VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

ROHAN B SHEKAR (1BM23CS272)

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 **ROHAN B SHEKAR (1BM23CS272)**, 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. Lakshmi Neelima M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Lab program 1:

1.a. 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>
#define MAX_SIZE 5
int top =-1;
int stack[MAX_SIZE];
void push (int value){
   if(top==MAX_SIZE -1){
   printf("Stack is full %d \n", value);
   return;
}
stack [++top]=value;
}
int pop(){
   if(top==-1){
      printf("Stack is empty %d \n");
      return -1;
   }
   return stack[top--];
}
void printstack() {
   int i;
   for (i=0; i<=top; i++){
      printf("%d", stack[i]);
   }
   printf("\n");
}
int main(){
   int flag=0;
   while(flag==0)
   {
      printf("Select one of the following stack operations \n");
      printf("1.Push \n");
      printf("2.Pop \n");
      printf("3.Display \n");
      printf("4.Exit \n");
      int option;
      printf("Enter 1,2,3 or 4 : ");
      scanf("%d",&option);
      if (option==1)
      {
         printf("Enter elements to be pushed : \n");
```

```
            int elem;
            scanf("%d",&elem);
            push(elem);
        }
        else if (option==2)
        {
            pop();
        }
        else if (option ==3)
        {
            printstack();
        }
        else if (option==4)
        {
            flag=1;
        }
        else
        {
            printf("Please enter 1,2,3 or 4 :  \n");
            return 0;
        }
    }

}
```

Output:

```
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 1
Enter elements to be pushed :
5
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 1
Enter elements to be pushed :
6
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 1
```

Enter elements to be pushed :
7
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 1
Enter elements to be pushed :
8
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 1
Enter elements to be pushed :
9
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 1
Enter elements to be pushed :
9
Stack is full 9
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 3
56789
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 2
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 2
Select one of the following stack operations
1.Push
2.Pop
3.Display

4.Exit
Enter 1,2,3 or 4 : 3
567
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 2
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 2
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 2
Select one of the following stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter 1,2,3 or 4 : 2
Stack is empty 0

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 <string.h>

int index1 = 0, pos = 0, top = -1, length;
char symbol, temp, infix[20], postfix[20], stack[20];

void infixtopostfix();
void push(char symbol);
char pop();
int pred(char symbol);

int main() {
   printf("Enter infix expression: \n");
   scanf("%s", infix);
   infixtopostfix();
   postfix[pos] = '\0';
   printf("\nInfix expression: \n%s", infix);
   printf("\nPostfix expression: \n%s\n", postfix);
   return 0;
}

void infixtopostfix() {
   length = strlen(infix);
```

```
push('#');

while (index1 < length) {
    symbol = infix[index1];
    switch (symbol) {
        case '(':
            push(symbol);
            break;
        case ')':
            temp = pop();
            while (temp != '(') {
                postfix[pos++] = temp;
                temp = pop();
            }
            break;
        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
            while (pred(stack[top]) >= pred(symbol)) {
                temp = pop();
                postfix[pos++] = temp;
            }
            push(symbol);
            break;
        default:
            postfix[pos++] = symbol;
    }
```

```
        index1++;
    }

    while (top >= 0) {
        temp = pop();
        postfix[pos++] = temp;
    }
}

void push(char symbol) {
    top++;
    stack[top] = symbol;
}

char pop() {
    return stack[top--];
}

int pred(char symbol) {
    switch (symbol) {
        case '^': return 3;
        case '*':
        case '/': return 2;
        case '+':
        case '-': return 1;
        case '(': return 0;
        case '#': return -1;
        default: return -1;
    }
```

}

Output –

Enter infix expression:

A*(B/D)-C+E

Infix expression:

A*(B/D)-C+E

Postfix expression:

ABD/*C-E+#

Lab Program – 3

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

```
#include<stdio.h>
#define SIZE 3
int queue[SIZE];
int front =-1;
int rear =-1;

void insert(int value){
   if (rear == SIZE -1){
      printf("Queue Overflow, Cannot insert %d\n",value);
   }
   else{
      if (front ==-1) front =0;
      rear++;
      queue[rear]=value;
      printf("Inserted %d into the queue \n",value);
   }
}

void delete(){
   if (front==-1 || front>rear) {
      printf("Queue Underflow, Cannot delete element \n");
   }
   else{
      printf("Deleted %d from the queue \n", queue[front]);
```

```
        front++;
        if(front>rear){
            front=rear=-1;
        }
    }
}

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



int main(){
    int choice,value;
    while(1){
        printf("\n 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:
                return 0;
            default:
                printf("Invalid choice, Please try again.\n");
        }
    }
    return 0;
}
```

