C Program (Short Questions Ans)

1. (a) Discuss about the following operators in C language with example.

a. Bitwise operators

b. Increment and decrement operators

c. Logical operators

(b) Write a program to perform swapping of two numbers without using temporary variable.

Ans:- **(a) Operators in C Language:**

a. Bitwise Operators: Bitwise operators perform operations at the bit level, manipulating individual bits of binary numbers.

- **Bitwise AND (&):** Performs a bitwise AND operation.

```
int a = 5;    // Binary: 0101
int b = 3;    // Binary: 0011
int result = a & b;  // Binary: 0001
```

- **Bitwise OR (|):** Performs a bitwise OR operation.

```
int a = 5;    // Binary: 0101
int b = 3;    // Binary: 0011
int result = a | b;  // Binary: 0111
```

- **Bitwise XOR (^):** Performs a bitwise XOR (exclusive OR) operation.

```
int a = 5;    // Binary: 0101
int b = 3;    // Binary: 0011
int result = a ^ b;  // Binary: 0110
```

- **Bitwise NOT(~):** Performs a bitwise NOT operation (complement).

```
int a = 5;    // Binary: 0101
int result = ~a;   // Binary: 1010 (2's complement)
```

b. Increment and Decrement Operators: Increment (++) and decrement (--) operators are used to increase or decrease the value of a variable by 1.

- **Increment (++):**

```
int x = 10;
x++;  // x becomes 11
```

- **Decrement (--):**

```
int y = 8;
y--;  // y becomes 7
```

c. Logical Operators: Logical operators perform logical operations on Boolean values (0 or 1).

- **Logical AND (&&):**

```
int p = 1;
int q = 0;
int result = p && q;  // result is 0 (false)
```

- **Logical OR (||):**

```
int p = 1;
int q = 0;
```

```
int result = p || q;  // result is 1 (true)
```

- **Logical NOT (!):**

```
int r = 1;
int result = !r;  // result is 0 (false)
```

(b) Swapping of Two Numbers without Using a Temporary Variable:

Source code:-

```
#include <stdio.h>
int main()
 {
   int num1, num2;
   // Input two numbers
   printf("Enter two numbers:");
   scanf("%d %d", &num1, &num2);
   // Swapping without a temporary variable
   num1 = num1 + num2;
   num2 = num1 - num2;
   num1 = num1 - num2;
   // Output the swapped numbers
   printf("After swapping:\n");
   printf("Number 1: %d\n", num1);
   printf("Number 2: %d\n", num2);
   return 0;
 }
```

Output:-

```
Enter two numbers: 4 5
After swapping:
Number 1: 5
Number 2: 4
```

2. (a) Define algorithm. Write algorithm for finding factorial of a number. (b) What is flowchart? Explain different symbols used for flowchart. (c) Explain about type conversion in C.

Ans:-

(a) Algorithm:

- **Definition:** An algorithm is a step-by-step procedure or a set of well-defined rules for solving a specific problem or accomplishing a particular task.
- Factorial Algorithm:

Step 1: Start

Step 2: Read a number n

Step 3: Initialize variables:

i = 1, fact = 1

Step 4: if i <= n go to step 4 otherwise go to step 7

Step 5: Calculate

fact = fact * i

Step 6: Increment the i by 1 (i=i+1) and go to step 3

Step 7: Print fact

Step 8: Stop

(b) Flowchart:

- **Definition:** A flowchart is a graphical representation of a process, showing the steps involved and the sequence in which they are executed. It is a widely used tool for designing, documenting, and analyzing algorithms or processes.
- **Symbols used in a flowchart:**
  - Flowline: The flowline shows the process's direction by connecting two blocks with one another.
  - Termina or Terminator: The terminal or terminator represents the start or end points of a flowchart process.
  - Process: The process symbol is the most common component of a flowchart and indicates a step in the process.
  - Comment or Annotation: You can indicate additional information about a step with a comment or annotation.
  - Decision: This symbol represents a decision you or your team need to make to get to the next step of the process. Typically, it's a true or false decision or a yes or no question that you need to answer.
  - Stored data: This symbolizes a data file or database.
  - "Or" symbol: This indicates that the process flow continues in three or more branches.
  - Input/Output: The input/output symbol represents the process of in- or outputting external data.
  - Display: This indicates a step that displays relevant information.
  - Document: This symbol represents a single document.
  - Delay: This symbol allows you to plan and represent any delay periods that will be part of the process.
  - Manual input: This symbol represents data or information that needs to be manually entered into a system.
  - Manual operation: This symbolizes a manual operation or adjustment to the process.
  - Off-page connector: This symbol is used to connect two symbols that are of different pages.
  - On-page connector: This dot can connect two symbols and replace long lines which allows for a cleaner flowchart.
  - Summoning junction symbol: This symbol is used to converge multiple branches back into a single process.

