PANIMALR ENGINEERING COLLEGE

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

UNIT- II ARRAYS AND STRINGS QB ANSWERS

1.What is an array? Write the syntax for array. (2 Marks)

An **array** in C is a collection of elements of the same data type, stored in contiguous memory locations. Arrays allow you to store multiple values under a single variable name, and you can access each element using its index.

Syntax for Declaring an Array:

```
data_type array_name[array_size];
```

- **data_type**: The type of data the array will store (e.g., int, float, char, etc.).
- **array_name**: The name you give to the array.
- **array_size**: The number of elements the array will hold. This must be a positive integer.

Example:

```
int numbers[5];  // Declares an array of 5 integers
```

This declares an array named numbers that can store 5 integers. The elements can be accessed using indices like numbers[0], numbers[1], and so on.

2. List out the advantages of Arrays. (2 Marks)

- **Efficient Access:** Elements in an array can be accessed directly using their index, making it a fast and efficient data structure for random access operations.
- **Contiguous Memory Allocation:** Arrays are typically allocated in contiguous memory locations, which can improve memory locality and cache performance.
- **Simple and Intuitive:** Arrays are easy to understand and use, making them a fundamental data structure in many programming languages.
- **Fixed Size:** The size of an array is fixed at creation, which can be beneficial in certain scenarios where you know the exact number of elements you need.
- **Efficient Iterations:** Iterating over all elements in an array is straightforward, using a loop or a dedicated iterator.
- **Support for Mathematical Operations:** Arrays can be used to represent vectors, matrices, and other mathematical structures, enabling efficient mathematical operations.
- **Pass-by-Reference:** In some languages, arrays can be passed by reference to functions, allowing for efficient modification of the original array.

3. What is arithmetic mode, give an example? (2 Marks)

Arithmetic mode, also known as the statistical mode, is the most frequently occurring value in a dataset. It's a measure of central tendency, similar to the mean (average) and median.
Example:
Consider the following dataset:

- 1, 2, 3, 3, 4, 5, 5, 5, 6, 7

In this dataset, the number **5** appears most frequently. Therefore, the arithmetic mode is **5**.

4. What is the role of strrev() function. (2 Marks)

The strrev() function in C is used to reverse a given string. It takes a string as input and returns the same string, but with its characters in reverse order.

The strrev() function is **not part of the C Standard Library**, so it may not be available in all compilers. For example, it's available in some older compilers like Turbo C but not in modern ones like GCC.

```
#include <stdio.h>
#include <string.h>
#include<conio.h>
int main() {
   char str[50] = "Hello, World!";
   clrscr();
   printf("Original String: %s\n", str);

   // Reversing the string
   strrev(str);

   printf("Reversed String: %s\n", str);
   getch();
   return 0;
}
```

Output

```
Original String: Hello, World!
Reversed String: !dlroW ,olleH
```

5. List the various string handling functions in C. (2 Marks)

C provides a number of string handling functions in the string.h library, which help in manipulating and working with strings

1. strlen()

- **Description**: Returns the length of a string (excluding the null terminator \0).

2. strcpy()

- **Description**: Copies a string from source to destination.

3. strcmp()

- **Description**: Compares two strings lexicographically (returns 0 if equal, a negative value if str1 is less than str2, and positive if str1 is greater).

4.strcat()

- **Description**: Appends the source string to the destination string.

6. Distinguish between one dimensional and two dimensional array. (2 Marks)

One-Dimensional Array:

- **Structure:** A linear collection of elements, arranged in a single row.
- **Access:** Elements are accessed using a single index.
- **Example:** `int numbers[5];` // An array of 5 integers

Two-Dimensional Array:

- **Structure:** A collection of elements arranged in rows and columns, forming a grid or table.
- **Access:** Elements are accessed using two indices: one for the row and one for the column.
- **Example:** `int matrix[3][4];` // A 3x4 matrix of integers

Key Differences:

Feature	One-Dimensional Array	Two-Dimensional Array
Structure	Linear	Grid

Indexing	Single index	Double index
Usage	Storing sequential data	Representing tabular data, matrices, images
Examples	Lists, vectors	Matrices, grids, tables

