DATA STRUCTURES AND ALGORITHMS LAB MANUAL

I MCA –I SEM(2025-26 AY)

Array-Based Practical Questions

1. Write a program to input elements into a 1D array and display them.
2.
3. Develop a C program to calculate the sum of all elements in an array.
4. Write a C program to find the maximum and minimum element from a given array.
5. Implement linear search to find an element in an array and display its position.
6. Create a program to input and display elements of a 2D array (matrix).
7. Write a program to perform the addition of two matrices.
8. Create a C program to multiply two matrices. Check for valid dimensions.
9. Write a program to find the transpose of a given matrix.

Pointer-Based Practical Questions

9. Write a C program to declare a pointer and access the value and address of a variable.
10. Demonstrate accessing array elements using pointer arithmetic.
11. Create a program to implement and print values using a pointer to a pointer (double pointer).
12. Develop a C program to swap two numbers using call by reference with pointers.
13. Write a program to dynamically allocate memory for an array using malloc() and display elements.
14. Create a program to display each character of a string using a pointer.
15. Write a C program to return a pointer from a function that returns the maximum of two numbers.
16. Write a program to perform and demonstrate pointer arithmetic operations like increment, addition, subtraction.

String Handling Practical Questions

17. Write a C program to find the length of a string using strlen().
18. Write a program to copy one string into another using strcpy().
19. Write a C program to concatenate two strings using strcat().
20. Compare two strings using strcmp() and print whether they are equal or not.
21. Reverse a string without using any library function. Display the original and reversed string.
22. Write a program to find the first occurrence of a character in a string using strchr().
23. Write a C program to find the first occurrence of a substring within a string using strstr().
24. Check if a given string is a palindrome (e.g., MADAM, MOM, etc.). Print the result.
25. Implement a singly linked list with operations: insert at beginning, insert at end, insert at position, delete by value, display, and search. Demonstrate each operation with sample inputs.
26. To implement a Stack using arrays with push, pop, and display operations.
27. To implement a Queue using arrays with enqueue, dequeue, and display operations.
28. To implement a Circular Queue using arrays.
29. To create a Binary Search Tree and perform inorder, preorder, and postorder traversals.
30. To write a program to search for an element in an array using linear search.

31, To write a program to search for an element in a **sorted array** using **iterative binary search**.
32. To write a program to search for an element in a **sorted array** using **recursive binary search**.

Experiment 1: Input and Display Array
Aim:
To write a C program to input elements into a 1D array and display them.
Program:

```
#include <stdio.h>

void main() {
   int arr[100], n;

   printf("Enter number of elements: ");
   scanf("%d", &n);

   printf("Enter %d elements:\n", n);
   for(int i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   printf("Array elements are:\n");
   for(int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }
}
```

Sample Output:
Enter number of elements: 5
Enter 5 elements:
10 20 30 40 50
Array elements are:
10 20 30 40 50

Experiment 2: Sum of Array Elements
Aim:
To write a C program to calculate the sum of all elements in a 1D array.
Program:

```
#include <stdio.h>

void main() {
   int arr[100], n, sum = 0;
```

```
    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements:\n", n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
        sum += arr[i];
    }

    printf("Sum of elements: %d\n", sum);
}
```

Sample Output:

```
Enter number of elements: 4
Enter 4 elements:
5 10 15 20
Sum of elements: 50
```

Experiment 3: Find Maximum and Minimum Element

Aim:

To write a C program to find the maximum and minimum elements in an array.

Program:

```
#include <stdio.h>

void main() {
    int arr[100], n, max, min;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements:\n", n);
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    max = min = arr[0];

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

```
        if(arr[i] > max)
            max = arr[i];
        if(arr[i] < min)
            min = arr[i];
    }

    printf("Maximum: %d\nMinimum: %d\n", max, min);
}
```

Sample Output:

```
Enter number of elements: 5
Enter 5 elements:
12 45 3 22 17
Maximum: 45
Minimum: 3
```

Experiment 4: Linear Search in an Array

Aim:

To write a C program to perform linear search on an array to find a given element.

Program:

```
#include <stdio.h>
void main() {
    int arr[100], n, key, found = 0;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements:\n", n);
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter element to search: ");
    scanf("%d", &key);

    for(int i = 0; i < n; i++) {
        if(arr[i] == key) {
            printf("Element found at position %d\n", i + 1);
            found = 1;
            break;
        }
    }

    if(!found)
        printf("Element not found\n");
}
```

Sample Output:

Enter number of elements: 5
Enter 5 elements:
10 20 30 40 50
Enter element to search: 30
Element found at position 3
Experiment 5: Input and Display a 2D Array
Aim:
To write a C program to input and display a two-dimensional (2D) array.
Program:

```
#include <stdio.h>

void main() {
   int a[10][10], rows, cols;

   printf("Enter rows and columns: ");
   scanf("%d %d", &rows, &cols);

   printf("Enter elements:\n");
   for(int i = 0; i < rows; i++)
      for(int j = 0; j < cols; j++)
         scanf("%d", &a[i][j]);

   printf("Matrix:\n");
   for(int i = 0; i < rows; i++) {
      for(int j = 0; j < cols; j++)
         printf("%d ", a[i][j]);
      printf("\n");
   }
}
```

Sample Output:
Enter rows and columns: 2 3
Enter elements:
1 2 3
4 5 6
Matrix:
1 2 3
4 5 6

Experiment 6: Addition of Two Matrices
Aim:
To write a C program to add two matrices of the same order.
Program:

```
#include <stdio.h>

void main() {
```

```
    int a[10][10], b[10][10], sum[10][10], rows, cols;

    printf("Enter rows and columns: ");
    scanf("%d %d", &rows, &cols);

    printf("Enter elements of first matrix:\n");
    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            scanf("%d", &a[i][j]);

    printf("Enter elements of second matrix:\n");
    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            scanf("%d", &b[i][j]);

    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            sum[i][j] = a[i][j] + b[i][j];

    printf("Sum matrix:\n");
    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < cols; j++)
            printf("%d ", sum[i][j]);
        printf("\n");
    }
}
```

Sample Output:

```
Enter rows and columns: 2 2
Enter elements of first matrix:
1 2
3 4
Enter elements of second matrix:
5 6
7 8
Sum matrix:
6 8
10 12
```

Experiment 7: Multiplication of Two Matrices

Aim:

To write a C program to multiply two matrices if the number of columns of the first matrix equals the number of rows of the second.

Program:

```
#include <stdio.h>

void main() {
   int a[10][10], b[10][10], result[10][10] = {0};
   int r1, c1, r2, c2;

   printf("Enter rows and columns of first matrix: ");
   scanf("%d %d", &r1, &c1);
   printf("Enter rows and columns of second matrix: ");
   scanf("%d %d", &r2, &c2);

   if (c1 != r2) {
      printf("Matrix multiplication not possible.\n");
      return;
   }

   printf("Enter elements of first matrix:\n");
   for(int i = 0; i < r1; i++)
      for(int j = 0; j < c1; j++)
         scanf("%d", &a[i][j]);

   printf("Enter elements of second matrix:\n");
   for(int i = 0; i < r2; i++)
      for(int j = 0; j < c2; j++)
         scanf("%d", &b[i][j]);

   for(int i = 0; i < r1; i++)
      for(int j = 0; j < c2; j++)
         for(int k = 0; k < c1; k++)
            result[i][j] += a[i][k] * b[k][j];

   printf("Resultant matrix:\n");
   for(int i = 0; i < r1; i++) {
```

```
        for(int j = 0; j < c2; j++)
            printf("%d ", result[i][j]);
        printf("\n");
    }
}
```

Sample Output:

```
Enter rows and columns of first matrix: 2 2
Enter rows and columns of second matrix: 2 2
Enter elements of first matrix:
1 2
3 4
Enter elements of second matrix:
5 6
7 8
Resultant matrix:
19 22
43 50
```

Experiment 8: Transpose of a Matrix

Aim:

To write a C program to compute the transpose of a matrix.

Program:

```
#include <stdio.h>
void main() {
    int a[10][10], trans[10][10], rows, cols;

    printf("Enter rows and columns of matrix: ");
```

```
    scanf("%d %d", &rows, &cols);

    printf("Enter matrix elements:\n");
    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            scanf("%d", &a[i][j]);