- Alternate process: The lines to this symbol are usually dotted. The symbol itself stands for an alternative to the normal process step in case one is needed.
- Predefined process: This symbol indicates a process that is already defined elsewhere.
- Multiple documents: This symbolizes multiple documents.
- Preparation or initialization: This symbol indicates a preparation or initialization step in the process.

(c) In C programming, type conversion refers to the process of converting a value from one data type to another. This is necessary when you want to perform operations or assignments involving variables of different types. Type conversion can be broadly categorized into two types: implicit and explicit.

Implicit Type Conversion (Type Coercion):

- Implicit type conversion is performed automatically by the compiler.
- It occurs when the data types involved in an operation are compatible, and the compiler automatically converts one type to another.
- This conversion is also known as type coercion.
- It is generally safe, as it is done by the compiler without any explicit instructions from the programmer.

```
int a = 5;
float b = 2.5;
// Implicit type conversion
float result = a + b;  // int 'a' is implicitly converted to float
```

Explicit Type Conversion (Type Casting):

- Explicit type conversion is performed by the programmer using casting operators.
- It involves a manual conversion from one data type to another.
- This type of conversion is also known as type casting.
- It should be used with caution, as it may result in loss of data or unexpected behavior if not done carefully.

3. (a) Explain the different types of loops in C with syntax.
(b) Develop a C program to generate and plot the Pascal triangle.

Ans:- **(a) Different Types of Loops in C with Syntax:**

a. **While loop:-** A while loop is the most straightforward looping structure.

Syntax:-

```
while (condition)
{
  // Code to be executed
}
```

It is an entry-controlled loop. In while loop, a condition is evaluated before processing a body of the loop. If a condition is true then and only then the body of a loop is executed. After the body of a loop is executed then control again goes back at the beginning, and the condition is

checked if it is true, the same process is executed until the condition becomes false. Once the condition becomes false, the control goes out of the loop.After exiting the loop, the control goes to the statements which are immediately after the loop.

Example:-

```
#include<stdio.h>
int main()
{
      int num=1;
      while(num<=10)
      {
                printf("%d\n",num);
                num++;
      }
      return 0;
}
```

b. **Do-While loop:-** A do while loop in C is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.

Syntax:-

```
do
{
   // Code to be executed
} while (condition);
```

In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

Example:-

```
#include<stdio.h>
int main()
{
   int num=1;
   do
   {
            printf("%d\n",2*num);
            num++;
   }
    while(num<=10);
   return 0;
}
```

c. **For loop:-** A for loop is a more efficient loop structure in 'C' programming.

Syntax:-

```
for (initialization; condition; update)
{
   // Code to be executed
}
```

- The initial value of the for loop is performed only once.
- The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
- The incrementation/decrementation increases (or decreases) the counter by a set value.

Example:-

```
#include<stdio.h>
int main()
{
   int number;
   for(number=1;number<=10;number++)
  {
    printf("%d\n",number);
   }
   return 0;
 }
```

(b) C Program to Generate and Plot Pascal's Triangle:

Source code:-

```
#include <stdio.h>
// Function to calculate factorial
int factorial(int n)
 {
   if (n == 0 || n == 1)
     return 1;
  else
     return n * factorial(n - 1);
  }
// Function to calculate combination (n choose k)
int nCr(int n, int r)
{
  return factorial(n) / (factorial(r) * factorial(n - r));
}
// Function to print Pascal's Triangle
void printPascalsTriangle(int rows)
{
  for (int i = 0; i < rows; i++)
  {
```

```
        // Print spaces for alignment
        for (int j = 0; j < rows - i - 1; j++)
        {
            printf("   ");
        }
        // Print the numbers
        for (int j = 0; j <= i; j++)
        {
            printf("%6d", nCr(i, j));
        }
        printf("\n");
    }
}
int main()
{
    int rows;
    // Input the number of rows
    printf("Enter the number of rows for Pascal's Triangle: ");
    scanf("%d", &rows);
    // Print Pascal's Triangle
    printPascalsTriangle(rows);
    return 0;
}
```

Output:-

```
Enter the number of rows for Pascal's Triangle: 4
1
1 1
1 2 1
1 3 3 1
```

4. (a) What are the main elements of an array declaration? How to initialize an array?