7. Construct the list of operations in an array. (2 Marks)

Arrays are versatile data structures that support a wide range of operations. Here are some of the most common operations performed on arrays:

Operation	**Description**
Traversal	Accessing elements one by one
Insertion	Adding an element at a specific index
Deletion	Removing an element from a specific index
Searching	Finding the index of an element (e.g., linear, binary search)
Updating	Modifying the value of an element
Sorting	Arranging elements in a particular order (ascending/descending)
Merging	Combining two arrays
Splitting	Breaking an array into smaller arrays
Reversing	Changing the order of elements in the array
Copying	Copying elements to another array
Resizing	Changing the size of a dynamically allocated array

8. State any four features of array. (2 Marks)

 Fixed Size: Once an array is created, its size is fixed. This means you cannot dynamically add or remove elements without creating a new array.
 Contiguous Memory Allocation: Elements in an array are stored in contiguous memory locations, which allows for efficient random access.
 Indexing: Elements in an array are accessed using their index, which starts from 0. This provides a direct way to locate and manipulate specific elements.
 Homogeneous Data Type: All elements in an array must be of the same data type (e.g., integers, floating-point numbers, characters).

9. Examine the output of the following Code: (2 Marks)

```
main()
 {
char x;
x = ‘a’;
printf(“%d\n”, x);
 }
```

The output of the code will be:
97
Here's a breakdown of the code:

1. **char x;**: This line declares a variable x of type char, which is used to store a single character.
2. **x = 'a';**: This line assigns the character 'a' to the variable x.
3. **printf("%d\n", x);**: This line prints the value of x to the console. The format specifier %d indicates that the value should be printed as a decimal integer.

Since 'a' is represented by the ASCII code 97, the output will be the integer 97.

10. Distinguish between sorting and searching. (2 Marks)

Feature	Sorting	Searching
Definition	Organizing elements in a specific order	Finding the position of a specific element
Purpose	To arrange data for easier processing	To locate a specific element
Algorithms	Bubble Sort, Quick Sort, Merge Sort	Linear Search, Binary Search
Time Complexity	$O(n \log n)$ to $O(n^2)$ (varies by algorithm)	$O(n)$ (Linear Search) or $O(\log n)$ (Binary Search)
Input	Entire dataset	Only the target element
Output	A sorted version of the dataset	Index or location of the element
Process	Comparison and swapping	Comparison and matching
Use Cases	Database indexing, visualization	Database queries, item lookup

Descriptive Questions (16 Marks)

1. Write a C program to add and subtract two 3 x 3 matrices. (16 Marks)

```
#include <stdio.h>
#include<conio.h>
#define SIZE 3 // Define the size of the matrix

// Function to print a matrix
void printMatrix(int matrix[SIZE][SIZE]) {
   for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE; j++) {
         printf("%d ", matrix[i][j]);
      }
      printf("\n");
   }
}

int main() {
   int matrixA[SIZE][SIZE], matrixB[SIZE][SIZE], sum[SIZE][SIZE], difference[SIZE][SIZE];
clrscr();
   // Input elements for the first matrix
   printf("Enter elements of the first 3x3 matrix:\n");
   for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE; j++) {
         printf("Enter element [%d][%d]: ", i, j);
         scanf("%d", &matrixA[i][j]);
      }
   }

   // Input elements for the second matrix
   printf("Enter elements of the second 3x3 matrix:\n");
```

```c
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            printf("Enter element [%d][%d]: ", i, j);
            scanf("%d", &matrixB[i][j]);
        }
    }

    // Calculate the sum and difference of the matrices
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            sum[i][j] = matrixA[i][j] + matrixB[i][j];       // Addition
            difference[i][j] = matrixA[i][j] - matrixB[i][j]; // Subtraction
        }
    }

    // Print the results
    printf("\nSum of the two matrices:\n");
    printMatrix(sum);

    printf("\nDifference of the two matrices:\n");
    printMatrix(difference);
   getch();
    return 0;
}
```
Output
```
Enter elements of the first 3x3 matrix:
Enter element [0][0]: 1
Enter element [0][1]: 2
Enter element [0][2]: 3
Enter element [1][0]: 4
Enter element [1][1]: 5
Enter element [1][2]: 6
Enter element [2][0]: 7
Enter element [2][1]: 8
Enter element [2][2]: 9

Enter elements of the second 3x3 matrix:
Enter element [0][0]: 9
Enter element [0][1]: 8
Enter element [0][2]: 7
Enter element [1][0]: 6
Enter element [1][1]: 5
Enter element [1][2]: 4
Enter element [2][0]: 3
Enter element [2][1]: 2
Enter element [2][2]: 1

Sum of the two matrices:
10 10 10
10 10 10
10 10 10

Difference of the two matrices:
-8 -6 -4
-2 0 2
4 6 8
```
Explanation:

1. **Define Matrix Size**: The SIZE macro is defined to indicate that the matrices are 3x3.
2. **Input Matrices**: The program prompts the user to enter the elements of two 3x3 matrices.
3. **Calculate Sum and Difference**:
    - It uses nested loops to calculate the sum and difference of the matrices element-wise.
4. **Print the Result**: The results (sum and difference) are printed using a helper function printMatrix().

2. Write a C program to find the number of vowels, consonants, digits and white spaces in a string. (16 Marks)

```
#include <stdio.h>
#include <ctype.h> // For character classification functions
#include<conio.h>
int main() {
   char str[100]; // Declare a string of max length 99 + null terminator
   int vowels = 0, consonants = 0, digits = 0, whitespaces = 0; // Initialize counters
  clrscr();
   // Input the string from the user
   printf("Enter a string: ");
   fgets(str, sizeof(str), stdin); // Use fgets to read the string including spaces

   // Traverse the string
   for (int i = 0; str[i] != '\0'; i++) {
      char ch = tolower(str[i]); // Convert to lowercase for uniformity

      // Check for vowels
      if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
         vowels++;
      }
      // Check for consonants
      else if (isalpha(ch)) { // Check if it's an alphabetic character
         consonants++;
      }
      // Check for digits
      else if (isdigit(ch) ) {
         digits++;
      }
      // Check for whitespace
      else if (isspace(ch)) {
         whitespaces++;
      }
   }

   // Print the results
   printf("Number of Vowels: %d\n", vowels);
   printf("Number of Consonants: %d\n", consonants);
   printf("Number of Digits: %d\n", digits);
   printf("Number of Whitespaces: %d\n", whitespaces);
  getch();
   return 0;
}
```

output

```
Enter a string: Hello World! 123
Number of Vowels: 3
Number of Consonants: 7
Number of Digits: 3
Number of Whitespaces: 2
```

Explanation:

1. **Input String**: The program uses fgets() to read a string from the user, which allows spaces in the input.
2. **Character Classification**: It uses the ctype.h library functions:
   - tolower(): Converts characters to lowercase to simplify vowel checking.
   - isalpha(): Checks if a character is an alphabetic letter.
   - isdigit(): Checks if a character is a digit.
   - isspace(): Checks if a character is a whitespace character.
3. **Counters**: Four counters are initialized to keep track of vowels, consonants, digits, and whitespaces.
4. **Loop Through String**: The program traverses the string character by character, updating the respective counters based on the character type.
5. **Output**: Finally, the program prints the counts of vowels, consonants, digits, and whitespaces.