    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            trans[j][i] = a[i][j];
    printf("Transpose of the matrix:\n");
    for(int i = 0; i < cols; i++) {
        for(int j = 0; j < rows; j++)
            printf("%d ", trans[i][j]);
        printf("\n");
    }
}
```

Output:

```
Enter rows and columns of matrix: 2 3
Enter matrix elements:
1 2 3
4 5 6
Transpose of the matrix:
1 4
2 5
3 6
```

Experiment 9: Basic Pointer Declaration and Access

Aim:

To write a C program to demonstrate pointer declaration, initialization, and accessing values using a pointer.

Program:

```
#include <stdio.h>

void main() {
    int a = 10;
    int *p = &a;

    printf("Value of a: %d\n", a);
    printf("Address of a: %p\n", &a);
    printf("Pointer p points to: %p\n", p);
    printf("Value pointed by p: %d\n", *p);
}
```

Sample Output:

```
Value of a: 10
Address of a: 0x7ffc3b0c1234
Pointer p points to: 0x7ffc3b0c1234
```

Value pointed by p: 10

Experiment 10: Pointer and Array Relationship
Aim:
To write a C program to access array elements using a pointer.
Program:

```
#include <stdio.h>

void main() {
   int arr[5] = {10, 20, 30, 40, 50};
   int *p = arr;

   for(int i = 0; i < 5; i++) {
      printf("arr[%d] = %d, *(p + %d) = %d\n", i, arr[i], i, *(p + i));
   }
}
```

Sample Output:

```
arr[0] = 10, *(p + 0) = 10
arr[1] = 20, *(p + 1) = 20
arr[2] = 30, *(p + 2) = 30
arr[3] = 40, *(p + 3) = 40
arr[4] = 50, *(p + 4) = 50
```

Experiment 11: Pointer to Pointer (Double Pointer)
Aim:
To write a C program to demonstrate the use of a pointer to a pointer (double pointer).
Program:

```
#include <stdio.h>

void main() {
   int x = 100;
   int *p = &x;
   int **pp = &p;

   printf("Value of x: %d\n", x);
   printf("Value using *p: %d\n", *p);
   printf("Value using **pp: %d\n", **pp);
}
```

Sample Output:

```
Value of x: 100
Value using *p: 100
Value using **pp: 100
```

Experiment 12: Swapping Two Numbers Using Pointers

Aim:
To write a C program to swap two numbers using call by reference with pointers.
Program:

```
#include <stdio.h>

void swap(int *x, int *y) {
   int temp = *x;
   *x = *y;
   *y = temp;
}

void main() {
   int a = 5, b = 10;

   printf("Before swapping: a = %d, b = %d\n", a, b);
   swap(&a, &b);
   printf("After swapping: a = %d, b = %d\n", a, b);
}
```

Sample Output:
Before swapping: a = 5, b = 10
After swapping: a = 10, b = 5

Experiment 13: Dynamic Memory Allocation (Using malloc)
Aim:
To write a C program to dynamically allocate memory using malloc() and initialize array elements.
Program:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int *ptr;
   int n;

   printf("Enter number of elements: ");
   scanf("%d", &n);

   ptr = (int*) malloc(n * sizeof(int));

   if (ptr == NULL) {
      printf("Memory not allocated.\n");
      return 1;
```

```
    }

    for (int i = 0; i < n; i++) {
        ptr[i] = i + 1;
    }
    printf("Dynamically allocated array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", ptr[i]);
    }
    free(ptr);
    return 0;
}
```

Sample Output:

```
Enter number of elements: 5
Dynamically allocated array:
1 2 3 4 5
```

Experiment 14: String Manipulation Using Pointers

Aim:

To write a C program to print a string character-by-character using a pointer.