(b) What is the difference between an array and pointer?

(c) Write a program to print the array elements in reverse order.

Ans:- (a) Main Elements of an Array Declaration and Array Initialization:

- **Array Type:** Specifies the type of elements the array will hold (e.g., int, float, char).
- **Array Name:** A valid identifier representing the array.
- **Array Size:** Indicates the number of elements in the array. It must be a constant integer expression.

Example:

```
int numbers[5];  // Declaration of an integer array named 'numbers' with size 5
```

- Array Initialization: Arrays can be initialized at the time of declaration.

Example:

```
int values[3] = {1, 2, 3};  // Initializing an array with values 1, 2, and 3
```

- If the array size is not specified, the compiler will determine it based on the number of values provided.

Example:

```
int numbers[] = {1, 2, 3, 4, 5};  // Compiler infers the size as 5
```

- Individual elements of the array can also be initialized separately.

Example:

```
int marks[3];
marks[0] = 85;
marks[1] = 90;
marks[2] = 78;
```

(b)

Feature	Array	Pointer
Memory Allocation	Contiguous block of memory for elements	Memory address pointing to a variable
Size	Fixed size at compile time	Can be dynamically allocated and resized
Initialization	Can be initialized at declaration or later	Must be initialized explicitly
Dynamic Memory Allocation	Not applicable	Can be dynamically allocated using **malloc**
Size Determination	Determined at compile time	Size is not inherently fixed, can vary
Access Syntax	Uses square brackets **[]**	Uses dereference operator *****
Example	**c int arr[3] = {1, 2, 3};**	**c int* ptr; ptr =**

(c)C program to print the array elements in reverse order:-

Source code:-

```
#include <stdio.h>
int main()
{
   int arr[]={1, 2, 3, 4, 5};
   int length = sizeof(arr)/sizeof(arr[0]);
   printf("Original array: \n");
   for (int i = 0; i < length; i++)
   {
     printf("%d ", arr[i]);
   }
   printf("\n");
   printf("Array in reverse order: \n");
   for (int i = length-1; i >= 0; i--)
```

```
    {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:-

Original array:

1 2 3 4 5

Array in reverse order:

5 4 3 2 1

5. (a) How to declare and initialize a Two-dimensional array? Discuss with examples.

(b) Write a C program to print the sum of diagonal elements of 2-D matrix.

(c) Write a C program to illustrate the use of indirection operator to access the value pointed by a pointer.

Ans:- (a) Declaring and Initializing a Two-dimensional Array:

- **Declaration:** data_type array_name[rows][columns];

- data_type: The type of data the array will hold (e.g., int, float, char).

- array_name: The name you choose for the array.

- rows: The number of rows in the array.

- columns: The number of columns in each row.

- **Initialization:**

1. During declaration:

```
int numbers[2][3] = {{1, 2, 3}, {4, 5, 6}};  // 2 rows, 3 columns
```

2. Using multiple lines (less common):

```
int numbers[2][3];
numbers[0][0] = 1;
numbers[0][1] = 2;
// ... and so on
```

3. Using nested loops:

```
int numbers[2][3];
for (int i = 0; i < 2; i++)
{
 for (int j = 0; j < 3; j++)
 {
    numbers[i][j] = i * 4 + j + 1;  // Example assignment
 }
}
```

Accessing elements:

```
int element = numbers[row_index][column_index];
```

Example usage:

```
#include <stdio.h>
int main()
{
    int matrix[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
    printf("Element at (1, 2): %d\n", matrix[1][2]); // Output: 7
    return 0;
}
```

(b) C Program to Print the Sum of Diagonal Elements of a 2-D Matrix:

Source code:-

```
#include <stdio.h>
#define ROWS 3
#define COLS 3
int main()
{
 int matrix[ROWS][COLS] =
  {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
  };
  int sum = 0;
 // Calculate the sum of diagonal elements
 for (int i = 0; i < ROWS; i++)
 {
   for (int j = 0; j < COLS; j++)
   {
      if (i == j)
      {
         sum += matrix[i][j];
       }
    }
  }
 // Print the sum
 printf("Sum of diagonal elements: %d\n", sum);
 return 0;
}
```

Output:-

```
Sum of diagonal elements: 15
```

(c) C Program Illustrating the Use of Indirection Operator:

Source code:-

