**LeetCode Problem:**

**ScoreOfParentheses:**

int scoreOfParentheses(char\* s) {

int n=strlen(s),ans=0;

int d=0,i=0;

while(i<n) {

if(s[i]=='(') d++;

else {

d--;

if(i>0 && s[i-1]=='(') ans+=1<<d;

}

i++;

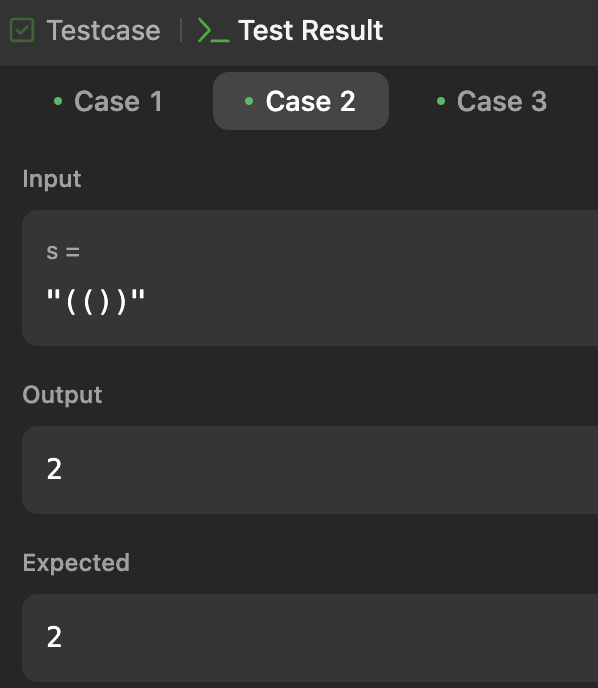
}

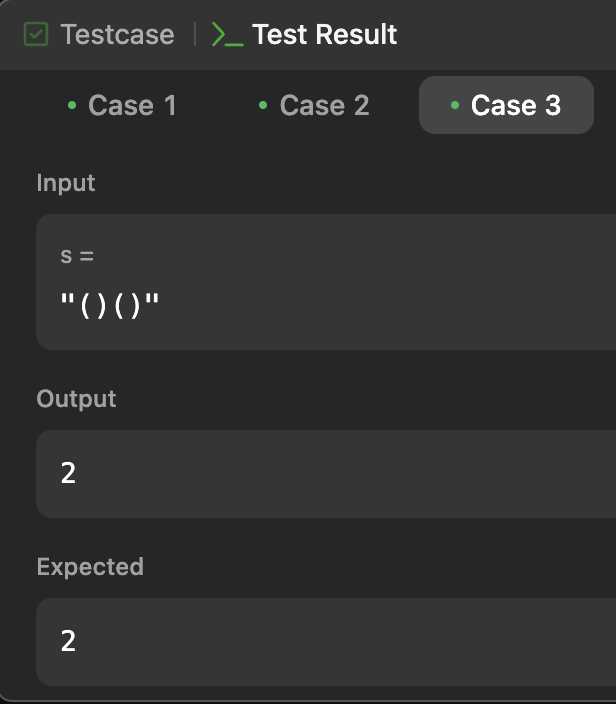
return ans;

}

**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 andpost order

c) To display the elements in the tree.

#include <stdio.h>

#include <stdlib.h>

typedef struct TreeNode {

int data;

struct TreeNode\* left;

struct TreeNode\* right;

} TreeNode;

TreeNode\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

TreeNode\* insertNode(TreeNode\* root, int data) {

if (root == NULL) {

return createNode(data);

}

if (data < root->data) {

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

} else if (data > root->data) {

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

}

return root;

}

void inorderTraversal(TreeNode\* root) {

if (root != NULL) {

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

}

void preorderTraversal(TreeNode\* root) {

if (root != NULL) {

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

preorderTraversal(root->left);

preorderTraversal(root->right);