Program:

```
#include <stdio.h>

void main() {
    char str[] = "Hello";
    char *p = str;

    printf("Characters in the string:\n");
    while(*p != '\0') {
        printf("%c ", *p);
        p++;
    }
}
```

Sample Output:

```
Characters in the string:
H e l l o
```

Experiment 15: Function Returning Pointer
Aim:
To write a C program to find the maximum of two numbers using a function that returns a pointer.
Program:

```
#include <stdio.h>

int* getMax(int *x, int *y) {
   return (*x > *y) ? x : y;
}

void main() {
   int a = 20, b = 30;
   int *max = getMax(&a, &b);
   printf("Maximum is: %d\n", *max);
}
```

Sample Output:
Maximum is: 30

Experiment 16: Pointer Arithmetic
Aim:
To write a C program to demonstrate pointer arithmetic (increment, addition, subtraction).
Program:

```
#include <stdio.h>

void main() {
   int arr[] = {5, 10, 15, 20, 25};
   int *ptr = arr;

   printf("Array elements using pointer arithmetic:\n");
   for (int i = 0; i < 5; i++) {
      printf("Value at *(ptr + %d) = %d\n", i, *(ptr + i));
```

```
    }

    printf("\nPointer increment operations:\n");
    ptr = arr;
    printf("Original: %p -> %d\n", ptr, *ptr);
    ptr++;
    printf("After ptr++: %p -> %d\n", ptr, *ptr);
    ptr += 2;
    printf("After ptr += 2: %p -> %d\n", ptr, *ptr);
    ptr--;
    printf("After ptr--: %p -> %d\n", ptr, *ptr);
}
```

Sample Output:

```
Array elements using pointer arithmetic:
Value at *(ptr + 0) = 5
Value at *(ptr + 1) = 10
Value at *(ptr + 2) = 15
Value at *(ptr + 3) = 20
Value at *(ptr + 4) = 25

Pointer increment operations:
Original: 0x7ffd2a3c0f50 -> 5
After ptr++: 0x7ffd2a3c0f54 -> 10
After ptr += 2: 0x7ffd2a3c0f5c -> 20
After ptr--: 0x7ffd2a3c0f58 -> 15
```

Experiment 17: strlen() – Length of a String

Aim:

To write a C program to find the length of a string using strlen().

Program:

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

void main() {
    char str[] = "Hello World";
    printf("Length of string: %lu\n", strlen(str));
}
```

Sample Output:

```
Length of string: 11
```

Experiment 18: strcpy() – Copy One String to Another

Aim:

To write a C program to copy one string to another using strcpy().

Program:

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

```
void main() {
   char src[] = "C Programming";
   char dest[50];

   strcpy(dest, src);
   printf("Copied string: %s\n", dest);
}
```
Sample Output:
Copied string: C Programming

Experiment 19: strcat() – Concatenate Two Strings
Aim:
To write a C program to concatenate two strings using strcat().
Program:
```
#include <stdio.h>
#include <string.h>

void main() {
   char str1[100] = "Hello ";
   char str2[] = "World";

   strcat(str1, str2);
   printf("Concatenated string: %s\n", str1);
}
```
Sample Output:
Concatenated string: Hello World

Experiment 20: strcmp() – Compare Two Strings
Aim:
To write a C program to compare two strings using strcmp().
Program:
```
#include <stdio.h>
#include <string.h>

void main() {
   char str1[] = "Apple";
   char str2[] = "Banana";

   int result = strcmp(str1, str2);

   if (result == 0)
      printf("Strings are equal\n");
   else if (result < 0)
      printf("str1 is less than str2\n");
   else
```

```
        printf("str1 is greater than str2\n");
}
```
Sample Output:
str1 is less than str2

Experiment 21: Reverse a String (Manual Method)
Aim:
To write a C program to reverse a string without using a library function.
Program:
```
#include <stdio.h>
#include <string.h>

void reverse(char *str) {
    int len = strlen(str);
    for(int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = temp;
    }
}

void main() {
    char str[] = "Programming";
    reverse(str);
    printf("Reversed string: %s\n", str);
}
```
Sample Output:
Reversed string: gnimmargorP

Experiment 22: strchr() – First Occurrence of a Character
Aim:
To write a C program to find the first occurrence of a character using strchr().
Program:
```
#include <stdio.h>
#include <string.h>

void main() {
    char str[] = "Hello World";
    char *ptr = strchr(str, 'o');

    if (ptr)
        printf("Character found at position: %ld\n", ptr - str);
    else
```

