VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

RITHVIK KUMAR R K(1BM23CS269)

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 RITHVIK KUMAR R K **(1BM23CS269)**, 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.

Prof. Namratha M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

LeetCode Programs

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:

```
Stack menu
1. Push
2.Pop
3.Display
4.Exit

Enter your choice1

Enter value to be pushed:5
The value 5 has been pushed onto the stack
Stack menu
1. Push
2.Pop
3.Display
4.Exit

Enter your choice 1

Enter value to be pushed: 6
The value 6 has been pushed onto the stack
Stack menu
1. Push
2.Pop
3.Display
4.Exit
```

```
Enter your choice1

Enter value to be pushed:12
The value 12 has been pushed onto the stack
Stack menu
1. Push
2.Pop
3.Display
4.Exit

Enter your choice2

12 value popped from the stack
Stack menu
1. Push
2.Pop
3.Display
4.Exit

Enter your choice3

12
6
5
```

```
Stack menu
1. Push
2.Pop
3.Display
4.Exit

Enter your choice4

Exiting the program
Process returned 0 (0x0)   execution time : 32.236 s
Press any key to continue.
```

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>

#define MAX 100

char stack[MAX];

int top = -1;

void push(char c) {
    if (top < MAX - 1) {
        stack[++top] = c;
    }
}

char pop() {
    return (top >= 0) ? stack[top--] : '\0';
}

char peek() {
    return (top >= 0) ? stack[top] : '\0';
}

int precedence(char c) {
    if (c == '+' || c == '-') return 1;
    if (c == '*' || c == '/') return 2;
    return 0;
}

int isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}

int isOperand(char c){
     return(c>='0'&&c<='9')||(c>='A'&&c<='Z')||(c>='a'&&c<='z');
}
```

```
void infixToPostfix(char* infix, char* postfix) {
    int j = 0; // Postfix index

    for (int i = 0; infix[i] != '\0'; i++) {
        char c = infix[i];

        if (isOperand(c)) { // Operand
            postfix[j++] = c;
        } else if (c == '(') {
            push(c);
        } else if (c == ')') {
            while (peek() != '(') {
                postfix[j++] = pop();
            }
            pop(); // Remove '(' from stack
        } else if (isOperator(c)) {
            while (top != -1 && precedence(peek()) >= precedence(c)) {
                postfix[j++] = pop();
            }
            push(c);
        }
    }

   while (top != -1) {
        postfix[j++] = pop();
    }
    postfix[j] = '\0'; // Null-terminate the postfix expression
}
```

```
int main() {
    char infix[MAX], postfix[MAX];
    printf("Enter a valid parenthesized infix expression: ");
    fgets(infix, sizeof(infix), stdin);
    infixToPostfix(infix, postfix);
    printf("Postfix expression: %s\n", postfix);
    return 0;
}
```

OUTPUT:

```
Enter a valid parenthesized infix expression: (A*B/C(E*F^G))
Postfix expression: AB*CEFG*/
```

Lab Program 3:

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.

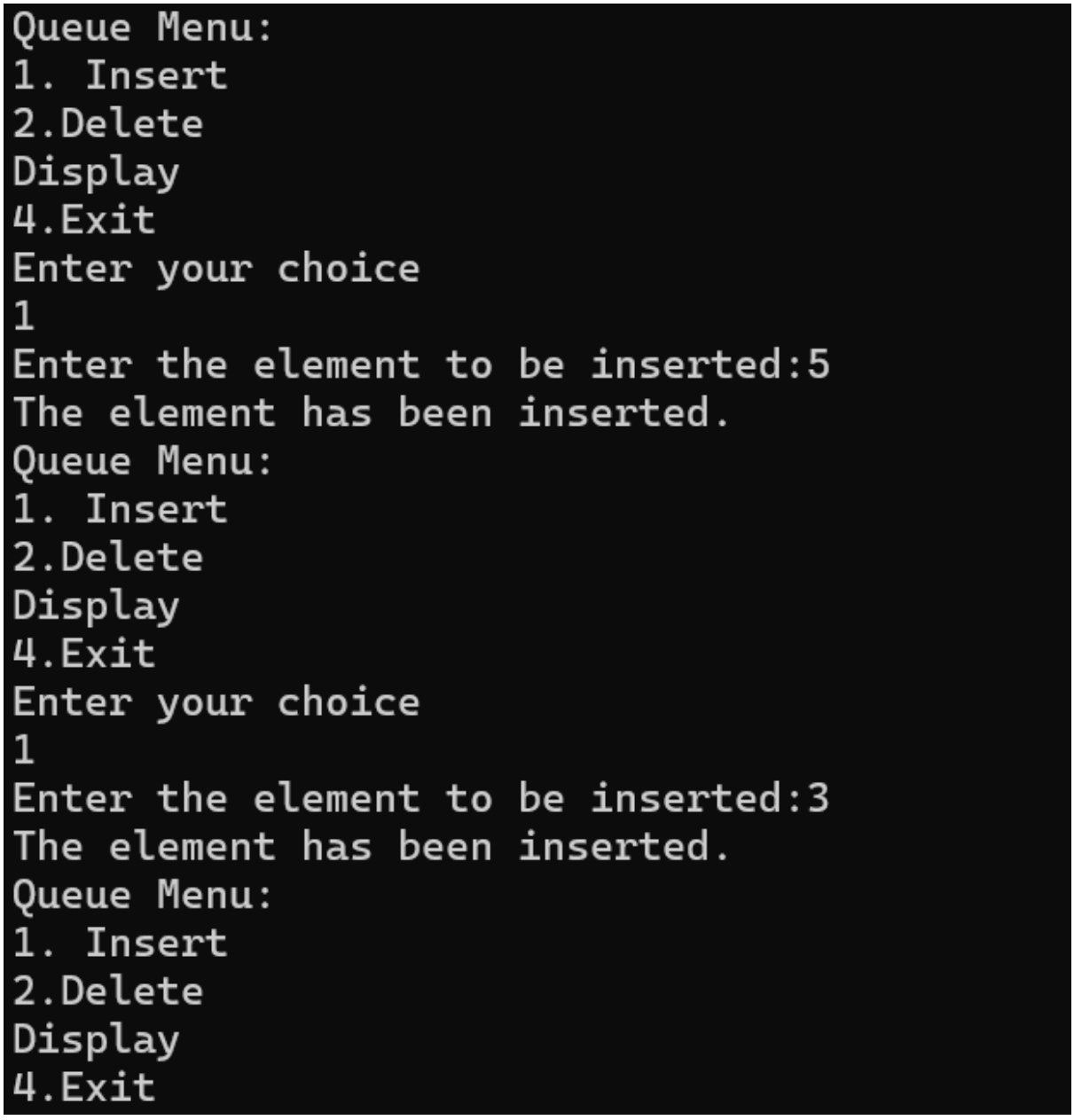
```
#include<stdio.h>
#define MAX 5
void enqueue(int queue[], int *front, int *rear, int value)
{
  if(*rear==MAX-1)
    printf("Queue is full. Element cannot be inserted.\n");
  else
  {
    if(*front==-1)
      *front=0;
    (*rear)++;
    queue[*rear]=value;
    printf("The element has been inserted.\n");
  }
}
int dequeue(int queue[], int *front, int *rear)
{
 if(*front==-1||*front>*rear)
 {
   printf("Queue is empty. cannot delete element from the queue.\n");
   return -1;
 }
 int deletedvalue=queue[*front];
 (*front)++;
 return deletedvalue;
```