}

}

void postorderTraversal(TreeNode\* root) {

if (root != NULL) {

postorderTraversal(root->left);

postorderTraversal(root->right);

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

}

}

void displayTree(TreeNode\* root) {

printf("Elements in the tree (inorder traversal): ");

inorderTraversal(root);

printf("\n");

}

int main() {

TreeNode\* root = NULL;

int choice, data;

do {

printf("\n1. Insert\n2. Inorder Traversal\n3. Preorder Traversal\n4. Postorder Traversal\n5. Display Tree\n6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert into the tree: ");

scanf("%d", &data);

root = insertNode(root, data);

break;

case 2:

printf("Inorder Traversal: ");

inorderTraversal(root);

printf("\n");

break;

case 3:

printf("Preorder Traversal: ");

preorderTraversal(root);

printf("\n");

break;

case 4:

printf("Postorder Traversal: ");

postorderTraversal(root);

printf("\n");

break;

case 5:

displayTree(root);

break;

case 6:

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

break;

default:

printf("Invalid choice! Please enter a valid option.\n");

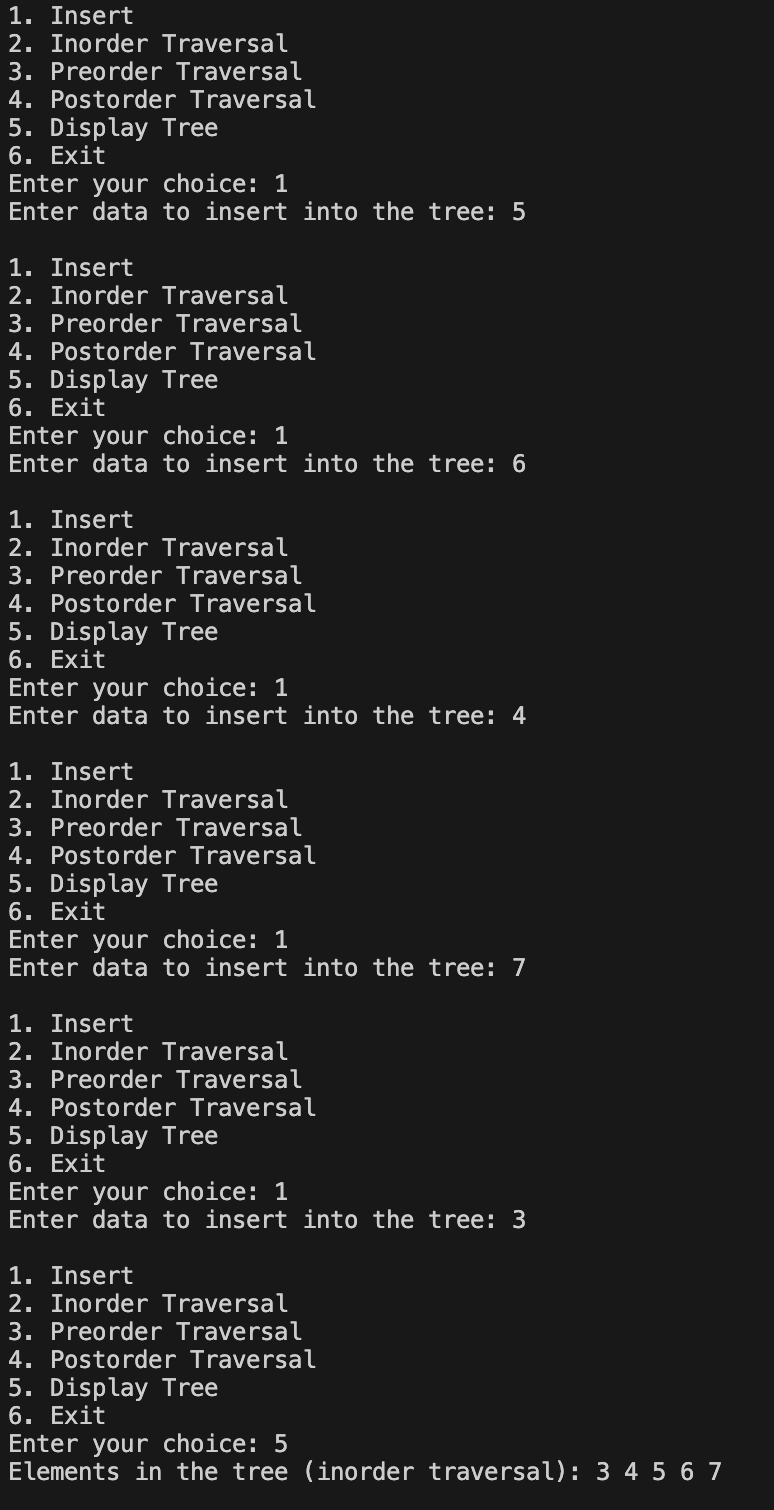
}

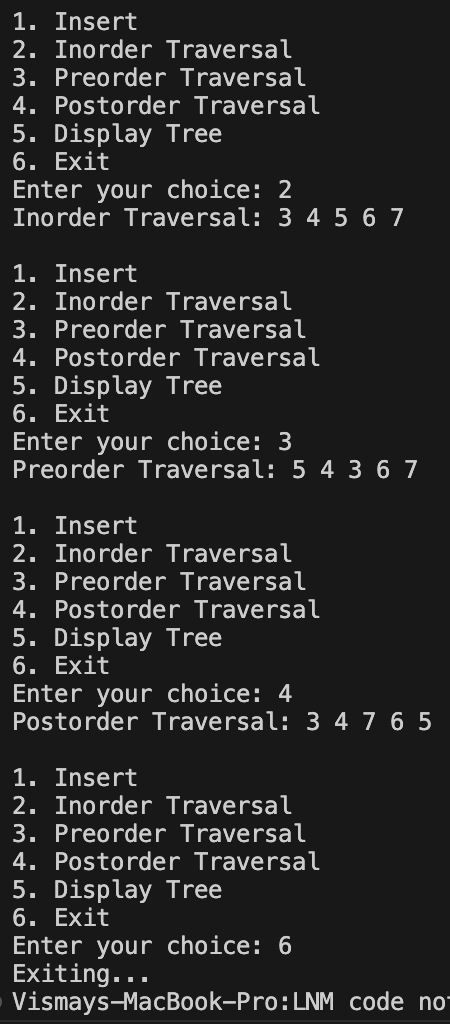
} while (choice != 6);

return 0;

}

**Output:**

****

****

**Leet Code Problem:**

**Delete the Middle Node Of a Linked List:**

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

if (head == NULL) return NULL;

struct ListNode\* prev = (struct ListNode\*)malloc(sizeof(struct ListNode));

prev->val = 0;

prev->next = head;

struct ListNode\* slow = prev;

struct ListNode\* fast = head;

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

slow = slow->next;

fast = fast->next->next;

}

struct ListNode\* temp = slow->next;

slow->next = slow->next->next;

free(temp);

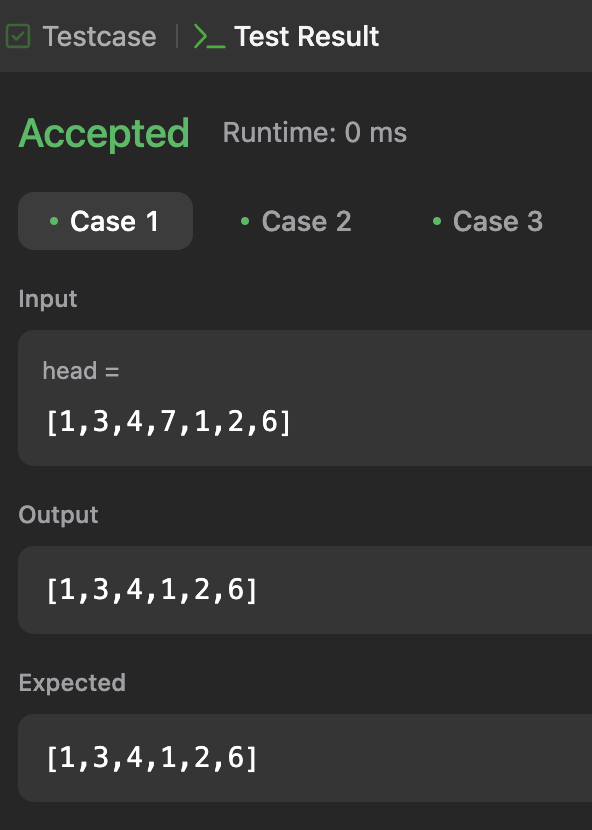
struct ListNode\* newHead = prev->next;