```
#include <stdio.h>
int main()
{
   int value = 42;
   int *ptr = &value;  // Pointer pointing to the address of 'value'
   // Using indirection operator to access the value pointed by the pointer
   printf("Value pointed by the pointer: %d\n", *ptr);
  return 0;
 }
```

Output:-

Value pointed by the pointer: 42

6. (a) Write short notes on nested functions.

(b) Write a C program to explain call-by-reference parameter passing technique.

(c) Write a C program to illustrate call-by-value parameter passing technique.

Ans:- **(a) Nested Functions:-**

In C, nested functions are functions that are defined within another function. They have access to the variables and parameters of the outer function, and can be used to encapsulate functionality within a specific scope. Nested functions in C are not standard and are not supported by all compilers.

Example:-

```
#include <stdio.h>
int main()
{
   int outerFunction(int x)
  {
    int innerFunction(int y)
    {
      return y * 2;
    }
    return innerFunction(x) + 3;
  }
 printf("%d\n", outerFunction(5)); // Output: 13
 return 0;
}
```

(b) Call-by-Reference Parameter Passing:

In C, call-by-reference is achieved using pointers. When a variable is passed by reference, the function receives the address of the variable, allowing it to modify the original value.

Source code:-

```
#include <stdio.h>
// Function to swap two integers using call-by-reference
void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
 }
int main()
{
   int x = 5, y = 10;
   printf("Before swap: x = %d, y = %d\n", x, y);
   // Pass addresses of x and y to the swap function
   swap(&x, &y);
   printf("After swap: x = %d, y = %d\n", x, y);
   return 0;
 }
```

Output:-

```
Before swap: x = 5 , y = 10
After swap: x = 10 , y = 5
```

(c) Call-by-Value Parameter Passing:

Source code:-

```
#include <stdio.h>
// Function to swap two integers using call-by-value
void swap(int a, int b)
{
   int temp = a;
   a = b;
   b = temp;
 }
int main()
{
     int num1 = 5, num2 = 10;
     printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);
     // Calling the swap function with call-by-value
    swap(num1, num2);
```

```
        printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);
 return 0;
}
```

Output:-

Before swapping: num1 = 5, num2 = 10

After swapping: num1 = 5 , num3 = 10

7. (a) Why C is called function oriented language? What are the different types of functions available in C? Give examples of each type.

(b) Write a C program to find the factorial of a number using functions.

Ans:- **(a) C as a Function-Oriented Language:**

C is often referred to as a "function-oriented" language because it revolves around the concept of functions. In C, a program is structured as a collection of functions, and the execution starts from the main function. Functions in C allow for modular and organized code, making it easier to understand, maintain, and debug. The language provides a variety of built-in functions, and users can define their own functions as well.

- Types of Functions in C: There are two types of functions in C:
  - **Library Functions:** These are functions that are declared in the C header files. Some examples of library functions are:
    - printf(): This function is used to print output on the console.
    - scanf(): This function is used to read input from the console.
    - malloc(): This function is used to allocate memory dynamically.
    - free(): This function is used to free memory that was allocated dynamically.
    - strlen(): This function is used to calculate the length of a string.
    - strcmp(): This function is used to compare two strings.
  - **User-defined Functions:** These are functions that are created by the programmer. Some examples of user-defined functions are:
    - add(): This function is used to add two numbers.
    - subtract(): This function is used to subtract two numbers.
    - multiply(): This function is used to multiply two numbers.
    - divide(): This function is used to divide two numbers.
    - average(): This function is used to calculate the average of a set of numbers.

(b) C Program to Find the Factorial of a Number using Functions:

Source code:-

```
#include <stdio.h>
int fact(int n)
{
        if(n==0)
        return 1;
        else
```

```
        return (n*fact(n-1));
}
int main ()
{
        int n,factorial;
        printf ("Enter no:");
        scanf ("%d",&n);
        factorial=fact(n);
        printf ("The factorial of given number is =%d",factorial);
        return 0;
}
```

Output:-

Enter no: 5

The factorial of given number is = 120

8. (a) What do you mean by storage class?

(b) What are the different types of storage classes available in C? Explain each of them.

Ans:- **(a) Storage Class:** A storage class represent the visibility and a location of a variable. It tells from what part of code we can access a variable. A storage class in C is used to describe the following things.

1. The variable scope
2. The location where the variable will be stored
3. The initialized value of a variable
4. A lifetime of a variable
5. Who can access a variable

(b) Types of Storage Classes in C:

1. **Automatic Storage Class (auto):**
   - Variables declared with auto storage class are local by default.
   - They are created when a function is called and destroyed when the function exits.
   - This is the default storage class for all local variables.