```
        printf("Character not found\n");
}
```
Sample Output:
Character found at position: 4

Experiment 23: strstr() – First Occurrence of a Substring
Aim:
To write a C program to find the first occurrence of a substring using strstr().
Program:
```
#include <stdio.h>
#include <string.h>

void main() {
    char str[] = "Welcome to C programming";
    char *ptr = strstr(str, "C");

    if (ptr)
        printf("Substring found at position: %ld\n", ptr - str);
    else
        printf("Substring not found\n");
}
```
Sample Output:
Substring found at position: 11

Experiment 24: Palindrome Check for a String
Aim:
To write a C program to check whether a given string is a palindrome.
Program:
```
#include <stdio.h>
#include <string.h>

int isPalindrome(char str[]) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        if (str[i] != str[len - 1 - i])
            return 0;
    }
    return 1;
}

void main() {
    char str[] = "MADAM";

    if (isPalindrome(str))
        printf("%s is a palindrome\n", str);
    else
```

```
        printf("%s is not a palindrome\n", str);
}
```
Sample Output:
MADAM is a palindrome

Experiment 25: Singly Linked List Operations
Aim:
To write a C program to implement a singly linked list with operations:
✅Create list
✅Insert (beginning, end, position)
✅Delete by value
✅Search
✅Display

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

struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

// Insert at beginning
struct Node* insertAtBeginning(struct Node* head, int value) {
    struct Node* newNode = createNode(value);
    newNode->next = head;
    return newNode;
}

// Insert at end
struct Node* insertAtEnd(struct Node* head, int value) {
    struct Node* newNode = createNode(value);
    if (head == NULL)
        return newNode;

    struct Node* temp = head;
    while (temp->next != NULL)
        temp = temp->next;
```

```
    temp->next = newNode;
    return head;
}

// Insert at specific position (1-based)
struct Node* insertAtPosition(struct Node* head, int value, int pos) {
    if (pos == 1)
        return insertAtBeginning(head, value);

    struct Node* newNode = createNode(value);
    struct Node* temp = head;
    for (int i = 1; i < pos - 1 && temp != NULL; i++)
        temp = temp->next;

    if (temp == NULL) {
        printf("Invalid position\n");
        free(newNode);
        return head;
    }

    newNode->next = temp->next;
    temp->next = newNode;
    return head;
}

// Delete a node by value
struct Node* deleteByValue(struct Node* head, int value) {
    struct Node* temp = head;
    struct Node* prev = NULL;

    if (temp != NULL && temp->data == value) {
        head = temp->next;
        free(temp);
        return head;
    }

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

    if (temp == NULL) {
        printf("Value not found\n");
        return head;
    }
```

```
    prev->next = temp->next;
    free(temp);
    return head;
}

// Search for an element
int search(struct Node* head, int key) {
    int pos = 1;
    while (head != NULL) {
        if (head->data == key)
            return pos;
        head = head->next;
        pos++;
    }
    return -1;
}

// Display the list
void traverse(struct Node* head) {
    printf("Linked List: ");
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

// Main function
void main() {
    struct Node* head = NULL;
    int choice, value, position;

    while (1) {
        printf("\n--- Menu ---\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete by Value\n");
        printf("5. Search Element\n");
        printf("6. Display List\n");
        printf("7. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

```
            printf("Enter value: ");
            scanf("%d", &value);
            head = insertAtBeginning(head, value);
            break;
        case 2:
            printf("Enter value: ");
            scanf("%d", &value);
            head = insertAtEnd(head, value);
            break;
        case 3:
            printf("Enter position: ");
            scanf("%d", &position);
            printf("Enter value: ");
            scanf("%d", &value);
            head = insertAtPosition(head, value, position);
            break;
        case 4:
            printf("Enter value to delete: ");
            scanf("%d", &value);
            head = deleteByValue(head, value);
            break;
        case 5:
            printf("Enter value to search: ");
            scanf("%d", &value);
            position = search(head, value);
            if (position != -1)
                printf("Element found at position %d\n", position);
            else
                printf("Element not found\n");
            break;
        case 6:
            traverse(head);
            break;
        case 7:
            exit(0);
        default:
            printf("Invalid choice\n");
    }
  }
}
```

Sample Output:

```
--- Menu ---
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete by Value
```

5. Search Element
6. Display List
7. Exit
Enter choice: 1
Enter value: 10

Enter choice: 2
Enter value: 30

Enter choice: 3
Enter position: 2
Enter value: 20

Enter choice: 6
Linked List: 10 -> 20 -> 30 -> NULL

Enter choice: 5
Enter value to search: 20
Element found at position 2

Enter choice: 4
Enter value to delete: 20

Enter choice: 6
Linked List: 10 -> 30 -> NULL

Experiment 26:
Aim:
To implement a Stack using arrays with push, pop, and display operations.