free(prev);

return newHead;

}

**Output:**

****

**Odd Even Linked List**

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

if(head==NULL || head->next==NULL)

return head;

struct ListNode\* oddH = NULL, \*oddT = NULL, \*evenH = NULL, \*evenT = NULL;

struct ListNode\* curr = head;

int i = 1;

while(curr != NULL){

if(i%2 != 0){

if(oddH == NULL){

oddH = curr;

oddT = curr;

}

else{

oddT -> next = curr;

oddT = curr;

}

}

else{

if(evenH == NULL){

evenH = curr;

evenT = curr;

}

else{

evenT -> next = curr;

evenT = curr;

}

}

i++;

curr = curr -> next;

}

evenT -> next = NULL;

oddT -> next = NULL;

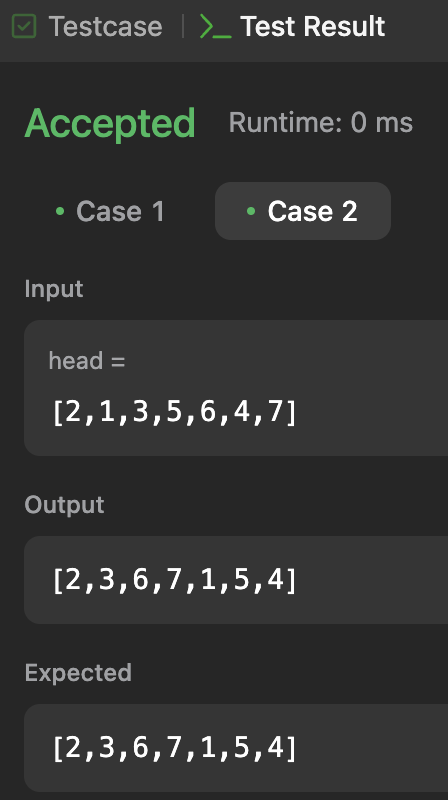
oddT->next = evenH;

return oddH;

}

**Output:**

****

****

**Lab program 9:**

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

#include <stdio.h>

void bfs(int a[10][10], int n, int u) {

int f = 0, r = -1, q[10] = {0}, v, s[10] = {0};

printf("The nodes visited from %d: ", u);

q[++r] = u;

s[u] = 1;

printf("%d ", u);

while (f <= r) {

u = q[f++];

for (v = 0; v < n; v++) {

if (a[u][v] == 1 && s[v] == 0) {

printf("%d ", v);

s[v] = 1;

q[++r] = v;

}

}

}

printf("\n");

}

int main() {

int n, a[10][10], source, i, j;

printf("\nEnter the number of nodes: ");

scanf("%d", &n);

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

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

for (j = 0; j < n; j++) {

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

}

}

for (source = 0; source < n; source++) {

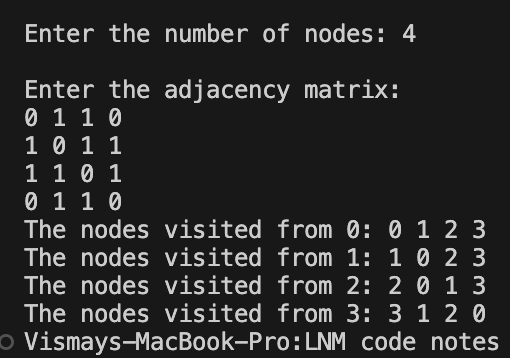
bfs(a, n, source);