Output -

```
Queue Operations:
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter the value to insert: 8
Inserted 8 into the queue

Queue Operations:
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter the value to insert: 9
Inserted 9 into the queue

Queue Operations:
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter the value to insert: 7
Inserted 7 into the queue
```

Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:1

Enter the value to insert: 5

Queue Overflow, Cannot insert 5

Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:3

Queue elements are:8 9 7

Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:2

Deleted 8 from the queue

Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:2

Deleted 9 from the queue

Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:2

Deleted 7 from the queue

Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:2

Queue Underflow, Cannot delete element

Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:4

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

```
#include<stdio.h>
#define SIZE 3
int queue[SIZE];
int front = -1;
int rear = -1;

void insert(int value){
   if ((front ==0 && rear==SIZE -1) || (rear == (front-1)%(SIZE-1))){
      printf("Queue Overflow, Cannot insert %d\n", value);
   }
   else if(front ==-1){
      front = rear =0;
      queue[rear]=value;
      printf("Inserted %d into the queue\n",value);
   }
   else if(rear ==SIZE -1 && front!=0){
      rear=0;
      queue[rear]=value;
      printf("Inserted %d into the queue\n", value);
   }

   else{
      rear++;
      queue[rear]=value;
      printf("Inserted %d into the queue\n", value);
```

```
    }
}

void delete() {
    if (front ==-1){
        printf("Queue Underflow, Cannot delete element\n");
    }
    else{
        printf("Deleted %d from the queue \n", queue[front]);
        if (front==rear){
            front=rear=-1;
        }
        else if(front ==SIZE-1){
            front=0;
        }
        else{
            front++;
        }
    }
}

void display() {
    if (front == -1) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements are: ");
    if (rear >= front) {
        for (int i = front; i <= rear; i++) {
```

```
            printf("%d ", queue[i]); // Add a space here
        }
    } else {
        for (int i = front; i < SIZE; i++) {
            printf("%d ", queue[i]); // Add a space here
        }
        for (int i = 0; i <= rear; i++) {
            printf("%d ", queue[i]); // Add a space here
        }
    }
    printf("\n");
}

int main(){
    int choice,value;
     while(1){
        printf("\n Circular 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:
            return 0;
        default:
            printf("Invalid choice, Please try again.\n");
      }
   }
   return 0;
}
```

Output -

```
Circular Queue Operations:
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter the value to insert: 8
Inserted 8 into the queue
```

Circular Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:1

Enter the value to insert: 9

Inserted 9 into the queue

Circular Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:1

Enter the value to insert: 7

Inserted 7 into the queue

Circular Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:1

Enter the value to insert: 5

Queue Overflow, Cannot insert 5

Circular Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:3

Queue elements are : 8 9 7

Circular Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:2

Deleted 8 from the queue

Circular Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:2

Deleted 9 from the queue

Circular Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:2

Deleted 7 from the queue

Circular Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:2

Queue Underflow, Cannot delete element

Circular Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:3

Queue is empty

Circular Queue Operations:

1.Insert

2.Delete

3.Display

4.Exit

Enter your choice:4

Lab Program – 4

Single Linked List Implementation

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

struct Node {
    int data;
    struct Node* next;
};
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = *head;

    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        return;
```

```
    }

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


void insertAtPosition(struct Node** head, int data, int position) {
    if (position < 1) {
        printf("Position must be greater than or equal to 1.\n");
        return;
    }

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

    if (position == 1) {
        newNode->next = *head;
        *head = newNode;
        return;
    }

    struct 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");
    } else {
        newNode->next = temp->next;
        temp->next = newNode;
    }
}


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

    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}

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

    if ((*head)->next == NULL) {
        free(*head);
```

```
        *head = NULL;
        return;
    }

    struct Node* temp = *head;
    while (temp->next != NULL && temp->next->next != NULL) {
        temp = temp->next;
    }

    free(temp->next);
    temp->next = NULL;
}


void deleteAtPosition(struct Node** head, int position) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }

    if (position == 1) {
        struct Node* temp = *head;
        *head = (*head)->next;
        free(temp);
        return;
    }

    struct Node* temp = *head;
    for (int i = 1; i < position - 1 && temp != NULL; i++) {
```

```
        temp = temp->next;
    }

    if (temp == NULL || temp->next == NULL) {
        printf("Position out of range.\n");
    } else {
        struct Node* nodeToDelete = temp->next;
        temp->next = temp->next->next;
        free(nodeToDelete);
    }
}

void printList(struct Node* head) {
    if (head == NULL) {
        printf("The 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, data, position;
```