```
  if(*front>*rear)
  {
    *front=*rear=-1;
  }
}
void display(int queue[], int front, int rear)
{
  if(front==-1||front>rear)
  {
    printf("Queue is empty. Cannot display elements");

  }
  else
  {
    for(int i=front; i<=rear; i++)
      {
        printf("%d", queue[i]);
      }
    printf("\n");
  }

}
int main()
{
  int choice, value;
  int queue[MAX];
  int front=-1;
  int rear=-1;
  do
```

```
{
  printf("Queue Menu:\n");
  printf("1. Insert\n");
  printf("2.Delete\n");
  printf("Display\n");
  printf("4.Exit\n");
  printf("Enter your choice\n");
  scanf("%d", &choice);

  switch(choice)
    {
      case 1:
           printf("Enter the element to be inserted:");
           scanf("%d", &value);
           enqueue(queue, &front, &rear, value);
           break;
      case 2:
           value=dequeue(queue, &front, &rear);
           if(value!=-1)
           {
             printf("The deleted element is %d\n", value);

           }
          break;
      case 3:
           display(queue, front, rear);
           break;
      case 4:
            printf("Exited the program\n");
```

```
            break;
        default:
            printf("Enter a valid choice\n");
            break;
      }
   }while(choice!=4);
}
```

OUTPUT:



```
Enter your choice
2
The deleted element is 5
Queue Menu:
1. Insert
2.Delete
Display
4.Exit
Enter your choice
3
3
Queue Menu:
1. Insert
2.Delete
Display
4.Exit
Enter your choice
4
Exited the program

Process returned 0 (0x0)   execution time : 42.481 s
Press any key to continue.
```

Lab program 4:

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>
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 insertatfirst(struct node **head, int data)
{
   struct node *newnode = CreateNode(data);
   newnode->next = *head;
   *head = newnode;
}

void inseratend(struct node **head, int data)
{
   struct node *newnode = CreateNode(data);
   if (*head == NULL)
```

```
    {
        *head = newnode;
    }
    else
    {
        struct node *temp = *head;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = newnode;
    }
}
void insertatposition(struct node **head, int data, int pos)
{
    if (pos == 0)
    {
        insertatfirst(head, data);
    }
    else
    {
        struct node *newnode = CreateNode(data);
        struct node *temp = *head;
        for (int i = 0; temp != NULL && i < pos - 1; i++)
        {
            temp = temp->next;
        }
        newnode->next = temp->next;
        temp->next = newnode;
```

```
    }
}
void display(struct node **head)
{
    if (*head == NULL)
    {
        printf("Empty");
    }
    else
    {
        struct node *temp = *head;
        while (temp != NULL)
        {
            printf("%d->", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}
int main()
{
    struct node **head = NULL;
    while (1)
    {
        printf("To insert\n");
        printf("1 for at beginning\n");
        printf("2 for at end\n");
        printf("3 for at position\n");
        printf("4 to display\n");
```

```
printf("5 to exit\n");
int n;
scanf("%d", &n);
if (n == 1)
{
   int data;
   printf("Enter the value\n");
   scanf("%d", &data);
   insertatfirst(&head, data);
}
else if (n == 2)
{
   int data;
   printf("Enter the value\n");
   scanf("%d", &data);
   inseratend(&head, data);
}
else if (n == 3)
{
   int data;
   printf("Enter the value\n");
   scanf("%d", &data);
   printf("Enter the position");
   int pos;
   scanf("%d", &pos);
   if (pos < 0)
   {
      printf("invalid input");
      break;
```

```
        }
        insertatposition(&head, data, pos);
      }
      else if (n == 4)
      {
        display(&head);
      }
      else
        break;
   }
}
```

OUTPUT:

```
To insert
1 for at beginning
2 for at end
3 for at position
4 to display
5 to exit
1
Enter the value
5
To insert
1 for at beginning
2 for at end
3 for at position
4 to display
5 to exit
2
Enter the value
6
To insert
1 for at beginning
2 for at end
3 for at position
4 to display
5 to exit
3
Enter the value
1
Enter the position1
```

```
To insert
1 for at beginning
2 for at end
3 for at position
4 to display
5 to exit
4
5->1->6->NULL
To insert
1 for at beginning
2 for at end
3 for at position
4 to display
5 to exit
5

Process returned 0 (0x0)   execution time : 34.632 s
Press any key to continue.
```

LAB PROGRAM 5:

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 insert(struct node **head, int data)
{
   struct node *newnode = CreateNode(data);
   newnode->next = *head;
   *head = newnode;
}
void display(struct node **head)
{
   if (*head == NULL)
   {
      printf("Empty");
```

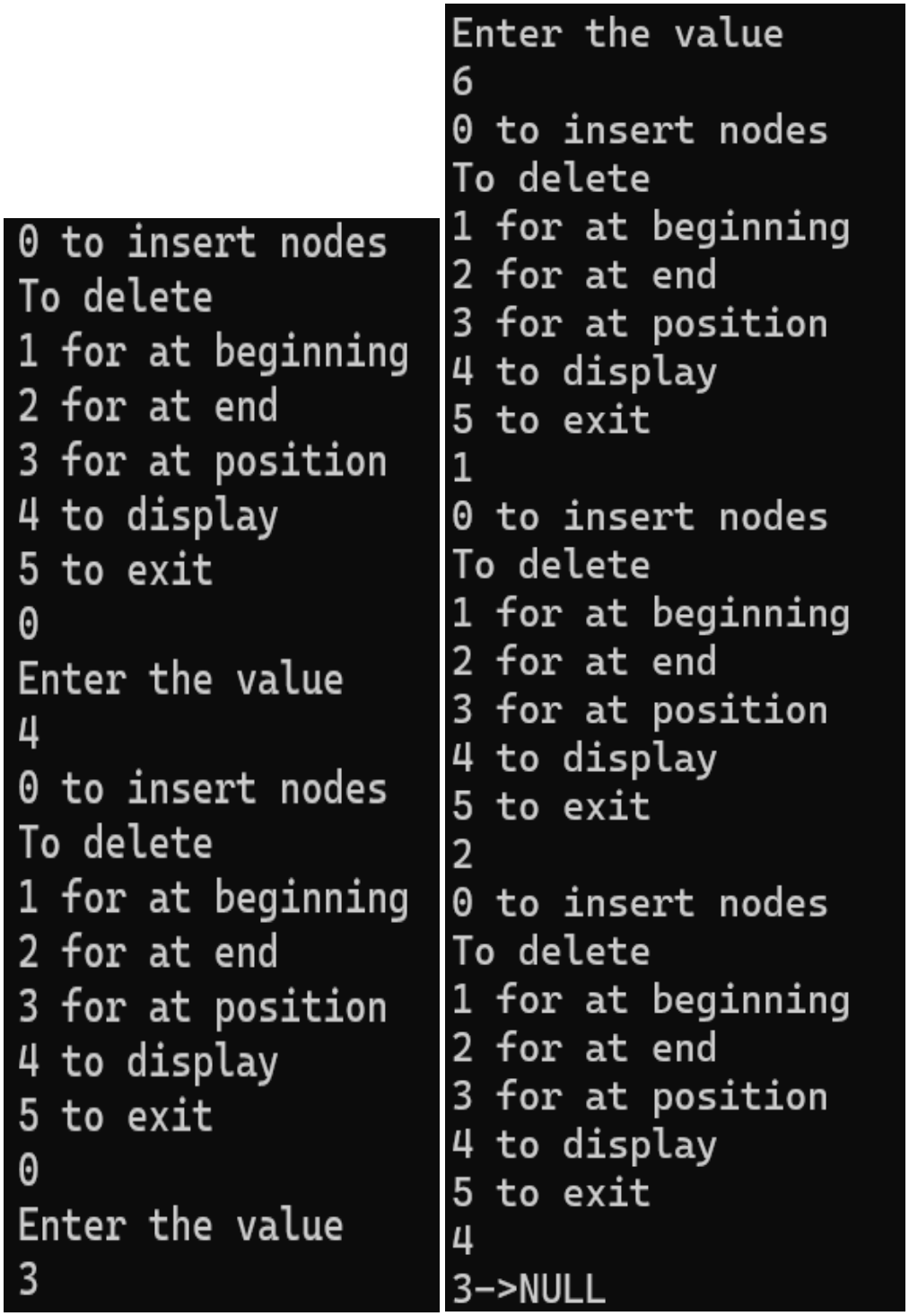
```
    }
    else
    {
        struct node *temp = *head;
        while (temp != NULL)
        {
            printf("%d->", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

void deleteatfirst(struct node **head)
{
    if (*head == NULL)
    {
        printf("Empty\n");
    }
    else
    {
        struct node *temp = *head;
        *head = temp->next;
        free(temp);
    }
}

void deleteatend(struct node **head)
{
```

```
    if (*head == NULL)
    {
        printf("Empty\n");
    }
    else
    {
        struct node *temp = *head;
        while (temp->next->next != NULL)
        {
            temp = temp->next;
        }
        free(temp->next);
        temp->next = NULL;
    }
}

void deleteatposition(struct node **head, int pos)
{
    if (*head == NULL)
    {
        printf("Empty\n");
    }
    else
    {
        struct node *temp = *head;
        for (int i = 0; temp != NULL && i < pos - 1; i++)
        {
            temp = temp->next;
        }
```

```
        struct node *afternode = temp->next->next;
        free(temp->next);
        temp->next = afternode;
    }
}

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

    while (1)
    {
        printf("0 to insert nodes\n");
        printf("To delete\n");
        printf("1 for at beginning\n");
        printf("2 for at end\n");
        printf("3 for at position\n");
        printf("4 to display\n");
        printf("5 to exit\n");
        int n;
        scanf("%d", &n);
        if (n == 0)
        {
            int data;
            printf("Enter the value\n");
            scanf("%d", &data);
            insertatfirst(&head, data);
        }
```

```
        else if(n==1)
        {
            deleteatfirst(&head);
        }
        else if (n == 2)
        {
            deleteatend(&head);
        }
        else if (n == 3)
        {
            ;
            printf("Enter the position");
            int pos;
            scanf("%d", &pos);
            if (pos < 0)
            {
                printf("invalid input");
                break;
            }
            deleteatposition(&head, pos);
        }
        else if (n == 4)
        {
            display(&head);
        }
        else
            break;
    }
}
```

OUPUT:


0 to insert nodes
To delete
1 for at beginning
2 for at end
3 for at position
4 to display
5 to exit
0
Enter the value
4
0 to insert nodes
To delete
1 for at beginning
2 for at end
3 for at position
4 to display
5 to exit
0
Enter the value
3
Enter the value
6
0 to insert nodes
To delete
1 for at beginning
2 for at end
3 for at position
4 to display
5 to exit
1
0 to insert nodes
To delete
1 for at beginning
2 for at end
3 for at position
4 to display
5 to exit
2
0 to insert nodes
To delete
1 for at beginning
2 for at end
3 for at position
4 to display
5 to exit
4
3->NULL

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;
};
struct node *CreateNode(int data)
{
   struct node *newnode = (struct node *)malloc(sizeof(struct node));
   newnode->data = data;
   newnode->next = NULL;
   return newnode;
}
void insertatfirst(struct node **head, int data)
{
   struct node *newnode = CreateNode(data);
   newnode->next = *head;
   *head = newnode;
}
struct node *Concate(struct node *head1,struct node *head2){
   struct node *temp = head1;
   while(temp->next!=NULL){
      temp = temp->next;
   }
   temp->next = head2;
```

```
    return head1;
};
struct node *Sort(struct node *head){
    struct node *temp,*current;
    current = head;
    while(current!=NULL){
        temp = head;
        while(temp->next!=NULL){
            if(temp->data > temp->next->data){
                int t = temp->data;
                temp->data = temp->next->data;
                temp->next->data = t;
            }
            temp = temp->next;
        }
        current = current->next;
    }

    return head;
}
struct node *Reverse(struct node *head){
    struct node *temp,*prev,*next;
    prev = NULL;
    temp = head;

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