```
#include <stdio.h>
#define SIZE 100

int stack[SIZE];
int top = -1;

// Push operation
void push(int value) {
    if(top == SIZE - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack[++top] = value;
}

// Pop operation
void pop() {
```

```
    if(top == -1) {
        printf("Stack Underflow\n");
        return;
    }
    printf("Popped: %d\n", stack[top--]);
}

// Display operation
void display() {
    if(top == -1) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack: ");
    for(int i = top; i >= 0; i--)
        printf("%d ", stack[i]);
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\n--- Stack Menu ---\n");
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
```

```
}
```
Output:
```
--- Stack Menu ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 10
Enter your choice: 1
Enter value to push: 20
Enter your choice: 3
Stack: 20 10
Enter your choice: 2
Popped: 20

Enter your choice: 3
Stack: 10
```

27.Aim:
To implement a Queue using arrays with enqueue, dequeue, and display operations.

```
#include <stdio.h>
#define SIZE 100

int queue[SIZE];
int front = -1, rear = -1;

// Enqueue operation
void enqueue(int value) {
    if (rear == SIZE - 1) {
        printf("Queue Overflow\n");
        return;
    }
    if (front == -1)
        front = 0;
    queue[++rear] = value;
}

// Dequeue operation
void dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue Underflow\n");
        return;
    }
    printf("Dequeued: %d\n", queue[front++]);
```

```
}

// Display operation
void display() {
    if (front == -1 || front > rear) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue: ");
    for (int i = front; i <= rear; i++)
        printf("%d ", queue[i]);
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\n--- Queue Menu ---\n");
        printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter your 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:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
}
```

Output:

```
--- Queue Menu ---
1. Enqueue
2. Dequeue
```

3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10

Enter your choice: 1
Enter value to enqueue: 20

Enter your choice: 3
Queue: 10 20

Enter your choice: 2
Dequeued: 10

Enter your choice: 3
Queue: 20

28. Aim:To implement a Circular Queue using arrays.

```
#include <stdio.h>
#define SIZE 5

int queue[SIZE];
int front = -1, rear = -1;

// Enqueue operation
void enqueue(int value) {
    if ((rear + 1) % SIZE == front) {
        printf("Queue Overflow\n");
        return;
    }

    if (front == -1)  // First element
        front = rear = 0;
    else
        rear = (rear + 1) % SIZE;

    queue[rear] = value;
    printf("%d enqueued to circular queue\n", value);
}

// Dequeue operation
void dequeue() {
    if (front == -1) {
        printf("Queue Underflow\n");
        return;
```

```
    }

    printf("Dequeued: %d\n", queue[front]);

    if (front == rear)  // Only one element
        front = rear = -1;
    else
        front = (front + 1) % SIZE;
}

// Display operation
void display() {
    if (front == -1) {
        printf("Queue is empty\n");
        return;
    }