```
while (1) {
    printf("\nSingly Linked List Operations:\n");
    printf("1. Insert at Beginning\n");
    printf("2. Insert at End\n");
    printf("3. Insert at Any Position\n");
    printf("4. Delete from Beginning\n");
    printf("5. Delete from End\n");
    printf("6. Delete from Any Position\n");
    printf("7. Print List\n");
    printf("8. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter data to insert at the beginning: ");
            scanf("%d", &data);
            insertAtBeginning(&head, data);
            break;

        case 2:
            printf("Enter data to insert at the end: ");
            scanf("%d", &data);
            insertAtEnd(&head, data);
            break;

        case 3:
            printf("Enter data to insert: ");
```

```
            scanf("%d", &data);
            printf("Enter position: ");
            scanf("%d", &position);
            insertAtPosition(&head, data, position);
            break;

        case 4:
            deleteFromBeginning(&head);
            break;

        case 5:
            deleteFromEnd(&head);
            break;

        case 6:
            printf("Enter position to delete: ");
            scanf("%d", &position);
            deleteAtPosition(&head, position);
            break;

        case 7:
            printList(head);
            break;

        case 8:
            exit(0);

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

```
        }
    }

    return 0;
}
```

Output -

Singly Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Insert at Any Position

4. Delete from Beginning

5. Delete from End

6. Delete from Any Position

7. Print List

8. Exit

Enter your choice: 1

Enter data to insert at the beginning: 6

Singly Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Insert at Any Position

4. Delete from Beginning

5. Delete from End

6. Delete from Any Position

7. Print List

8. Exit

Enter your choice: 2

Enter data to insert at the end: 1

Singly Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Insert at Any Position

4. Delete from Beginning

5. Delete from End

6. Delete from Any Position

7. Print List

8. Exit

Enter your choice: 3

Enter data to insert: 5

Enter position: 2

Singly Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Insert at Any Position

4. Delete from Beginning

5. Delete from End

6. Delete from Any Position

7. Print List

8. Exit

Enter your choice: 7

6 -> 5 -> 1 -> NULL

Singly Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Insert at Any Position

4. Delete from Beginning

5. Delete from End

6. Delete from Any Position

7. Print List

8. Exit

Enter your choice: 4

Singly Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Insert at Any Position

4. Delete from Beginning

5. Delete from End

6. Delete from Any Position

7. Print List

8. Exit

Enter your choice: 7

5 -> 1 -> NULL

Singly Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Insert at Any Position

4. Delete from Beginning

5. Delete from End

6. Delete from Any Position

7. Print List

8. Exit

Enter your choice: 5

Singly Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Insert at Any Position

4. Delete from Beginning

5. Delete from End

6. Delete from Any Position

7. Print List

8. Exit

Enter your choice: 7

5 -> NULL

Singly Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Insert at Any Position

4. Delete from Beginning

5. Delete from End

6. Delete from Any Position

7. Print List

8. Exit

Enter your choice: 6

Enter position to delete: 1

Singly Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Insert at Any Position

4. Delete from Beginning

5. Delete from End

6. Delete from Any Position

7. Print List

8. Exit

Enter your choice: 7

The list is empty.

Lab Program – 5

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 insertEnd(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 displayList(struct Node* head) {
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

void sortList(struct Node** head) {
    if (*head == NULL) return;

    struct Node* i = *head;
    while (i != NULL) {
        struct Node* j = i->next;
        while (j != NULL) {
            if (i->data > j->data) {
                int temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
            j = j->next;
        }
        i = i->next;
    }
}
```

```
void reverseList(struct Node** head) {
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* 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, n, i;

do {
    printf("\nMenu:\n");
    printf("1. Insert into List 1\n");
    printf("2. Insert into List 2\n");
    printf("3. Display List 1\n");
    printf("4. Display List 2\n");
    printf("5. Sort List 1\n");
    printf("6. Reverse List 1\n");
    printf("7. Concatenate List 2 into List 1\n");
    printf("8. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter number of elements to insert in List 1: ");
            scanf("%d", &n);
            for (i = 0; i < n; i++) {
                printf("Enter value: ");
                scanf("%d", &value);
                insertEnd(&list1, value);
            }
            break;

        case 2:
            printf("Enter number of elements to insert in List 2: ");
```