3. Write a C program for matrix multiplication. (16 Marks)

C program that performs matrix multiplication for two matrices. The program prompts the user to input the dimensions and elements of the matrices, performs the multiplication, and then displays the resulting matrix.

```c
#include <stdio.h>
#include<conio.h>
#define MAX 10 // Define maximum size for matrices

// Function to print a matrix
void printMatrix(int matrix[MAX][MAX], int rows, int cols) {
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
      printf("%d ", matrix[i][j]);
    }
    printf("\n");
  }
}

int main() {
  int matrixA[MAX][MAX], matrixB[MAX][MAX], result[MAX][MAX];
  int rowA, colA, rowB, colB;
 clrscr()
  // Input dimensions for the first matrix
  printf("Enter rows and columns for the first matrix: ");
  scanf("%d %d", &rowA, &colA);

  // Input elements for the first matrix
  printf("Enter elements of the first matrix:\n");
  for (int i = 0; i < rowA; i++) {
    for (int j = 0; j < colA; j++) {
      printf("Element [%d][%d]: ", i, j);
      scanf("%d", &matrixA[i][j]);
    }
  }
```

```
    // Input dimensions for the second matrix
    printf("Enter rows and columns for the second matrix: ");
    scanf("%d %d", &rowB, &colB);

    // Check if multiplication is possible
    if (colA != rowB) {
        printf("Matrix multiplication not possible. Columns of first matrix must match rows of second
matrix.\n");
        return 1; // Exit the program
    }

    // Input elements for the second matrix
    printf("Enter elements of the second matrix:\n");
    for (int i = 0; i < rowB; i++) {
        for (int j = 0; j < colB; j++) {
            printf("Element [%d][%d]: ", i, j);
            scanf("%d", &matrixB[i][j]);
        }
    }

    // Initialize the result matrix
    for (int i = 0; i < rowA; i++) {
        for (int j = 0; j < colB; j++) {
            result[i][j] = 0; // Initialize to 0
        }
    }

    // Perform matrix multiplication
    for (int i = 0; i < rowA; i++) {
        for (int j = 0; j < colB; j++) {
            for (int k = 0; k < colA; k++) {
                result[i][j] += matrixA[i][k] * matrixB[k][j];
            }
        }
    }

    // Print the result
    printf("\nResultant Matrix after multiplication:\n");
    printMatrix(result, rowA, colB);
   getch();
    return 0;
}
```
output

```
Enter rows and columns for the first matrix: 2 3
Enter elements of the first matrix:
Element [0][0]: 1
Element [0][1]: 2
Element [0][2]: 3
Element [1][0]: 4
Element [1][1]: 5
Element [1][2]: 6
Enter rows and columns for the second matrix: 3 2
Enter elements of the second matrix:
Element [0][0]: 7
Element [0][1]: 8
Element [1][0]: 9
Element [1][1]: 10
Element [2][0]: 11
Element [2][1]: 12

Resultant Matrix after multiplication:
58 64
139 154
```
Explanation:

1. **Matrix Size**: The program defines a maximum size for the matrices (MAX) to ensure they don't exceed a certain limit.
2. **Input for First Matrix**: The user is prompted to enter the dimensions and elements of the first matrix.
3. **Input for Second Matrix**: The user is prompted to enter the dimensions and elements of the second matrix.
4. **Dimension Check**: The program checks if matrix multiplication is possible. The number of columns in the first matrix must match the number of rows in the second matrix. If not, it prints an error message and exits.
5. **Matrix Multiplication**: The multiplication is performed using three nested loops. The outer two loops iterate over the rows of the first matrix and the columns of the second matrix, while the innermost loop calculates the sum of products.
6. **Output**: Finally, the resulting matrix is printed using a helper function printMatrix().

4. Illustrate the concept of mean, median with an example program. (16 Marks)

Mean:

- The **mean** (average) is calculated by summing all the numbers in a dataset and then dividing by the count of those numbers.

Median:

- The **median** is the middle value of a dataset when it is sorted in ascending or descending order.
  - If the number of observations is odd, the median is the middle element.
  - If the number of observations is even, the median is the average of the two middle elements.

```c
#include <stdio.h>
#include <stdlib.h> // For the qsort function
#include<conio.h>
// Function to compare two integers (for qsort)
int compare(const void *a, const void *b) {
  return (*(int*)a - *(int*)b;
}

// Function to calculate mean
float calculateMean(int arr[], int n) {
  int sum = 0;
  for (int i = 0; i < n; i++) {
    sum += arr[i];
  }
  return (float)sum / n; // Return mean as a float
}

// Function to calculate median
float calculateMedian(int arr[], int n) {
  qsort(arr, n, sizeof(int), compare); // Sort the array
  if (n % 2 == 0) {
    // If even, return average of two middle elements
    return (arr[n/2 - 1] + arr[n/2]) / 2.0;
  } else {
    // If odd, return the middle element
    return arr[n/2];
  }
}

int main() {
  int n;
 clrscr();
  // Input the number of elements
  printf("Enter the number of elements: ");
  scanf("%d", &n);

  int arr[n]; // Declare an array of size n

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

  // Calculate mean and median
  float mean = calculateMean(arr, n);
  float median = calculateMedian(arr, n);

  // Output the results
  printf("\nMean: %.2f\n", mean);
  printf("Median: %.2f\n", median);
  getch();
  return 0;
}
```