```
        temp = next;
    }
    head = prev;

    return head;
}
void display(struct node **head)
{
    if (*head == NULL)
    {
        printf("Empty");
    }
    else
    {
        struct node *temp = *head;
        while (temp != NULL)
        {
            printf("%d->", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}
int main() {
    struct node *head1 = NULL;
    struct node *head2 = NULL;
    int choice, value;

    printf("Enter 5 elements for Linked List 1:\n");
```

```
for (int i = 0; i < 5; i++) {
    printf("Enter element %d: ", i + 1);
    scanf("%d", &value);
    insertatfirst(&head1, value);
}
printf("Enter 5 elements for Linked List 2:\n");
for (int i = 0; i < 5; i++) {
    printf("Enter element %d: ", i + 1);
    scanf("%d", &value);
    insertatfirst(&head2, value);
}
printf("Initial Linked Lists:\n");
display(&head1);
display(&head2);
do {
    printf("\nChoose an operation:\n");
    printf("1. Sort\n");
    printf("2. Reverse\n");
    printf("3. Concatenate\n");
    printf("4. Display Linked Lists\n");
    printf("0. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            head1 = Sort(head1);
            head2 = Sort(head2);
            printf("Sorted Linked Lists:\n");
```

```
            display(&head1);
            display(&head2);
            break;
        case 2:
            head1 = Reverse(head1);
            head2 = Reverse(head2);
            printf("reversed Linked Lists:\n");
            display(&head1);
            display(&head2);
            break;
        case 3:
            head1 = Concate(head1, head2);
            printf("concatenated Linked Lists:\n");
            display(&head1);
            display(&head2);
            break;
        case 4:
            printf("Current Linked Lists:\n");
            display(&head1);

            break;
        case 0:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 0);
```

```
    return 0;
}
```

OUTPUT:

```
Enter 5 elements for Linked List 1:
Enter element 1: 4
Enter element 2: 3
Enter element 3: 6
Enter element 4: 7
Enter element 5: 1
Enter 5 elements for Linked List 2:
Enter element 1: 0
Enter element 2: 12
Enter element 3: 23
Enter element 4: 4
Enter element 5: 56
Initial Linked Lists:
1->7->6->3->4->NULL
56->4->23->12->0->NULL

Choose an operation:
1. Sort
2. Reverse
3. Concatenate
4. Display Linked Lists
0. Exit
Enter your choice: 3
concatenated Linked Lists:
1->7->6->3->4->56->4->23->12->0->NULL
56->4->23->12->0->NULL
```

```
Choose an operation:
1. Sort
2. Reverse
3. Concatenate
4. Display Linked Lists
0. Exit
Enter your choice: 1
Sorted Linked Lists:
0->1->3->4->4->6->7->12->23->56->NULL
6->7->12->23->56->NULL

Choose an operation:
1. Sort
2. Reverse
3. Concatenate
4. Display Linked Lists
0. Exit
Enter your choice: 2
reversed Linked Lists:
56->23->12->7->6->NULL
0->1->3->4->4->6->NULL
```

6b) WAP to Implement Single Link List
to simulate Stack & Queue
Operations. (stack and queue to be implemented as one program)

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
};
struct node *front = NULL, *rear = NULL;
struct node *head = NULL;
void enqueue(int val) {
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = val;
    newNode->next = NULL;

    if (front == NULL && rear == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
}
void dequeue() {
    struct node *temp;
    if (front == NULL) {
        printf("Queue is Empty. Unable to perform dequeue.\n");
    } else {
        temp = front;
```

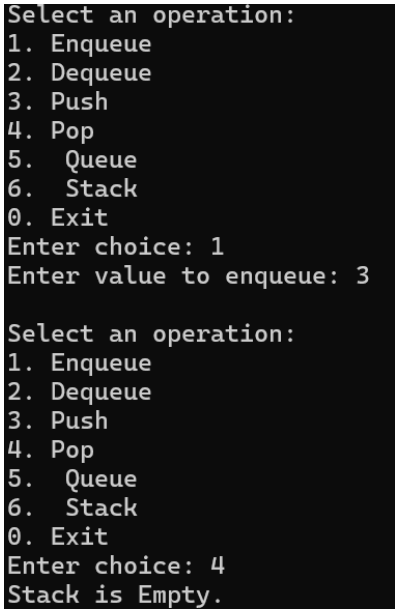
```
        front = front->next;
        if (front == NULL) {
            rear = NULL;
        }
        printf("Dequeued element = %d\n", temp->data);
        free(temp);
    }
}
void printQueue() {
    struct node *temp = front;
    printf("Queue: ");
    while (temp) {
        printf("%d->", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
void push(int val) {
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = val;
    newNode->next = head;
    head = newNode;
}
void pop() {
    struct node *temp;
    if (head == NULL) {
        printf("Stack is Empty.\n");
    } else {
        printf("Popped element = %d\n", head->data);
```

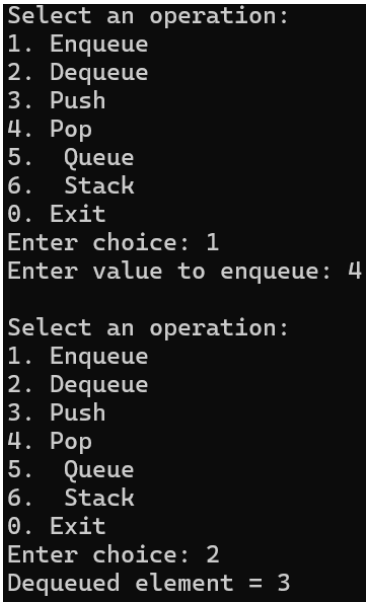
```
        temp = head;
        head = head->next;
        free(temp);
    }
}
void printStack() {
    struct node *temp = head;
    printf("Stack: ");
    while (temp != NULL) {
        printf("%d->", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
int main() {
    int choice, value;

    do {
        printf("\nSelect an operation:\n");
        printf("1. Enqueue \n");
        printf("2. Dequeue \n");
        printf("3. Push \n");
        printf("4. Pop \n");
        printf("5.  Queue\n");
        printf("6.  Stack\n");
        printf("0. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
```

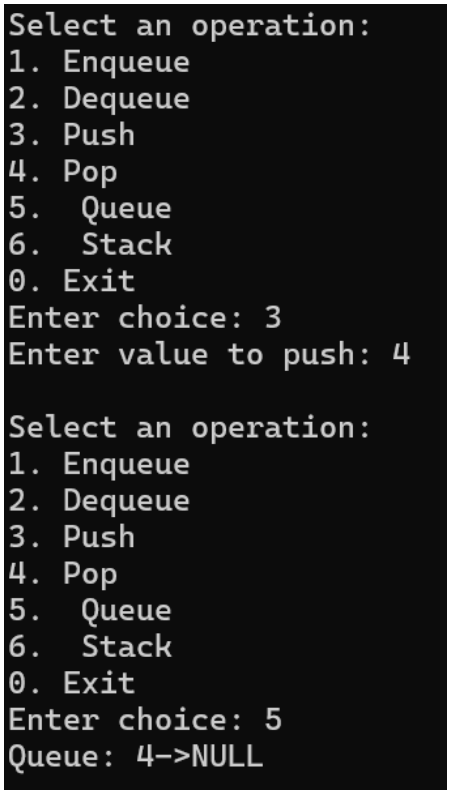
```
switch (choice) {
    case 1:
        printf("Enter value to enqueue: ");
        scanf("%d", &value);
        enqueue(value);
        break;
    case 2:
        dequeue();
        break;
    case 3:
        printf("Enter value to push: ");
        scanf("%d", &value);
        push(value);
        break;
    case 4:
        pop();
        break;
    case 5:
        printQueue();
        break;
    case 6:
        printStack();
        break;
    case 0:
        printf("Exiting program.\n");
        break;
    default:
        printf("Invalid choice. Please try again.\n");
}
```

```
    } while (choice != 0);


    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
Display the contents of the list