```
            scanf("%d", &n);
            for (i = 0; i < n; i++) {
                printf("Enter value: ");
                scanf("%d", &value);
                insertEnd(&list2, value);
            }
            break;

        case 3:
            printf("List 1: ");
            displayList(list1);
            break;

        case 4:
            printf("List 2: ");
            displayList(list2);
            break;

        case 5:
            sortList(&list1);
            printf("List 1 sorted.\n");
            break;

        case 6:
            reverseList(&list1);
            printf("List 1 reversed.\n");
            break;

        case 7:
```

```
            concatenateLists(&list1, &list2);
            printf("List 2 concatenated into List 1.\n");
            break;

        case 8:
            printf("Exiting...\n");
            break;

        default:
            printf("Invalid choice!\n");
      }
   } while (choice != 8);

   return 0;
}
```

Output –

```
Menu:
1. Insert into List 1
2. Insert into List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 2 into List 1
```

8. Exit

Enter your choice: 1

Enter number of elements to insert in List 1: 4

Enter value: 1

Enter value: 2

Enter value: 3

Enter value: 4

Menu:

1. Insert into List 1

2. Insert into List 2

3. Display List 1

4. Display List 2

5. Sort List 1

6. Reverse List 1

7. Concatenate List 2 into List 1

8. Exit

Enter your choice: 2

Enter number of elements to insert in List 2: 2

Enter value: 6

Enter value: 7

Menu:

1. Insert into List 1

2. Insert into List 2

3. Display List 1

4. Display List 2

5. Sort List 1

6. Reverse List 1

7. Concatenate List 2 into List 1

8. Exit

Enter your choice: 3

List 1: 1 -> 2 -> 3 -> 4 -> NULL

Menu:

1. Insert into List 1

2. Insert into List 2

3. Display List 1

4. Display List 2

5. Sort List 1

6. Reverse List 1

7. Concatenate List 2 into List 1

8. Exit

Enter your choice: 4

List 2: 6 -> 7 -> NULL

Menu:

1. Insert into List 1

2. Insert into List 2

3. Display List 1

4. Display List 2

5. Sort List 1

6. Reverse List 1

7. Concatenate List 2 into List 1

8. Exit

Enter your choice: 5

List 1 sorted.

Menu:

1. Insert into List 1

2. Insert into List 2

3. Display List 1

4. Display List 2

5. Sort List 1

6. Reverse List 1

7. Concatenate List 2 into List 1

8. Exit

Enter your choice: 6

List 1 reversed.

Menu:

1. Insert into List 1

2. Insert into List 2

3. Display List 1

4. Display List 2

5. Sort List 1

6. Reverse List 1

7. Concatenate List 2 into List 1

8. Exit

Enter your choice: 3

List 1: 4 -> 3 -> 2 -> 1 -> NULL

Menu:

1. Insert into List 1

2. Insert into List 2

3. Display List 1

4. Display List 2

5. Sort List 1

6. Reverse List 1

7. Concatenate List 2 into List 1

8. Exit

Enter your choice: 7

List 2 concatenated into List 1.

Menu:

1. Insert into List 1

2. Insert into List 2

3. Display List 1

4. Display List 2

5. Sort List 1

6. Reverse List 1

7. Concatenate List 2 into List 1

8. Exit

Enter your choice: 3

List 1: 4 -> 3 -> 2 -> 1 -> 6 -> 7 -> NULL

Lab Program - 6

WAP to implement Single Link List to simulate Stack and Queue Operations.

```
#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 push(struct Node** top, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *top;
    *top = newNode;
    printf("Pushed %d onto the stack.\n", data);
}

int pop(struct Node** top) {
    if (*top == NULL) {
        printf("Stack underflow! Cannot pop.\n");
```

```
        return -1;
    }
    struct Node* temp = *top;
    int poppedData = temp->data;
    *top = temp->next;
    free(temp);
    return poppedData;
}

void enqueue(struct Node** front, struct Node** rear, int data) {
    struct Node* newNode = createNode(data);
    if (*rear == NULL) {
        *front = *rear = newNode;
        printf("Enqueued %d into the queue.\n", data);
        return;
    }
    (*rear)->next = newNode;
    *rear = newNode;
    printf("Enqueued %d into the queue.\n", data);
}

int dequeue(struct Node** front) {
    if (*front == NULL) {
        printf("Queue underflow! Cannot dequeue.\n");
        return -1;
    }
    struct Node* temp = *front;
    int dequeuedData = temp->data;
    *front = temp->next;
```

```
    if (*front == NULL) {
        *front = NULL;
    }
    free(temp);
    return dequeuedData;
}

void displayList(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* stackTop = NULL;
    struct Node* queueFront = NULL;
    struct Node* queueRear = NULL;
    int choice, data;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Push onto Stack\n");
```