}

return 0;

}

**Output:**

****

**b)Write a program to check wheater given graph is connected or not using DFS method**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 100

int n;

int a[MAX\_SIZE][MAX\_SIZE];

int s[MAX\_SIZE];

void dfs(int v) {

s[v] = 1;

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

if (a[v][i] && !s[i]) {

dfs(i);

}

}

}

int main() {

int i, j, count = 0;

printf("\nEnter number of vertices: ");

scanf("%d", &n);

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

s[i] = 0;

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

a[i][j] = 0;

}

}

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

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

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

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

}

}

dfs(1);

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

if (s[i]) {

count++;

}

}

if (count == n) {

printf("Graph is connected\n");

} else {

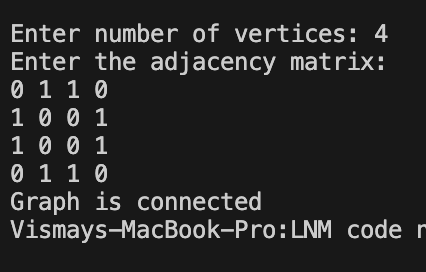
printf("Graph is not connected\n");

}

return 0;

}

**Output:**

****

**LeetCode Problem:**

**a)Delete Node In BST**

struct TreeNode\* deleteNode(struct TreeNode\* root, int key) {

if (root) {

if (key < root->val)

root->left = deleteNode(root->left, key);

else if (key > root->val)

root->right = deleteNode(root->right, key);

else {

if (!root->left && !root->right)

return NULL;

if (!root->left || !root->right)

return root->left ? root->left : root->right;

struct TreeNode\* temp = root->left;

while (temp->right != NULL)

temp = temp->right;

root->val = temp->val;

root->left = deleteNode(root->left, temp->val);