Output

```
Enter the number of elements: 5
Enter the elements:
Element [0]: 12
Element [1]: 4
Element [2]: 5
Element [3]: 3
Element [4]: 8

Mean: 6.40
Median: 5.00
```

Explanation of the Program:

1. **Include Libraries**: The program includes stdio.h for standard input/output and stdlib.h for using the qsort() function to sort the array.
2. **Compare Function**: The compare function is defined to be used with qsort() for sorting the integers in ascending order.
3. **Mean Calculation**: The calculateMean function sums all the elements in the array and divides by the number of elements to compute the mean.
4. **Median Calculation**:
   - The calculateMedian function first sorts the array.
   - It then checks if the number of elements is even or odd to determine how to calculate the median.
5. **Main Function**:
   - It prompts the user to enter the number of elements and the elements themselves.
   - It then calls the functions to calculate and display the mean and median.

5. Classify various string handling functions with an example program. (16 Marks)

In C, string handling functions are provided in the <string.h> library. These functions are classified into several categories based on their functionality. Below is a classification of various string handling functions along with a sample program demonstrating their usage.

Classification of String Handling Functions:

1. **String Length Functions**:
   - strlen(): Returns the length of a string.
2. **String Copy Functions**:
   - strcpy(): Copies one string to another.
   - strncpy(): Copies a specified number of characters from one string to another.
3. **String Concatenation Functions**:
   - strcat(): Concatenates (appends) one string to another.
   - strncat(): Appends a specified number of characters from one string to another.
4. **String Comparison Functions**:
   - strcmp(): Compares two strings.
   - strncmp(): Compares a specified number of characters of two strings.
5. **String Search Functions**:
   - strchr(): Searches for the first occurrence of a character in a string.
   - strstr(): Searches for the first occurrence of a substring in a string.
6. **String Modification Functions**:
   - strrev(): Reverses a string.
   - strupr(): Converts a string to uppercase.
   - strlwr(): Converts a string to lowercase.

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include<conio.h>
int main() {
   char str1[100], str2[100], str3[100];
   int length;
  clrscr();
   // Input string
   printf("Enter the first string: ");
   fgets(str1, sizeof(str1), stdin);
   str1[strcspn(str1, "\n")] = 0; // Remove newline character if present

   // String length
   length = strlen(str1);
   printf("Length of the first string: %d\n", length);

   // Copy string
   strcpy(str2, str1);
   printf("Copied string: %s\n", str2);

   // Concatenate strings
   printf("Enter the second string: ");
   fgets(str3, sizeof(str3), stdin);
   str3[strcspn(str3, "\n")] = 0; // Remove newline character if present

   strcat(str1, str3);
   printf("Concatenated string: %s\n", str1);
```

```
    // Compare strings
    int cmpResult = strcmp(str1, str2);
    if (cmpResult == 0) {
        printf("The strings are equal.\n");
    } else {
        printf("The strings are not equal.\n");
    }

    // Search for a character
    char ch;
    printf("Enter a character to search in the concatenated string: ");
    scanf(" %c", &ch);
    char *ptr = strchr(str1, ch);
    if (ptr != NULL) {
        printf("Character '%c' found at position: %ld\n", ch, ptr - str1 + 1);
    } else {
        printf("Character '%c' not found.\n", ch);
    }

    // Convert to uppercase
    for (int i = 0; str1[i]; i++) {
        str1[i] = toupper(str1[i]);
    }
    printf("String in uppercase: %s\n", str1);