```
printf("2. Pop from Stack\n");
printf("3. Display Stack\n");
printf("4. Enqueue into Queue\n");
printf("5. Dequeue from Queue\n");
printf("6. Display Queue\n");
printf("7. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter the element to push: ");
        scanf("%d", &data);
        push(&stackTop, data);
        break;
    case 2:
        data = pop(&stackTop);
        if (data != -1) {
            printf("Popped: %d\n", data);
        }
        break;
    case 3:
        printf("Stack: ");
        displayList(stackTop);
        break;
    case 4:
        printf("Enter the element to enqueue: ");
        scanf("%d", &data);
        enqueue(&queueFront, &queueRear, data);
```

```
            break;
        case 5:
            data = dequeue(&queueFront);
            if (data != -1) {
                printf("Dequeued: %d\n", data);
            }
            break;
        case 6:
            printf("Queue: ");
            displayList(queueFront);
            break;
        case 7:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }
   return 0;
}
```

Output -

Menu:

1. Push onto Stack

2. Pop from Stack

3. Display Stack

4. Enqueue into Queue

5. Dequeue from Queue

6. Display Queue

7. Exit

Enter your choice: 1

Enter the element to push: 4

Pushed 4 onto the stack.

Menu:

1. Push onto Stack

2. Pop from Stack

3. Display Stack

4. Enqueue into Queue

5. Dequeue from Queue

6. Display Queue

7. Exit

Enter your choice: 1

Enter the element to push: 5

Pushed 5 onto the stack.

Menu:

1. Push onto Stack

2. Pop from Stack

3. Display Stack

4. Enqueue into Queue

5. Dequeue from Queue

6. Display Queue

7. Exit

Enter your choice: 1

Enter the element to push: 6

Pushed 6 onto the stack.

Menu:

1. Push onto Stack

2. Pop from Stack

3. Display Stack

4. Enqueue into Queue

5. Dequeue from Queue

6. Display Queue

7. Exit

Enter your choice: 3

Stack: 6 -> 5 -> 4 -> NULL

Menu:

1. Push onto Stack

2. Pop from Stack

3. Display Stack

4. Enqueue into Queue

5. Dequeue from Queue

6. Display Queue

7. Exit

Enter your choice: 2

Popped: 6

Menu:

1. Push onto Stack

2. Pop from Stack

3. Display Stack

4. Enqueue into Queue

5. Dequeue from Queue

6. Display Queue

7. Exit

Enter your choice: 3

Stack: 5 -> 4 -> NULL

Menu:

1. Push onto Stack

2. Pop from Stack

3. Display Stack

4. Enqueue into Queue

5. Dequeue from Queue

6. Display Queue

7. Exit

Enter your choice: 4

Enter the element to enqueue: 5

Enqueued 5 into the queue.

Menu:

1. Push onto Stack

2. Pop from Stack

3. Display Stack

4. Enqueue into Queue

5. Dequeue from Queue

6. Display Queue

7. Exit

Enter your choice: 4

Enter the element to enqueue: 8

Enqueued 8 into the queue.

Menu:

1. Push onto Stack

2. Pop from Stack

3. Display Stack

4. Enqueue into Queue

5. Dequeue from Queue

6. Display Queue

7. Exit

Enter your choice: 4

Enter the element to enqueue: 9

Enqueued 9 into the queue.

Menu:

1. Push onto Stack

2. Pop from Stack

3. Display Stack

4. Enqueue into Queue

5. Dequeue from Queue

6. Display Queue

7. Exit

Enter your choice: 6

Queue: 5 -> 8 -> 9 -> NULL

Menu:

1. Push onto Stack

2. Pop from Stack

3. Display Stack

4. Enqueue into Queue

5. Dequeue from Queue

6. Display Queue

7. Exit

Enter your choice:

5

Dequeued: 5

Menu:

1. Push onto Stack

2. Pop from Stack

3. Display Stack

4. Enqueue into Queue

5. Dequeue from Queue

6. Display Queue

7. Exit

Enter your choice: 6

Queue: 8 -> 9 -> NULL

Lab Program – 7

WAP to implement doubly link list with primitive operations create a doubly linked list. Insert a new node to the left of the node. Delete the node based on a specific value. 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));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

void append(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
```

```
    Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}

void insertLeft(Node** head, int target, int data) {
    Node* temp = *head;
    while (temp != NULL && temp->data != target) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Target node not found.\n");
        return;
    }
    Node* newNode = createNode(data);
    newNode->next = temp;
    newNode->prev = temp->prev;
    if (temp->prev != NULL) {
        temp->prev->next = newNode;
    } else {
        *head = newNode;
    }
    temp->prev = newNode;
}

void deleteNode(Node** head, int value) {
```

```
    Node* temp = *head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Node with value %d not found.\n", value);
        return;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    } else {
        *head = temp->next;
    }
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    free(temp);
    printf("Node with value %d deleted.\n", value);
}

void display(Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    Node* temp = head;
    printf("List contents: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
```

```
        temp = temp->next;
    }
    printf("\n");
}

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

    do {
        printf("\n--- Doubly Linked List Operations ---\n");
        printf("1. Append a node\n");
        printf("2. Insert to the left of a node\n");
        printf("3. Delete a node\n");
        printf("4. Display the list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to append: ");
                scanf("%d", &value);
                append(&head, value);
                break;

            case 2:
                printf("Enter the target value: ");
                scanf("%d", &target);
```

```
            printf("Enter value to insert to the left of %d: ", target);
            scanf("%d", &value);
            insertLeft(&head, target, value);
            break;

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

        case 4:
            display(head);
            break;

        case 5:
            printf("Exiting...\n");
            break;

        default:
            printf("Invalid choice! Please try again.\n");
      }
   } while (choice != 5);

   return 0;
}
```