    printf("Circular Queue: ");
    int i = front;
    while (1) {
        printf("%d ", queue[i]);
        if (i == rear)
            break;
        i = (i + 1) % SIZE;
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\n--- Circular Queue Menu ---\n");
        printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter your 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:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice! Try again.\n");
        }
    }
}
```

--- Circular Queue Menu ---

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter value to enqueue: 10

10 enqueued to circular queue

Enter your choice: 1

Enter value to enqueue: 20

20 enqueued to circular queue

Enter your choice: 3

Circular Queue: 10 20

Enter your choice: 2

Dequeued: 10

Enter your choice: 3

Circular Queue: 20

29. To create a Binary Search Tree and perform inorder, preorder, and postorder traversals.

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

struct Node {
    int data;
    struct Node *left, *right;
};

// Function to create a new node
struct Node *createNode(int value) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Insert node into BST
struct Node *insert(struct Node *root, int value) {
    if (root == NULL)
        return createNode(value);

    if (value < root->data)
```

```
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);

    return root;
}

// Inorder Traversal
void inorder(struct Node *root) {
    if (root) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

// Preorder Traversal
void preorder(struct Node *root) {
    if (root) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
```

```
}

// Postorder Traversal
void postorder(struct Node *root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

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

    while (1) {
        printf("\n--- BST Menu ---\n");
        printf("1. Insert\n2. Inorder\n3. Preorder\n4. Postorder\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
```

```
        printf("Enter value to insert: ");
        scanf("%d", &value);
        root = insert(root, value);
        break;
    case 2:
        printf("Inorder Traversal: ");
        inorder(root);
        printf("\n");
        break;
    case 3:
        printf("Preorder Traversal: ");
        preorder(root);
        printf("\n");
        break;
    case 4:
        printf("Postorder Traversal: ");
        postorder(root);
        printf("\n");
        break;
    case 5:
        return 0;
    default:
        printf("Invalid choice! Try again.\n");
```

```
        }

    }

}
```

Output:

--- BST Menu ---

1. Insert

2. Inorder

3. Preorder

4. Postorder

5. Exit

Enter your choice: 1

Enter value to insert: 50

Enter your choice: 1

Enter value to insert: 30

Enter your choice: 2

Inorder Traversal: 30 50

Enter your choice: 3

Preorder Traversal: 50 30

30. Aim:To write a program to search for an element in an array using linear search.

```
#include <stdio.h>

int linearSearch(int arr[], int n, int key) {
    for(int i = 0; i < n; i++) {
```

```
        if(arr[i] == key)
            return i; // Element found
    }
    return -1; // Not found
}

int main() {
    int arr[100], n, key;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements:\n", n);
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter the element to search: ");
    scanf("%d", &key);

    int result = linearSearch(arr, n, key);

    if(result != -1)
        printf("Element found at index %d\n", result);
    else
        printf("Element not found in the array.\n");

    return 0;
}
```

Output:

```
Enter number of elements: 6
Enter 6 elements:
12 45 78 34 89 23
Enter the element to search: 34
Element found at index 3
```

31. To write a program to search for an element in a **sorted array** using **iterative binary search**.

```
#include <stdio.h>

int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1;

    while(low <= high) {
        int mid = (low + high) / 2;
```

```
        if(arr[mid] == key)
            return mid;
        else if(arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }

    return -1; // Not found
}

int main() {
    int arr[100], n, key;

    printf("Enter the number of elements (sorted array): ");
    scanf("%d", &n);

    printf("Enter %d sorted elements:\n", n);
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter the element to search: ");
    scanf("%d", &key);

    int result = binarySearch(arr, n, key);

    if(result != -1)
        printf("Element found at index %d\n", result);
    else
        printf("Element not found in the array.\n");

    return 0;
}
```

Output:

```
Enter the number of elements (sorted array): 5
Enter 5 sorted elements:
11 22 33 44 55
Enter the element to search: 33
Element found at index 2
```

32.To write a program to search for an element in a **sorted array** using **recursive binary search**.

```
#include <stdio.h>

int binarySearchRecursive(int arr[], int low, int high, int key) {
```

```
    if(low > high)
        return -1;

    int mid = (low + high) / 2;

    if(arr[mid] == key)
        return mid;
    else if(arr[mid] > key)
        return binarySearchRecursive(arr, low, mid - 1, key);
    else
        return binarySearchRecursive(arr, mid + 1, high, key);
}

int main() {
    int arr[100], n, key;

    printf("Enter the number of elements (sorted array): ");
    scanf("%d", &n);

    printf("Enter %d sorted elements:\n", n);
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter the element to search: ");
    scanf("%d", &key);

    int result = binarySearchRecursive(arr, 0, n - 1, key);

    if(result != -1)
        printf("Element found at index %d\n", result);
    else
        printf("Element not found in the array.\n");

    return 0;
}
```

Output: Enter the number of elements (sorted array): 6
Enter 6 sorted elements:
5 10 15 20 25 30
Enter the element to search: 20
Element found at index 3