```
#include <stdio.h>

#include <stdlib.h>

// node creation

struct Node {

  int data;

  struct Node* next;

  struct Node* prev;

};

// insert node at the front

    void insertFront(struct Node** head, int data) {

      // allocate memory for newNode

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

      // assign data to newNode

      newNode->data = data;

      // make newNode as a head

      newNode->next = (*head);

      // assign null to prev

      newNode->prev = NULL;

      // previous of head (now head is the second node) is newNode

      if ((*head) != NULL)

        (*head)->prev = newNode;


      // head points to newNode

      (*head) = newNode;
```

```
    }
// Function to delete a node from a specified position
    void deleteAtPosition(struct Node** head, int position)
    {
        if (*head == NULL) {
            printf("The list is already empty.\n");
            return;
        }
       struct Node* temp = *head;
        if (position == 1) {
            if (*head == NULL) {
            printf("The list is already empty.\n");
            return;
        }
        struct Node* temp = *head;
        *head = (*head)->next;
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
        free(temp);
            return;
        }
        for (int i = 1; temp != NULL && i < position; i++) {
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Position is greater than the number of "
                   "nodes.\n");
            return;
```

```
        }
        if (temp->next != NULL) {
            temp->next->prev = temp->prev;
        }
        if (temp->prev != NULL) {
            temp->prev->next = temp->next;
        }
        free(temp);
    }
// print the doubly linked list
void displayList(struct Node* node) {
  struct Node* last;
  while (node != NULL) {
    printf("%d->", node->data);
    last = node;
    node = node->next;
  }
  if (node == NULL)
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL; // Initialize the doubly linked list as empty
    int choice, data, position;
    while (1) {
        printf("\n--- Doubly Linked List Operations ---\n");
        printf("1. Insert at Front\n");
        printf("2. Delete at Position\n");
        printf("3. Display List\n");
```

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

switch (choice) {
    case 1:
        printf("Enter data to insert: ");
        scanf("%d", &data);
        insertFront(&head, data);
        printf("Node inserted at the front.\n");
        break;

    case 2:
        printf("Enter the position to delete: ");
        scanf("%d", &position);
        deleteAtPosition(&head, position);
        break;

    case 3:
        printf("The list is: ");
        displayList(head);
        break;
    case 4:
        printf("Exiting the program.\n");
        // Free all remaining nodes in the list before exiting
        while (head != NULL) {
            struct Node* temp = head;
            head = head->next;
            free(temp);
```

```
            }
            return 0;

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

OUTPUT:

```
--- Doubly Linked List Operations ---
1. Insert at Front
2. Delete at Position
3. Display List
4. Exit
Enter your choice: 1
Enter data to insert: 5
Node inserted at the front.

--- Doubly Linked List Operations ---
1. Insert at Front
2. Delete at Position
3. Display List
4. Exit
Enter your choice: 1
Enter data to insert: 6
Node inserted at the front.

--- Doubly Linked List Operations ---
1. Insert at Front
2. Delete at Position
3. Display List
4. Exit
Enter your choice: 2
Enter the position to delete: 1
```

```
--- Doubly Linked List Operations ---
1. Insert at Front
2. Delete at Position
3. Display List
4. Exit
Enter your choice: 3
The list is: 5->NULL
```

LAB PROGRAM 8:

Binary search tree

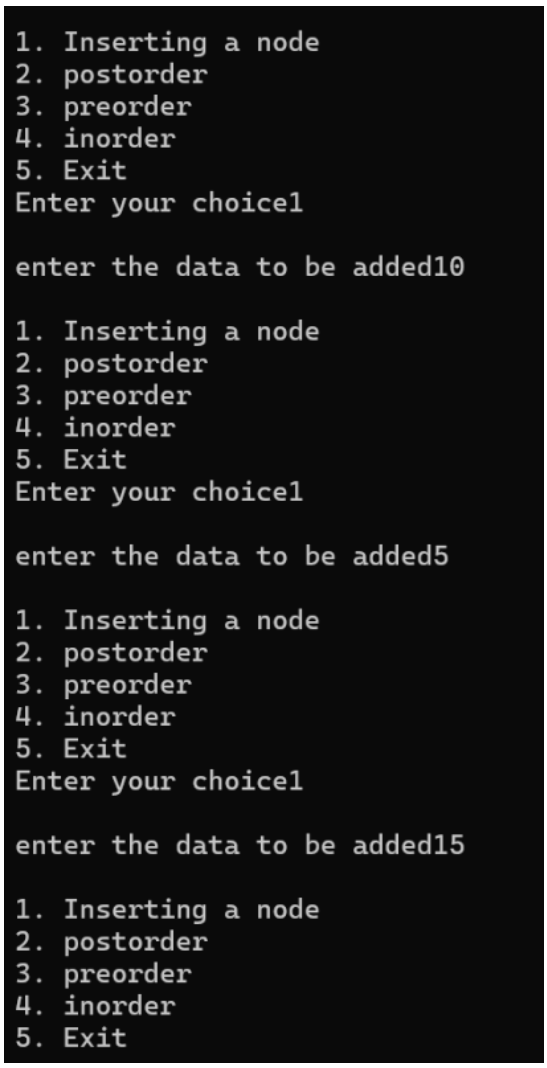
```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node
{
int data;
struct Node* left;
struct Node* right;
}Node;
Node* createNode(int data)
{
Node* newNode=(Node*)malloc(sizeof(Node));
newNode->data=data;
newNode->left=NULL;
newNode->right=NULL;
return newNode;
}
Node* insert(Node* root,int value)
{
if(root==NULL)
{
return createNode(value);
}
if(value<root->data)
{
root->left=insert(root->left,value);
}
```

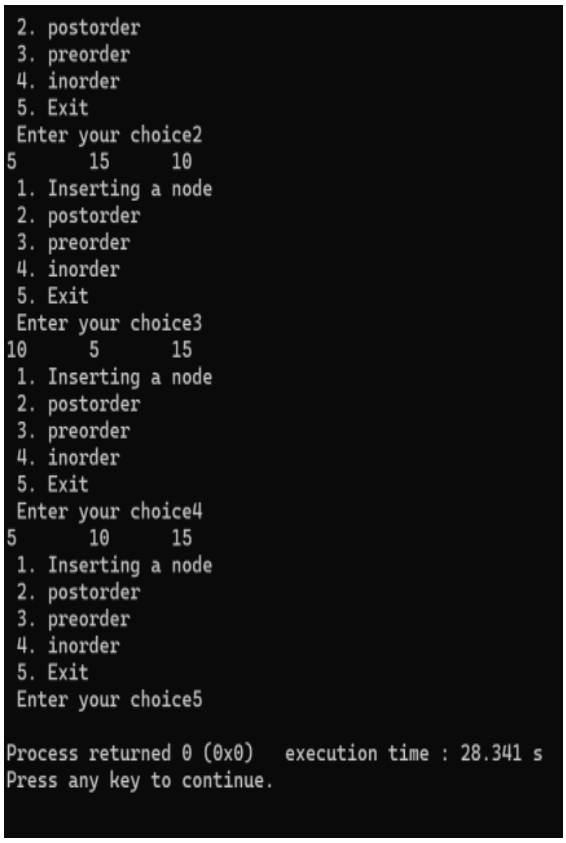
```
else if(value>root->data)
{
root->right=insert(root->right,value);
}
return root;
}
void postorder(Node* root)
{
if(root!=NULL)
{
postorder(root->left);
postorder(root->right);
printf("%d \t",root->data);
}
}
void preorder(Node* root)
{
if(root!=NULL)
{
printf("%d \t",root->data);
preorder(root->left);
preorder(root->right);
}
}
void inorder(Node* root)
{
if(root!=NULL)
{
inorder(root->left);
```