Output -

--- Doubly Linked List Operations ---

1. Append a node

2. Insert to the left of a node

3. Delete a node

4. Display the list

5. Exit

Enter your choice: 1

Enter value to append: 2

--- Doubly Linked List Operations ---

1. Append a node

2. Insert to the left of a node

3. Delete a node

4. Display the list

5. Exit

Enter your choice: 1

Enter value to append: 3

--- Doubly Linked List Operations ---

1. Append a node

2. Insert to the left of a node

3. Delete a node

4. Display the list

5. Exit

Enter your choice: 1

Enter value to append: 4

--- Doubly Linked List Operations ---

1. Append a node

2. Insert to the left of a node

3. Delete a node

4. Display the list

5. Exit

Enter your choice: 4

List contents: 2 3 4

--- Doubly Linked List Operations ---

1. Append a node

2. Insert to the left of a node

3. Delete a node

4. Display the list

5. Exit

Enter your choice: 2

Enter the target value: 2

Enter value to insert to the left of 2: 1

--- Doubly Linked List Operations ---

1. Append a node

2. Insert to the left of a node

3. Delete a node

4. Display the list

5. Exit

Enter your choice: 4

List contents: 1 2 3 4

--- Doubly Linked List Operations ---

1. Append a node

2. Insert to the left of a node

3. Delete a node

4. Display the list

5. Exit

Enter your choice: 3

Enter value of the node to delete: 1

Node with value 1 deleted.

--- Doubly Linked List Operations ---

1. Append a node

2. Insert to the left of a node

3. Delete a node

4. Display the list

5. Exit

Enter your choice: 4

List contents: 2 3 4

Lab Program – 8

WAP

- **to construct a binary search tree,**
- **to traverse the tree using all the methods i.e., in-order, preorder and postorder**
- **to display the elements in the tree**

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

struct Node {
    int data;
    struct Node *left, *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* insertNode(struct Node* 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(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

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

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

```
int main() {
    struct Node* root = NULL;
    int choice, value;

    printf("Binary Search Tree Operations\n");
    printf("1. Insert\n");
    printf("2. In-order Traversal\n");
    printf("3. Pre-order Traversal\n");
    printf("4. Post-order Traversal\n");
    printf("5. Exit\n");

    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insertNode(root, value);
                break;
            case 2:
                printf("In-order Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Pre-order Traversal: ");
                preorderTraversal(root);
```

```
            printf("\n");
            break;
        case 4:
            printf("Post-order Traversal: ");
            postorderTraversal(root);
            printf("\n");
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

Output -

Binary Search Tree Operations

1. Insert

2. In-order Traversal

3. Pre-order Traversal

4. Post-order Traversal

5. Exit

Enter your choice: 1

Enter value to insert: 20

Enter your choice: 1

Enter value to insert: 30

Enter your choice: 1

Enter value to insert: 25

Enter your choice: 1

Enter value to insert: 40

Enter your choice: 2

In-order Traversal: 20 25 30 40

Enter your choice: 3

Pre-order Traversal: 20 30 25 40

Enter your choice: 4

Post-order Traversal: 25 40 30 20

Lab Program – 9

WAP to traverse a graph using BFS method

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

#define MAX 100

struct Queue {
    int items[MAX];
    int front, rear;
};

void initQueue(struct Queue* q) {
    q->front = -1;
    q->rear = -1;
}

int isEmpty(struct Queue* q) {
    return (q->front == -1);
}

void enqueue(struct Queue* q, int value) {
    if (q->rear == MAX - 1) {
        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) {
    int item;
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    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 startVertex, int numVertices) {
    struct Queue q;
    initQueue(&q);

    int visited[MAX] = {0};

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

    enqueue(&q, startVertex);
    visited[startVertex] = 1;
```

```
    while (!isEmpty(&q)) {
        int currentVertex = dequeue(&q);
        printf("%d ", currentVertex);

        for (int i = 0; i < numVertices; i++) {
            if (graph[currentVertex][i] == 1 && !visited[i]) {
                enqueue(&q, i);
                visited[i] = 1;
            }
        }
    }
}

int main() {
    int graph[MAX][MAX], numVertices, startVertex;

    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    printf("Enter the adjacency matrix of the graph (row by row):\n");
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            printf("Enter edge between vertex %d and vertex %d (0 or 1): ", i, j);
            scanf("%d", &graph[i][j]);
        }
    }

    printf("Enter the starting vertex (0 to %d): ", numVertices - 1);
```

```
    scanf("%d", &startVertex);

    BFS(graph, startVertex, numVertices);

    return 0;
}
```