```
auto int x; // 'auto' is optional, as it is the default storage class
```

2. **Register Storage Class (register):**
   - Variables declared with register storage class are stored in the CPU registers for faster access.
   - This storage class is a hint to the compiler to use register storage, but the compiler is not obliged to comply.
   - Similar to auto, the scope is local.

```
register int count;
```

3. **Static Storage Class (static):**
   - Variables declared as static have a lifespan equal to the entire program.
   - They retain their values between function calls.When used with local variables, it changes the scope to file scope (making it visible throughout the file).

```
static int globalVar; // File scope
```

4. **External Storage Class (extern):**
   - Extern is used to declare a variable that is defined in another file or at the global scope.
   - It is commonly used when you have global variables in one file that need to be accessed in another file.

```
extern int globalVar; // Declares a variable defined elsewhere
```

5. **Mutable Storage Class (mutable):**
   - Mutable is used in C++ for class members.
   - It allows a member of an object to be modified even if the object is declared as const.

```
class Example
{
    mutable int x; // Can be modified in a const member function
};
```

6. **Thread Local Storage Class (_Thread_local):**
   - Introduced in C11, _Thread_local is used to declare variables with thread-local storage.
   - Each thread gets its own instance of the variable.

```
_Thread_local int threadVar; // Thread-local variable
```

These storage classes provide flexibility in managing variables based on their scope, lifetime, and linkage in a program. The choice of a storage class depends on the specific requirements of the program and the desired behavior of the variables.

9. (a) Write detailed notes on C data types.

(b) What are header files? Give example. Why do we use header files?

Ans:- **(a) C Data Types:**

In C programming, data types are used to define the type of data that a variable can hold. The data type of a variable determines the amount of memory it occupies and the kind of operations that can be performed on it. C provides several built-in data types, which can be broadly categorized into the following types:

1. **Basic Data Types:**
   - int: Used to store integer values (whole numbers).
   - char: Used to store a single character.
   - float: Used to store floating-point numbers (decimal numbers).
   - double: Similar to float but provides more precision for floating-point numbers.
2. **Derived Data Types:**
   - Arrays: Collections of elements of the same data type.
   - Pointers: Variables that store memory addresses.

- Structures: Composite data types that group variables under a single name.
- Unions: Similar to structures but share the same memory location for all members.

3. Enumerated Data Types:

- enum: Allows you to define a set of named integer constants.

4. Void Data Type:

- void: Represents the absence of a data type. It is often used in function declarations to indicate that the function does not return a value.

5. User-Defined Data Types:

- typedef: Allows the programmer to create custom data types using existing ones.

Example:

Source code:-

```
#include <stdio.h>
int main()
{
      int age = 25;
     char grade = 'A';
     float salary = 55000.50;
     double pi = 3.141592653589793;
      printf("Age: %d\n", age);
      printf("Grade: %c\n", grade);
      printf("Salary: %.2f\n", salary);
      printf("Pi: %lf\n", pi);
   return 0;
 }
```

Output:-

```
Age: 25
Grade: a
Salary: 55000.50
Pi: 3.141593
```

(b) Header Files: Header files in C are files that contain function declarations, macro definitions, and other related information that can be used in multiple source files. They typically have a ".h" extension. Header files are included in C programs using the #include preprocessor directive.

Example:

```
// Example header file: math_operations.h
#ifndef MATH_OPERATIONS_H  // Header guards to prevent multiple inclusion
#define MATH_OPERATIONS_H
// Function declarations
int add(int a, int b);
int subtract(int a, int b);
double multiply(double a, double b);
```

```
double divide(double numerator, double denominator);
#endif
```

- Purpose of Header Files:

1. **Code Modularity:** Header files allow you to separate the interface (declarations) from the implementation, promoting modularity in your code.
2. **Code Reusability:** Functions and declarations defined in a header file can be reused across multiple source files.
3. **Readability:** Including only necessary headers keeps the main source file clean and makes it easier to understand and maintain.
4. **Avoiding Redundancy:** Header files help avoid redundancy by providing a centralized location for declarations that are used in multiple places.

10. (a) Classify the different types of decision making statements. Explain each of them.

(b) How switch case works without break statement. Write a Program to perform arithmetic operations using switch.

Ans:- **(a) Types of Decision-Making Statements:**

In C programming, decision-making statements allow you to control the flow of execution based on certain conditions. There are mainly three types of decision-making statements:

1. **if Statement:**

Syntax:

```
if (condition)
{
  // Code to be executed if the condition is true
}
```

The if statement is the simplest decision-making statement. It executes a block of code only if the specified condition is true.

2. **if-else Statement:**

Syntax:

```
if (condition)
{
  // Code to be executed if the condition is true
}
else
{
  // Code to be executed if the condition is false
}
```

The if-else statement allows you to execute different blocks of code based on whether the specified condition is true or false.

3. nested if-else Statement:

This involves using one or more if-else statements inside another. It allows for more complex decision-making scenarios.

Sysntax:

```
if (condition1)
{
   // Code to be executed if condition1 is true
   if (condition2)
   {
     // Code to be executed if both condition1 and condition2 are true
   }
   else
  {
    // Code to be executed if condition1 is true but condition2 is false
  }
}
else
{
  // Code to be executed if condition1 is false
}
```

4. switch Statement:

Syntax:

```
switch (expression)
{
  case constant1:  // Code to be executed if expression equals constant1
                   break;
  case constant2: // Code to be executed if expression equals constant2
                  break;
   // ... more cases ...
  default:
  // Code to be executed if expression doesn't match any case
}
```

The switch statement is used to select one of many code blocks to be executed based on the value of the expression.

(b) Switch Case without Break Statement: In C, when a switch statement is executed and a case block is matched, the code inside that case block is executed. If there is no break statement at the end of the case block, the control will fall through to the next case block, and subsequent code blocks will also be executed until a break statement is encountered or the end of the switch statement is reached.

perform arithmetic operations using switch:-

Sorce code:-

```
#include <stdio.h>
int main()
{
  char op;
  double num1, num2, result;
  printf("Enter an operator (+, -, *, /): ");
  scanf(" %c", &op);
  printf("Enter two numbers: ");
  scanf("%lf %lf", &num1, &num2);
  switch (op)
  {
     case '+': result = num1 + num2; break;
     case '-': result = num1 - num2; break;
     case '*': result = num1 * num2; break;
     case '/': result = (num2 != 0) ? num1 / num2 : (printf("Error: Division by zero.\n"), 1);
            break;
     default: printf("Error: Invalid operator.\n");
   }
   printf("Result: %.2lf\n", result);
   return 0;
}
```

Output:-

```
Enter an operator (+, -, *, /): +
Enter two numbers: 45   34
Result: 79.00
```

11. (a) Write an algorithm and develop a C program that reads N integer numbers and arrange them in ascending order.

(b) Write a program to check whether a string is palindrome or not.

Ans:-

(a) Sorting N Integer Numbers in Ascending Order:

Source code:-

```
#include <stdio.h>
void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
```

```
}
void bubbleSort(int arr[], int n)
{
   for (int i = 0; i < n - 1; i++)
   {
      for (int j = 0; j < n - i - 1; j++)
      {
         if (arr[j] > arr[j + 1])
        {
           swap(&arr[j], &arr[j + 1]);
        }
      }
    }
  }
  int main()
 {
    int N;
    printf("Enter the number of elements: ");
    scanf("%d", &N);
    int numbers[N];
    printf("Enter %d integer numbers:\n", N);
    for (int i = 0; i < N; i++)
       scanf("%d", &numbers[i]);
     // Sorting the numbers in ascending order
      bubbleSort(numbers, N);
     // Displaying the sorted numbers
     printf("\nNumbers in ascending order:\n");
     for (int i = 0; i < N; i++)
       printf("%d ", numbers[i]);
  return 0;
 }
```

Output:-

```
Enter the number of elements:3
Enter 3 integer numbers:
567
789
345
Number in ascending order:
345     567     789
```

(b) Checking if a String is a Palindrome:

Source code:-

```
#include <stdio.h>
#include <string.h>
int isPalindrome(char str[])
{
    int left = 0;
    int right = strlen(str) - 1;
    while (left < right)
    {
        if (str[left] != str[right])
        {
            return 0; // Not a palindrome
        }
        left++;
        right--;
    }
    return 1; // Palindrome
}
int main()
{
    char input[100];
    printf("Enter a string: ");
    scanf("%s", input);
    if (isPalindrome(input))
        printf("%s is a palindrome.\n", input);
    else
        printf("%s is not a palindrome.\n", input);
    return 0;
}
```

Output:-

Enter a string : 145

145 is not a palindrome

12. (a) Write a C Program to implement string copy operation that copies string str1 to another string str2 without using library function.

(b) Explain with example (i) Character string (ii) String literal.

Ans:-

(a) C Program to Implement String Copy Without Using Library Function:

Source code:-