```
printf("%d \t",root->data);
inorder(root->right);

}
}
int main()
{
Node* root=NULL;
int choice,value;
while(1)
{
printf("\n 1. Inserting a node");
printf("\n 2. postorder");
printf("\n 3. preorder");
printf("\n 4. inorder");
printf("\n 5. Exit");

printf("\n Enter your choice");
scanf("%d",&choice);
switch(choice)
{
case 1:printf("\n enter the data to be added");
scanf("%d",&value);
root=insert(root,value);
break;
case 2:postorder(root);
break;
case 3:preorder(root);
break;
```

```
case 4:inorder(root);
break;
case 5:exit(0);
break;
default:printf("\n Invalid Choice");
break;
}
}
return 0;
}
```

OUTPUT:





LAB PROGRAM 9:

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

```
#include <stdio.h>
#define MAX 5
void bfs(int adj[][MAX], int visited[], int start) {
    int queue[MAX], rear = -1, front = -1, i;
    for (int k = 0; k < MAX; k++)
        visited[k] = 0;
    queue[++rear] = start;
    ++front;
    visited[start] = 1;
    while (rear >= front) {
        start = queue[front++];
        printf("%d -> ", start);
        for (i = 0; i < MAX; i++) {
            if (adj[start][i] && visited[i] == 0) {
                queue[++rear] = i;
                visited[i] = 1;
            }
        }
    }
}
int main() {
    int visited[MAX] = {0};
    int adj[MAX][MAX], i, j;
    printf("Enter the adjacency matrix of the graph (%d x %d):\n", MAX, MAX);
    for (i = 0; i < MAX; i++)
        for (j = 0; j < MAX; j++)
            scanf("%d", &adj[i][j]);
```

```
    printf("BFS Traversal starting from node 0:\n");
    bfs(adj, visited, 0);
    return 0;
}
```

OUTPUT:

```
Enter the adjacency matrix of the graph (5 x 5):
1 0 0 1 1
1 0 1 0 0
1 1 0 1 0
0 0 0 0 1
1 1 1 0 0
BFS Traversal starting from node 0:
0 -> 3 -> 4 -> 1 -> 2 ->
Process returned 0 (0x0)   execution time : 50.693 s
Press any key to continue.
```

LAB PROGRAM

9B) DFS TRAVERSAL

```
#include <stdio.h>
#define MAX 5\
void dfs(int adj[][MAX], int visited[], int start) {
    int stack[MAX], top = -1, i;\
    for (int k = 0; k < MAX; k++)
        visited[k] = 0;\
    stack[++top] = start;
    visited[start] = 1;\
    while (top != -1) {
        start = stack[top--];
        printf("%d -> ", start);

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

            if (adj[start][i] && visited[i] == 0) {
                stack[++top] = i;
                visited[i] = 1;
            }
        }
    }
}
int main() {
    int visited[MAX] = {0};
    int adj[MAX][MAX], i, j;
```

```
    printf("Enter the adjacency matrix of the graph (%d x %d):\n", MAX, MAX);
    for (i = 0; i < MAX; i++)
        for (j = 0; j < MAX; j++)
            scanf("%d", &adj[i][j]);\
    // Perform DFS from the first node (0)
    printf("DFS Traversal starting from node 0:\n");
    dfs(adj, visited, 0);
    return 0;
}
```

OUTPUT:

```
Enter the adjacency matrix of the graph (5 x 5):
1 0 0 1 1
1 0 1 0 0
1 1 0 1 0
0 0 0 0 1
1 1 1 0 0
DFS Traversal starting from node 0:
0 -> 4 -> 2 -> 1 -> 3 ->
Process returned 0 (0x0)   execution time : 3.840 s
Press any key to continue.
```

LAB PROGRAM 10:

Linear Probing

```
#include <stdio.h>
#include<stdlib.h>
#define TABLE_SIZE 10
int h[TABLE_SIZE]={NULL};
void insert()
{
   int key,index,i,flag=0,hkey;
   printf("\nenter a value to insert into hash table\n");
   scanf("%d",&key);
   hkey=key%TABLE_SIZE;
   for(i=0;i<TABLE_SIZE;i++)
   {
    index=(hkey+i)%TABLE_SIZE;
       if(h[index] == NULL)
       {
       h[index]=key;
        break;
       }

   }
   printf("No of probes for %d is %d", key,i+1);
   if(i == TABLE_SIZE)
   printf("\nelement cannot be inserted\n");

}
void search()
```

```
{
int key,index,i,flag=0,hkey;
printf("\nenter search element\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
    for(i=0;i<TABLE_SIZE; i++)
    {
     index=(hkey+i)%TABLE_SIZE;
        if(h[index]==key)
        {
            printf("value is found at index %d",index);
            break;
        }
    }
    if(i == TABLE_SIZE)
    printf("\n value is not found\n");
}
void display()
{
    int i;
printf("\nelements in the hash table are \n");
 for(i=0;i< TABLE_SIZE; i++)
 printf("\nat index %d \t value =  %d",i,h[i]);
}
main()
{    int opt,i;
   while(1)
 {        printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
      scanf("%d",&opt);
```