    // Reverse the string (using a custom function)
    int len = strlen(str1);
    for (int i = 0; i < len / 2; i++) {
        char temp = str1[i];
        str1[i] = str1[len - i - 1];
        str1[len - i - 1] = temp;
    }
    printf("Reversed string: %s\n", str1);
   getch();
    return 0;
}
```
output

```
Enter the first string: Hello
Length of the first string: 5
Copied string: Hello
Enter the second string: World
Concatenated string: HelloWorld
The strings are not equal.
Enter a character to search in the concatenated string: o
Character 'o' found at position: 5
String in uppercase: HELLOWORLD
Reversed string: DLROWOLLEH
```

6. Implement the concept of selection sort with an example program. (16 Marks)

Selection Sort is a simple and intuitive sorting algorithm. It works by dividing the input list into two parts: a sorted part and an unsorted part. The algorithm repeatedly selects the smallest (or largest, depending on the order) element from the unsorted part and moves it to the end of the sorted part.

Algorithm:

1. Start with the first element as the minimum.

2. Compare it with the rest of the elements in the array.
3. If a smaller element is found, update the minimum.
4. After completing the comparisons, swap the found minimum with the first element.
5. Move to the next position and repeat the process for the unsorted part of the array.
6. Continue until the entire array is sorted.

```
#include <stdio.h>
#include<conio.h>
// Function to perform selection sort
void selectionSort(int arr[], int n) {
  for (int i = 0; i < n - 1; i++) {
    // Assume the minimum is the first element of the unsorted part
    int minIndex = i;

    // Find the minimum element in the unsorted array
    for (int j = i + 1; j < n; j++) {
      if (arr[j] < arr[minIndex]) {
        minIndex = j; // Update the index of the minimum element
      }
    }

    // Swap the found minimum element with the first element
    if (minIndex != i) {
      int temp = arr[i];
      arr[i] = arr[minIndex];
      arr[minIndex] = temp;
    }
  }
}

// Function to print the array
void printArray(int arr[], int size) {
  for (int i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
}

int main() {
  int n;
  clrscr();
  // Input the number of elements
  printf("Enter the number of elements: ");
  scanf("%d", &n);

  int arr[n]; // Declare an array of size n

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

  printf("Original array: ");
  printArray(arr, n);

  // Perform selection sort
```

```
    selectionSort(arr, n);

    printf("Sorted array: ");
    printArray(arr, n);
    getch();
    return 0;
}
```
output

```
Enter the number of elements: 5
Enter the elements:
Element [0]: 64
Element [1]: 25
Element [2]: 12
Element [3]: 22
Element [4]: 11
Original array: 64 25 12 22 11
Sorted array: 11 12 22 25 64
```
Explanation of the Program:

1. **Selection Sort Function**:
   - The selectionSort function iterates through the array and finds the minimum element in the unsorted part.
   - Once the minimum is found, it swaps it with the first element of the unsorted part.
2. **Print Function**:
   - The printArray function displays the elements of the array.
3. **Main Function**:
   - It prompts the user for the number of elements and their values.
   - It calls the selectionSort function to sort the array and then prints the sorted array.

7. Apply the insertion sort technique to sort the given numbers 56,1,7,9,7,14 with an example program. (16 Marks)

Insertion Sort is a simple sorting algorithm that works by repeatedly moving one element at a time to its correct position in the sorted part of the array. It's efficient for small datasets or partially sorted arrays.

```
#include <stdio.h>
#include<conio.h>
// Function to perform insertion sort
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i]; // The element to be inserted
        int j = i - 1;

        // Move elements of arr[0..i-1], that are greater than key,
        // to one position ahead of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key; // Insert the key at the correct position
    }
}

// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {56, 1, 7, 9, 7, 14}; // Given numbers
    clrscr();
    int n = sizeof(arr) / sizeof(arr[0]); // Calculate the number of elements

    printf("Original array: ");
    printArray(arr, n);

    // Perform insertion sort
    insertionSort(arr, n);

    printf("Sorted array: ");
    printArray(arr, n);
    getch();
    return 0;
}
```

Output