Output -

```
Enter the number of vertices: 4
Enter the adjacency matrix of the graph (row by row):
Enter edge between vertex 0 and vertex 0 (0 or 1): 0
Enter edge between vertex 0 and vertex 1 (0 or 1): 1
Enter edge between vertex 0 and vertex 2 (0 or 1): 0
Enter edge between vertex 0 and vertex 3 (0 or 1): 0
Enter edge between vertex 1 and vertex 0 (0 or 1): 1
Enter edge between vertex 1 and vertex 1 (0 or 1): 0
Enter edge between vertex 1 and vertex 2 (0 or 1): 1
Enter edge between vertex 1 and vertex 3 (0 or 1): 0
Enter edge between vertex 2 and vertex 0 (0 or 1): 0
Enter edge between vertex 2 and vertex 1 (0 or 1): 1
Enter edge between vertex 2 and vertex 2 (0 or 1): 0
Enter edge between vertex 2 and vertex 3 (0 or 1): 1
Enter edge between vertex 3 and vertex 0 (0 or 1): 0
Enter edge between vertex 3 and vertex 1 (0 or 1): 0
Enter edge between vertex 3 and vertex 2 (0 or 1): 1
Enter edge between vertex 3 and vertex 3 (0 or 1): 0
Enter the starting vertex (0 to 3): 1
BFS Traversal starting from vertex 1: 1 0 2 3
```

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

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

#define MAX 100

int adjMatrix[MAX][MAX];
bool visited[MAX];
int stack[MAX];
int top = -1;

void push(int vertex) {
    if (top == MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack[++top] = vertex;
}

int pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        return -1;
    }
    return stack[top--];
}
```

```
void dfsUsingStack(int startVertex, int numVertices) {
    push(startVertex);
    visited[startVertex] = true;

    while (top != -1) {
        int currentVertex = pop();

        for (int i = 0; i < numVertices; i++) {
            if (adjMatrix[currentVertex][i] == 1 && !visited[i]) {
                push(i);
                visited[i] = true;
            }
        }
    }
}

bool isConnected(int numVertices) {
    for (int i = 0; i < numVertices; i++) {
        visited[i] = false;
    }

    dfsUsingStack(0, numVertices);

    for (int i = 0; i < numVertices; i++) {
        if (!visited[i]) {
            return false;
        }
    }
    return true;
```

```
}

int main() {
    int numVertices, numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            adjMatrix[i][j] = 0;
        }
    }

    printf("Enter the edges (start_vertex end_vertex):\n");
    for (int i = 0; i < numEdges; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1;
    }

    if (isConnected(numVertices)) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }
```

```
    return 0;
}
```

Output -

```
Enter the number of vertices: 5
Enter the number of edges: 6
Enter the edges (start_vertex end_vertex):
0
1
0
2
1
2
1
4
2
3
3
4
The graph is connected.
```

Lab Program – 10

Linear Probing

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 <string.h> // For using string functions like strcpy and strlen
#define MAX 100

struct Employee {
    int k;
    char n[50];
};

struct Employee ht[MAX];
int ts;

void init() {
    for (int i = 0; i < MAX; i++) ht[i].k = -1;
}

int hash(int k) {
    return k % ts;
}

void insert(int k, char n[]) {
```

```
    int idx = hash(k);
    int startIdx = idx;
    while (ht[idx].k != -1) {
        idx = (idx + 1) % ts;
        if (idx == startIdx) {
            printf("Hash table is full!\n");
            return;
        }
    }
    ht[idx].k = k;
    strncpy(ht[idx].n, n, 49);
    ht[idx].n[49] = '\0';
}

void display() {
    for (int i = 0; i < ts; i++) {
        if (ht[i].k != -1)
            printf("Idx %d: Key = %d, Name = %s\n", i, ht[i].k, ht[i].n);
        else
            printf("Idx %d: Empty\n", i);
    }
}

int main() {
    int n;
    printf("Enter table size (max size %d): ", MAX);
    scanf("%d", &ts);

    if (ts <= 0 || ts > MAX) ts = MAX;
```

```
    init();

    printf("Enter number of employees: ");
    scanf("%d", &n);
    getchar();  // To consume the newline left by the previous scanf

    for (int i = 0; i < n; i++) {
        int k;
        char name[50];
        printf("Enter key and name for employee %d: ", i + 1);
        scanf("%d", &k);
        getchar();
        fgets(name, 50, stdin);
        name[strcspn(name, "\n")] = '\0';
        insert(k, name);
    }

    display();

    return 0;
}
```