```
    switch(opt)
    {
     case 1:insert();
            break;
     case 2:display();
            break;
    case 3:search();
            break;
    case 4:exit(0);
    }
 }
}
```

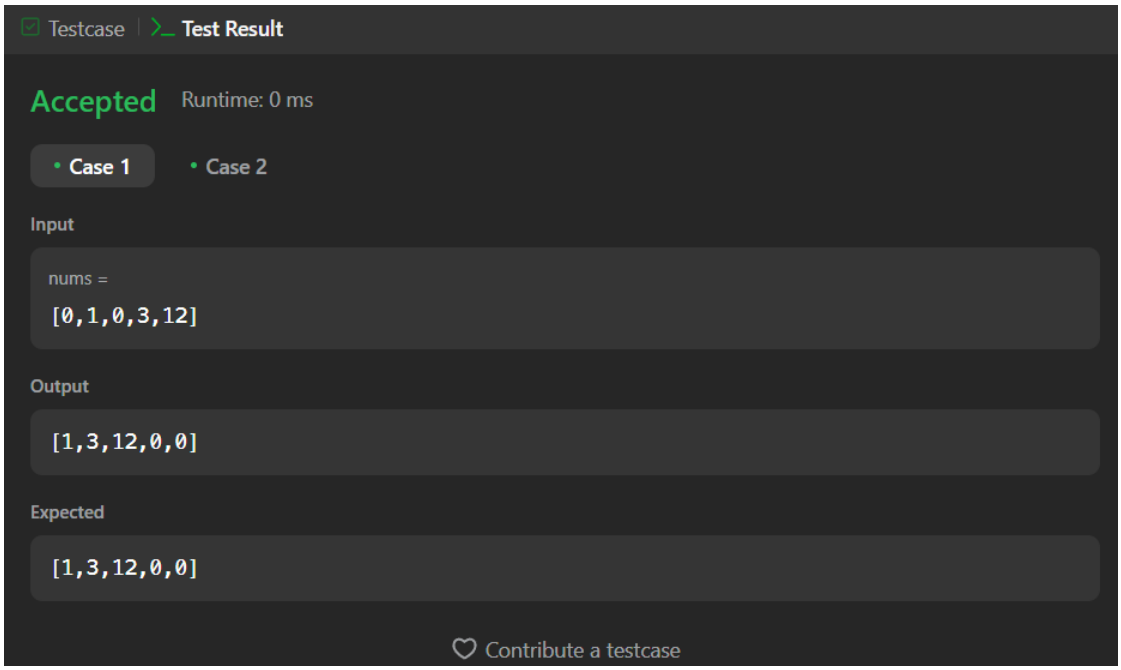
OUTPUT:

```
Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
4
No of probes for 4 is 1
Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
5
No of probes for 5 is 1
Press 1. Insert  2. Display     3. Search       4.Exit
2

elements in the hash table are

at index 0       value =  0
at index 1       value =  0
at index 2       value =  0
at index 3       value =  0
at index 4       value =  4
at index 5       value =  5
at index 6       value =  0
at index 7       value =  0
at index 8       value =  0
at index 9       value =  0
Press 1. Insert  2. Display     3. Search       4.Exit
3

enter search element
5
value is found at index 5
Press 1. Insert  2. Display     3. Search       4.Exit
```

Leetcode problems

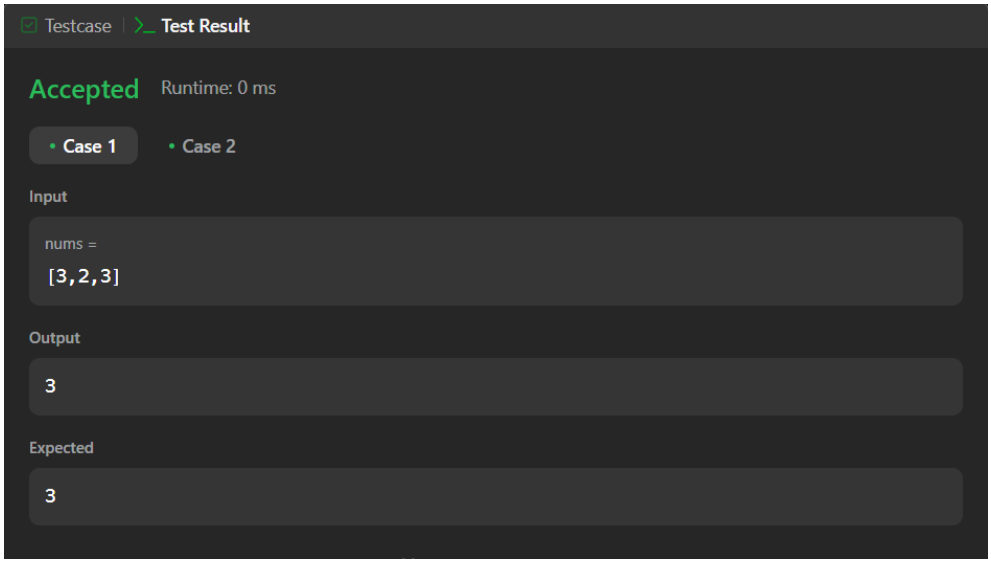
a) Update move zeroes