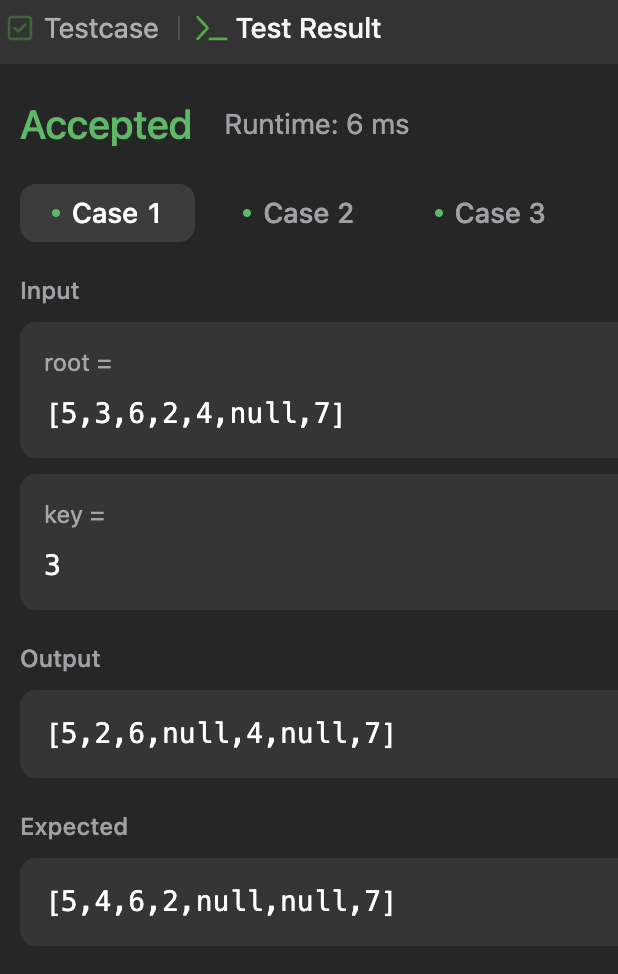
}

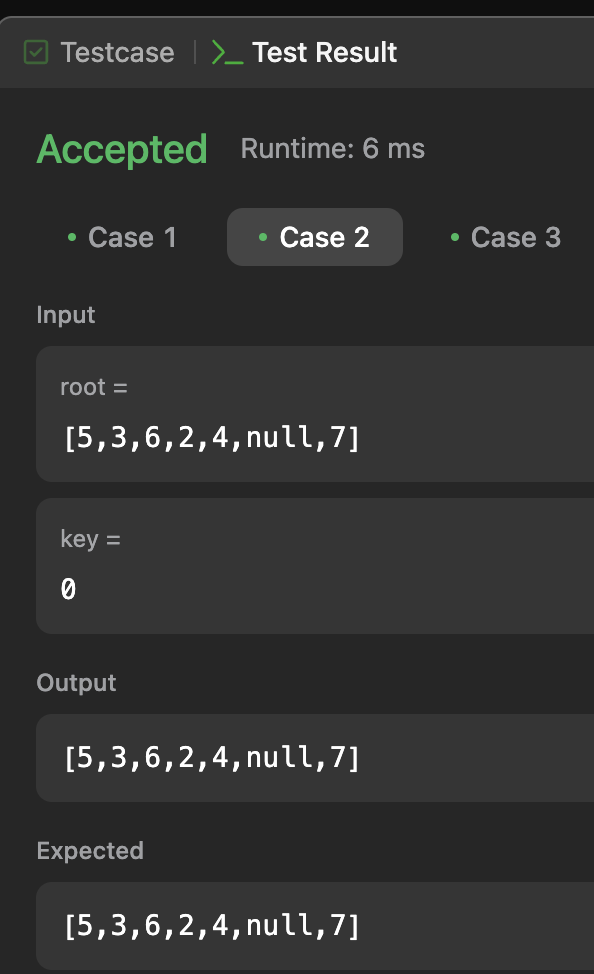
}

return root;

}

**Output:**

****

****

****

**b)Find Bottom Left Tree Value**

int findBottomLeftValue(struct TreeNode\* root) {

int value=root->val;

int mdepth=0;

void transverse(struct TreeNode\* p,int depth){

if(!p)

return;

if(depth>mdepth){

mdepth=depth;

value=p->val;

}

transverse(p->left,depth+1);

transverse(p->right,depth+1);