```
Original array: 56 1 7 9 7 14
Sorted array: 1 7 7 9 14 56
```

Explanation of the Program:

1. **Insertion Sort Function**:
   - The insertionSort function iterates through the array, starting from the second element.
   - It compares the current element (key) with the elements in the sorted part of the array and shifts larger elements one position to the right.
   - The key is then placed at its correct position.
2. **Print Function**:
   - The printArray function displays the elements of the array.
3. **Main Function**:
   - The program initializes the array with the given numbers (56, 1, 7, 9, 7, 14).
   - It calculates the number of elements and calls the insertionSort function to sort the array.
   - It then prints the sorted array.

8. Construct a c program to find an element using binary search. (16 Marks)

Binary Search is an efficient algorithm for finding an item from a sorted list of items. It works by repeatedly dividing the search interval in half. If the value of the search key is less than the item in the middle of the interval, the search continues in the lower half, or if it is greater, it continues in the upper half. This process repeats until the value is found or the interval is empty.

Algorithm:

1. Start with two pointers: low and high, representing the bounds of the search.
2. Calculate the middle index.
3. If the middle element is equal to the target value, return the index.
4. If the target value is less than the middle element, narrow the search to the lower half.
5. If the target value is greater, narrow the search to the upper half.
6. Repeat until the element is found or the search space is exhausted.

```
#include <stdio.h>
#include<conio.h>
// Function for binary search
int binarySearch(int arr[], int size, int target) {
    int low = 0;
    int high = size - 1;
```

```c
    while (low <= high) {
        int mid = low + (high - low) / 2; // Calculate mid index

        // Check if the target is present at mid
        if (arr[mid] == target) {
            return mid; // Target found
        }
        // If target is greater, ignore left half
        else if (arr[mid] < target) {
            low = mid + 1;
        }
        // If target is smaller, ignore right half
        else {
            high = mid - 1;
        }
    }

    return -1; // Target not found
}

// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19}; // Sorted array
    int size = sizeof(arr) / sizeof(arr[0]); // Calculate number of elements
    int target;
    clrscr();
    // Input target value
    printf("Enter the element to search: ");
    scanf("%d", &target);

    // Perform binary search
    int result = binarySearch(arr, size, target);

    // Output the result
    if (result != -1) {
        printf("Element %d found at index %d.\n", target, result);
    } else {
        printf("Element %d not found in the array.\n", target);
    }
    getch();
    return 0;
}
```

Output

```
Enter the element to search: 9
Element 9 found at index 4.
```

Explanation of the Program:

1. **Binary Search Function**:
   - The binarySearch function takes an array, its size, and the target element as input.

    - It initializes the low and high pointers and enters a loop that continues until the search space is exhausted.
    - It calculates the middle index and compares the middle element with the target value to decide which half of the array to continue searching.
2. **Print Function**:
    - The printArray function displays the elements of the array (not directly used in this example, but useful for debugging).
3. **Main Function**:
    - The program initializes a sorted array of integers.
    - It prompts the user to input the target element to search for.
    - It calls the binarySearch function and outputs the result.

9. construct the concept of Linear Search with an example program. (16 Marks)

Linear Search is a straightforward search algorithm that checks each element in a list sequentially until the desired element is found or the list ends. This method works on both sorted and unsorted arrays.

Algorithm:

1. Start from the first element and compare it with the target value.
2. If the current element matches the target, return its index.
3. If it does not match, move to the next element.
4. Repeat steps 2 and 3 until you find the target or reach the end of the list.
5. If the target is not found after checking all elements, return an indication that the element is absent (e.g., -1).

```
#include <stdio.h>
#include<conio.h>
// Function for linear search
int linearSearch(int arr[], int size, int target) {
   for (int i = 0; i < size; i++) {
      // Check if the current element matches the target
      if (arr[i] == target) {
         return i; // Target found
      }
   }
   return -1; // Target not found
}

// Function to print the array
void printArray(int arr[], int size) {
   for (int i = 0; i < size; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
}

int main() {
   int arr[] = {56, 1, 7, 9, 7, 14}; // Given numbers
   int size = sizeof(arr) / sizeof(arr[0]); // Calculate number of elements
   int target;
   clrscr();
   // Input target value
   printf("Original array: ");
   printArray(arr, size);
```

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

    // Perform linear search
    int result = linearSearch(arr, size, target);

    // Output the result
    if (result != -1) {
        printf("Element %d found at index %d.\n", target, result);
    } else {
        printf("Element %d not found in the array.\n", target);
    }
    getch();
    return 0;
}
```