```
void moveZeroes(int* nums, int numsSize) {
    int i=0, k=0;
    while( i<numsSize-k )
    {
        if(nums[i] == 0)
        {
            for(int j=i; j<numsSize-1; j++)
            {
                nums[j] = nums[j+1];
            }
            nums[numsSize-1] = 0;
            k++;
        }
        if( nums[i] != 0 )
            i++;
    }
}
```



b) Majority Element

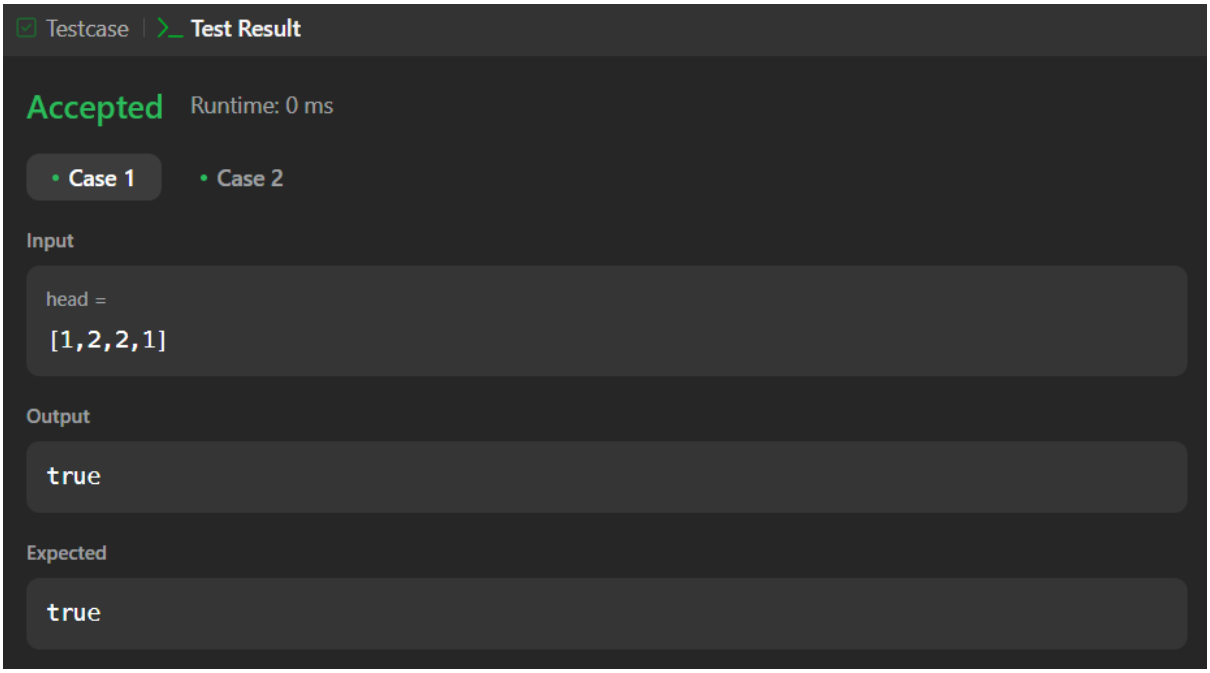
```
int majorityElement(int* nums, int numsSize) {
    int ele;
    int count;
    for(int i = 0; i<numsSize; i++)
    {
        count = 0;
        if(nums[i]==ele){
        continue;
        }
        ele = nums[i];
        for(int j = 0;j<numsSize;j++)
        {
            if(nums[j]==ele)
                count++;
        }
        if (count>(numsSize/2))
        {
            return ele;
            break;
        }

    }
    return ele;
}
```



c) Linked List Palindrome:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
bool isPalindrome(struct ListNode* head) {
    if (head == NULL || head->next == NULL) {
        return true;
    }
    struct ListNode *a = head, *b = head;
    while (b != NULL && b->next != NULL) {
        a = a->next;
        b = b->next->next;
    }
    struct ListNode *current = head;
    int stack[100000];
    int index = 0;

    while (current != a) {
        stack[index++] = current->val;
        current = current->next;
    }

    if (b != NULL) {
        a = a->next;
    }

    while (a != NULL) {
        if (a->val != stack[--index]) {
            return false;
        }
        a = a->next;
    }
    return true;
```

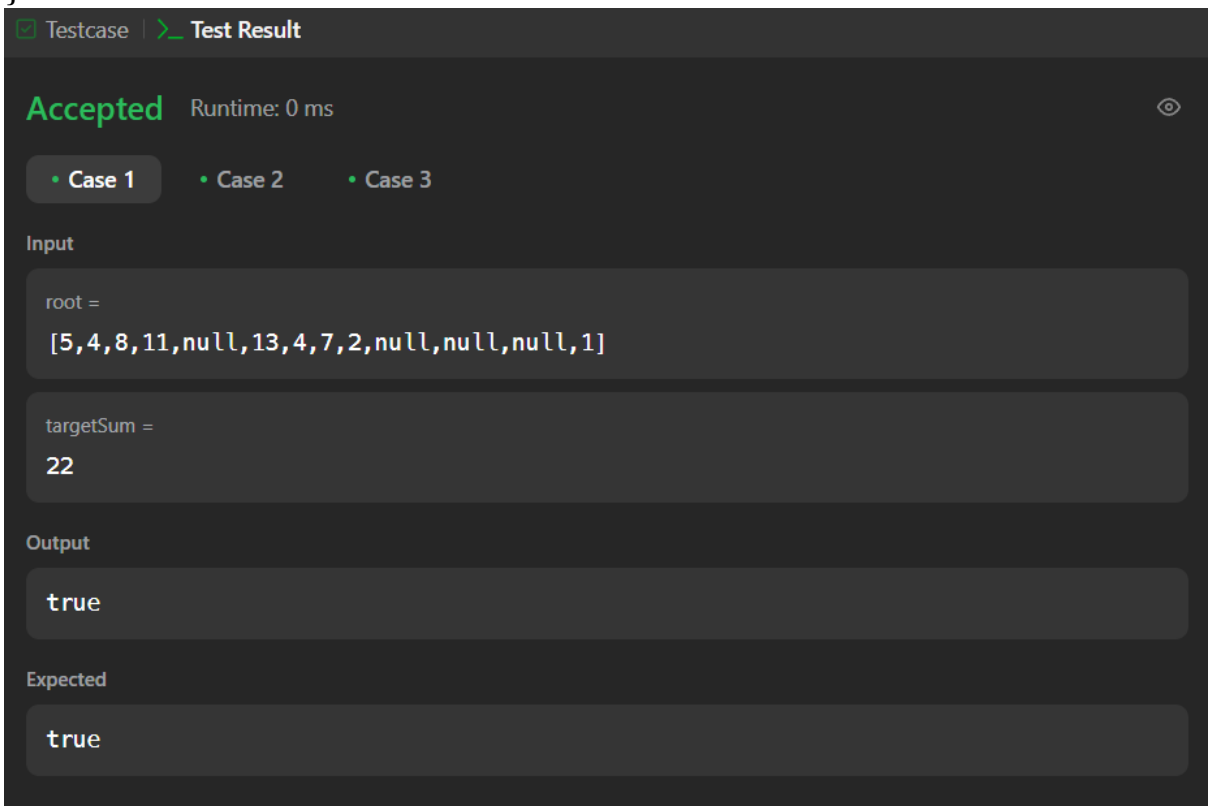
}



d) Path sum

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
bool hasPathSum(struct TreeNode* root, int targetSum) {
    if (root == NULL) {
        return false;
    }
    if (root->left == NULL && root->right == NULL) {
        return root->val == targetSum;
    }
    int remainingSum = targetSum - root->val;
    return hasPathSum(root->left, remainingSum) || hasPathSum(root-
>right, remainingSum);
```

}


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
root =
[5,4,8,11,null,13,4,7,2,null,null,null,1]
targetSum =
22
Output
true
Expected
true