Output –

Enter table size (max size 100): 4

Enter number of employees: 5

Enter key and name for employee 1: 100 Rahul

Enter key and name for employee 2: 1200 Mohith

Enter key and name for employee 3: 200

Raman

Enter key and name for employee 4: 400 Ramesh

Enter key and name for employee 5: 600 Rohith

Hash table is full!

Idx 0: Key = 100, Name = Rahul

Idx 1: Key = 1200, Name = Mohith

Idx 2: Key = 200, Name = Raman

Idx 3: Key = 400, Name = Ramesh

Leet Code

Q.No.283 Move Zeroes

Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements

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


            }
            nums[r]=0;
            r--;
        }

    }
    }
```

Leet Code

Q.No 169 Majority Element

Given an array nums of size n, return the majority element

The majority element is the element that appears more than [n/2] times. You may assume that the majority element exists in the array.

```
int findex(int* number, int value, int length) {
    for (int i = 0; i < length; i++) {
        if (number[i] == value) {
            return i;
        }
    }
    return -1;
}

int majorityElement(int* nums, int numsSize) {
    int ans = numsSize / 2;
    int c = 0;
    int number[numsSize];
    int numcount[numsSize];

    for (int i = 0; i < numsSize; i++) {
        int length = c;
        if (findex(number, nums[i], length) == -1) {
            number[c] = nums[i];
            numcount[c] = 1;
            c++;
        } else {
            int index = findex(number, nums[i], length);
            numcount[index]++;
        }
    }


    for (int i = 0; i < c; i++) {
        if (numcount[i] > ans) {
            return number[i];
        }
    }

    return -1;
}
```

HackerRank Program

Game of Two Stacks

```
#include <stdio.h>

int twoStacks(int maxSum, int* a, int n, int* b, int m) {
    int prefix_a[n + 1];
    prefix_a[0] = 0;
    for (int i = 0; i < n; i++) {
        prefix_a[i + 1] = prefix_a[i] + a[i];
    }

    int prefix_b[m + 1];
    prefix_b[0] = 0;
    for (int j = 0; j < m; j++) {
        prefix_b[j + 1] = prefix_b[j] + b[j];
    }
    int max_count = 0;
    int j = m;

    for (int i = 0; i <= n; i++) {
        if (prefix_a[i] > maxSum) {
            break;
        }
        while (j > 0 && prefix_a[i] + prefix_b[j] > maxSum) {
            j--;
        }
```

```
        max_count = (max_count > i + j) ? max_count : (i + j);
    }

    return max_count;
}
int main() {
    int g;
    scanf("%d", &g);

    while (g--) {
        int n, m, maxSum;
        scanf("%d %d %d", &n, &m, &maxSum);

        int a[n], b[m];



        for (int i = 0; i < n; i++) {
            scanf("%d", &a[i]);
        }



        for (int j = 0; j < m; j++) {
            scanf("%d", &b[j]);
        }
        printf("%d\n", twoStacks(maxSum, a, n, b, m));
    }
    return 0;
}
```

Leet Code

Q.No 234 Palindrome Linked List

```
/**
 * 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 *slow = head;
    struct ListNode *fast = head;

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

    struct ListNode *prev = NULL;
    struct ListNode *curr = slow;
    while (curr != NULL) {
        struct ListNode *next = curr->next;
        curr->next = prev;
```

```
        prev = curr;
        curr = next;
    }

    struct ListNode *first = head;
    struct ListNode *second = prev;
    while (second != NULL) {
        if (first->val != second->val) {
            return false;
        }
        first = first->next;
        second = second->next;
    }

    return true;
}
```

Leet Code

Q.No 112 Path Sum

```
class Solution {
public:
    bool hasPathSum(TreeNode* root, int targetSum) {
        if(root == NULL) {
            return false;
        }
        if(root -> val == targetSum && root -> left == NULL && root -> right == NULL) {
            return true;
        }
        return hasPathSum(root -> left, targetSum - root -> val) || hasPathSum(root -> right, targetSum - root -> val);
    }
};
```