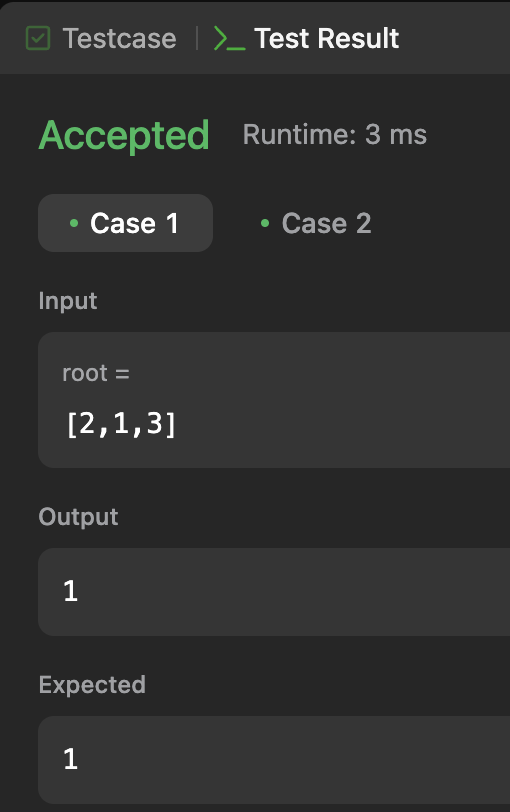
}

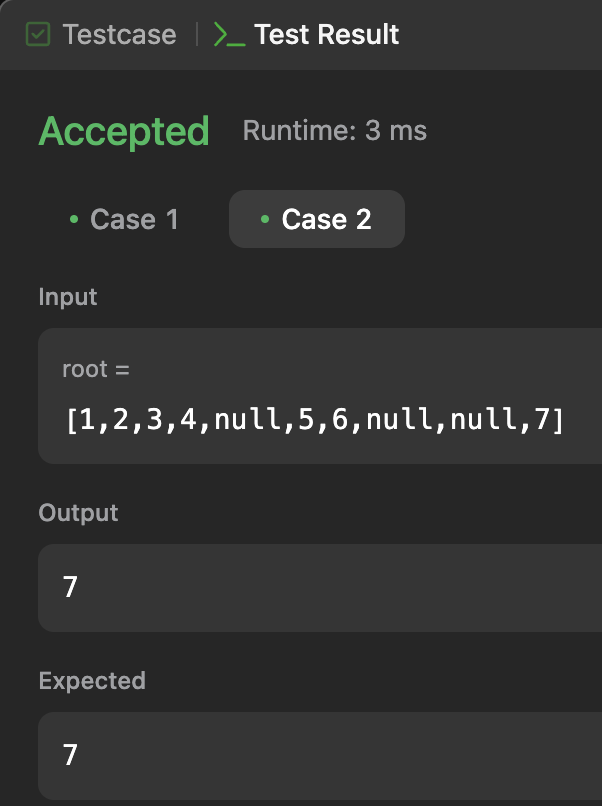
transverse(root,0);

return value;

}

**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>

#define MAX\_EMPLOYEES 100

#define HT\_SIZE 10

typedef struct {

int key;

} Employee;

typedef struct {

Employee\* entries[HT\_SIZE];

} HashTable;

int hash(int key) {

return key % HT\_SIZE;

}

void initHashTable(HashTable\* ht) {

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

ht->entries[i] = NULL;

}

}

void insertEmployee(HashTable\* ht, Employee\* emp) {

int index = hash(emp->key);

while (ht->entries[index] != NULL) {

index = (index + 1) % HT\_SIZE;

}

ht->entries[index] = emp;

}

void displayHashTable(HashTable\* ht) {

printf("\nHash Table:\n");

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

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

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

} else {

printf("Index %d: Empty\n", i);

}

}

}

int main() {

HashTable ht;

initHashTable(&ht);

int n;

printf("Enter the number of employee records: ");

scanf("%d", &n);

printf("Enter the employee keys:\n");

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

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

if (emp == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

scanf("%d", &emp->key);

insertEmployee(&ht, emp);

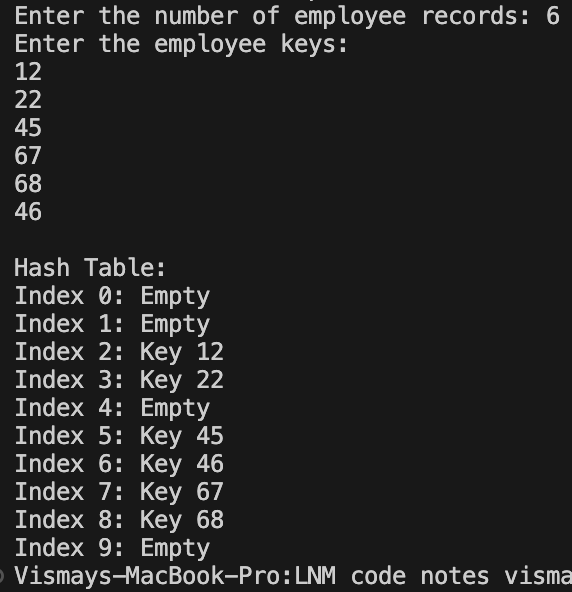
}

displayHashTable(&ht);

return 0;

}

**Output:**

****