Output

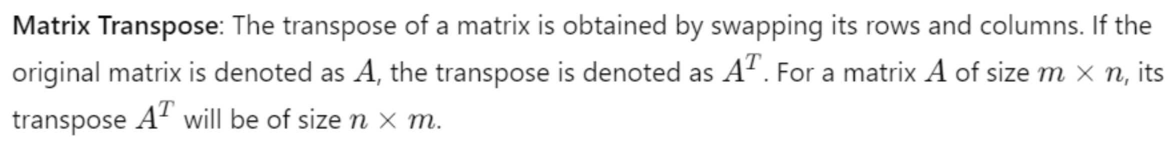
```
Original array: 56 1 7 9 7 14
Enter the element to search: 7
Element 7 found at index 2.
```

Explanation of the Program:

1. **Linear Search Function**:
   - The linearSearch function takes an array, its size, and the target element as inputs.
   - It iterates through the array and checks each element against the target.
   - If a match is found, it returns the index; if no match is found after checking all elements, it returns -1.
2. **Print Function**:
   - The printArray function displays the elements of the array, which is useful for understanding the current state of the data.
3. **Main Function**:
   - The program initializes an array of integers.
   - It prints the original array and prompts the user to input the target value to search for.
   - It calls the linearSearch function and outputs the result.

10. Analyze the concept of matrix determinant and transpose with an example program. (16 Marks)

11. **Matrix Determinant**: The determinant is a scalar value that can be computed from the elements of a square matrix. It provides important information about the matrix, such as whether it is invertible (a non-zero determinant indicates the matrix is invertible) and the scaling factor for transformations represented by the matrix.



Matrix Transpose: The transpose of a matrix is obtained by swapping its rows and columns. If the original matrix is denoted as A, the transpose is denoted as A^T. For a matrix A of size $m \times n$, its transpose A^T will be of size $n \times m$.

Program

```
#include <stdio.h>
#include<conio.h>
// Function to calculate the determinant of a 2x2 matrix
float determinant(float matrix[2][2]) {
    return (matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]);
}

// Function to calculate the transpose of a matrix
void transpose(float matrix[2][2], float result[2][2]) {
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 2; j++) {
         result[j][i] = matrix[i][j]; // Swap rows and columns
      }
   }
}

// Function to print a matrix
void printMatrix(float matrix[2][2]) {
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 2; j++) {
         printf("%.2f ", matrix[i][j]);
      }
      printf("\n");
   }
}

int main() {
   float matrix[2][2] = { {4, 2}, {3, 1} }; // Sample 2x2 matrix
   float det, transposed[2][2];
  clrscr();
   // Calculate the determinant
   det = determinant(matrix);
   printf("Matrix:\n");
   printMatrix(matrix);
   printf("Determinant: %.2f\n", det);

   // Calculate the transpose
   transpose(matrix, transposed);
   printf("Transpose:\n");
   printMatrix(transposed);
   getch();
   return 0;
}
```
Output
```
Matrix:
4.00 2.00
3.00 1.00
Determinant: 2.00
Transpose:
4.00 3.00
2.00 1.00
```
Explanation of the Program

1. **Determinant Function**:
   - The determinant function calculates the determinant of a 2x2 matrix using the formula: det(A)=a11×a22−a12×a21\text{det}(A) = a_{11} \times a_{22} - a_{12} \times a_{21}det(A)=a11×a22−a12×a21
2. **Transpose Function**:
   - The transpose function creates the transpose of a given matrix by swapping the rows and columns.
3. **Print Function**:
   - The printMatrix function prints the elements of a 2x2 matrix in a formatted manner.
4. **Main Function**:

- The program initializes a sample 2x2 matrix.
- It computes the determinant and prints it.
- It computes the transpose of the matrix and prints the transposed matrix.