```
#include <stdio.h>
void stringCopy(char str1[], char str2[])
{
   int i = 0;
   // Copy each character from str1 to str2
   while (str1[i] != '\0')
   {
      str2[i] = str1[i];
      i++;
   }
   // Add null character at the end of str2
   str2[i] = '\0';
}
int main()
{
   char sourceString[100], destinationString[100];
   printf("Enter a string: ");
   scanf("%s", sourceString);
   // Copying the string without using library function
   stringCopy(sourceString, destinationString);
   // Displaying the copied string
   printf("Original String: %s\n", sourceString);
   printf("Copied String: %s\n", destinationString);
   return 0;
}
```

Output:-

Enetr a string: BCA

Original string: BCA

Copied string: BCA

(b) Explanation with Examples:

(i) Character String: A character string in C is a sequence of characters stored in contiguous memory locations. It is represented using an array of characters. For example:

```
char name[] = "John";
```

Here, name is a character string containing the characters 'J', 'o', 'h', and 'n', followed by the null character '\0' indicating the end of the string.

(ii) String Literal: A string literal is a sequence of characters enclosed in double quotes. It is a constant array of characters and is automatically terminated with a null character '\0'.

```
char greeting[] = "Hello, World!";
```

In this example, "Hello, World!" is a string literal. The compiler automatically adds a null character at the end, making it a valid C string.

13. (a) Give the scope and life time of the following :

(i)External variable (ii) Static variable (iii) Automatic variable (iv) Register variable.

(b) Write a C program to check a number is a prime or not using recursion.

Ans:- **(a) Scope and Lifetime of Variables:**

(i) External Variable:

- **Scope:** External variables are declared outside of any function and are accessible to all functions in the entire program.
- **Lifetime:** The lifetime of an external variable is the entire duration of the program. It exists from the start of the program until it terminates.

(ii) Static Variable:

- **Scope:** Static variables have a scope limited to the block or function in which they are declared.
- **Lifetime:** The lifetime of a static variable is the entire duration of the program, similar to external variables. However, the static variable retains its value between function calls.

(iii) Automatic Variable:

- **Scope:** Automatic variables (also known as local variables) have a scope limited to the block or function in which they are declared.
- **Lifetime:** The lifetime of an automatic variable is limited to the duration of the block or function in which it is declared. It is created when the block is entered and destroyed when the block is exited.

(iv) Register Variable:

- **Scope:** Register variables have a scope limited to the block or function in which they are declared, similar to automatic variables.
- **Lifetime:** The lifetime of a register variable is also limited to the duration of the block or function in which it is declared. Like automatic variables, register variables are stored in registers for faster access, but the compiler may choose to ignore the "register" keyword.

(b) C Program to Check Prime Number Using Recursion:

Source code:-

```
#include <stdio.h>
// Function to check if a number is prime
int isPrime(int num, int i)
{
   if (i == 1)
  {
     return 1;  // Base case: 1 is not prime
   }
   else
   {
```

```
        if (num % i == 0)
        {
            return 0;  // If num is divisible by i, it's not prime
        }
        else
        {
            return isPrime(num, i - 1);  // Check the next divisor
        }
    }
}
int main()
{
    int num;
    // Input number from user
    printf("Enter a positive integer: ");
    scanf("%d", &num);
    // Check if the number is greater than 1 and call the recursive function
    if (num > 1 && isPrime(num, num / 2) == 1)
    {
        printf("%d is a prime number.\n", num);
    }
    else
    {
        printf("%d is not a prime number.\n", num);
    }
    return 0;
}
```

Output:-

Enter a positive integer: 45

45 is not prime no

14. (a) Write a program to maintain a record of "n‟ employee detail using an array of structures with three fields(id, name , salary) and print the details of employees whose salary is above 5000.

(b) Explain array of structure and structure within a structure with an example.

Ans:-

(a) Program to Maintain Employee Records:

Source code:

```
#include <stdio.h>
// Define the structure for employee details
struct Employee
{
    int id;
    char name[50];
    float salary;
};
int main()
{
    int n;
    printf("Enter the number of employees: ");
    scanf("%d", &n);
    struct Employee employees[n];
    for (int i = 0; i < n; i++)
    {
        printf("\nEnter details for Employee %d:\n", i + 1);
        printf("ID: ");
        scanf("%d", &employees[i].id);
        printf("Name: ");
        scanf("%s", employees[i].name);  // Assuming names do not have spaces
        printf("Salary: ");
        scanf("%f", &employees[i].salary);
    }
    // Print details of employees with salary above 5000
    printf("\nEmployees with salary above 5000:\n");
    for (int i = 0; i < n; i++)
    {
        if (employees[i].salary > 5000)
        {
            printf("ID: %d, Name: %s, Salary: %.2f\n", employees[i].id, employees[i].name,
            employees[i].salary);
        }
    }

    return 0;
}
```

Output:

```
Enter the number of employees: 3
Enter details for Employee 1:
```

```
ID: 1
Name: Tridib
Salary: 6000
Enter details for Employee 2:
ID: 2
Name: Soumadip
Salary: 5400
Enter details for Employee 3:
ID: 3
Name: Abhrajoyti
Salary: 5700
Employees with salary above 5000:
ID: 1, Name: Tridib, Salary: 6000.00
ID: 2, Name: Soumadip, Salary: 5400.00
ID: 3, Name: Abhrajoyti, Salary: 5700.00
```

(b) Array of Structure and Structure within a Structure:

- Array of Structure: An array of structures is a data structure in which each element is an instance of a structure. It allows you to store and organize multiple records of the same structure type in a contiguous block of memory.

Example:

```
#include <stdio.h>
struct Point
{
    int x;
    int y;
};
int main()
{
    struct Point points[3]; // Array of structures
    // Input coordinates for each point
    for (int i = 0; i < 3; i++)
    {
        printf("Enter coordinates for point %d: ", i + 1);
        scanf("%d %d", &points[i].x, &points[i].y);
    }
    // Print the entered coordinates
    for (int i = 0; i < 3; i++)
        printf("Point %d: (%d, %d)\n", i + 1, points[i].x, points[i].y);
    return 0;
}
```

Output:-

```
Enter coordinates for point 1:  23   46
Enter coordinates for point 2:  67   45
Enter coordinates for point 3:  78   37
Point 1:  (23, 46)
Point 2:  (67, 45)
Point 3:  (78, 37)
```

- **Structure within a Structure:** A structure within a structure, also known as a nested structure, involves defining one structure inside another. This allows you to create more complex data structures.

Example:

```
#include <stdio.h>
struct Date
{
   int day;
   int month;
   int year;
};
struct Employee
{
   int id;
   char name[50];
   float salary;
   struct Date joiningDate; // Structure within a structure
};
int main()
{
   struct Employee emp;
   // Input employee details
   printf("Enter Employee ID: ");
   scanf("%d", &emp.id);
   printf("Enter Employee Name: ");
   scanf("%s", emp.name);
   printf("Enter Employee Salary: ");
   scanf("%f", &emp.salary);
   printf("Enter Joining Date (day month year): ");
   scanf("%d %d %d", &emp.joiningDate.day, &emp.joiningDate.month,&emp.joiningDate.year);
   //Print employee details
   printf("\nEmployeeDetails:\n");
    printf("ID:%d\nName:%s\nSalary:%.2f\nJoiningDate:%d/%d/%d\n",emp.id,emp.name,emp.
```

```
    salary,emp.joiningDate.day,emp.joiningDate.month,emp.joiningDate.year);
    return 0;
  }
```

Output:-

```
Enter Employee ID: 2
Enter Employee Name: Tridib
Enter Employee Salary: 6800
Enter Joining Date (day month year): 17  03  2005
Employee Details:
ID: 2
Name: Tridib
Salary: 6800.00
Joining Date: 17/3/2005
```

15. (a) Enlist the File Operations. Explain each operations.

(b) List the opening modes in standard I/O.

Ans:- **(a) File Operations:**

1. **Opening a File:**
   - Description: This operation involves creating a connection between the program and a file on the disk. It prepares the file for subsequent operations like reading or writing.
   - Function/Method: open()
2. **Reading from a File:**
   - Description: This operation allows the program to retrieve data from an opened file. The data can be read in various ways, such as reading a single character, a line, or a block of data.
   - Function/Method: read(), readline(), readlines()
3. **Writing to a File:**
   - Description: This operation involves adding new data to a file. It can be done by writing a single character, a line, or a block of data to the file.
   - Function/Method: write(), writelines()
4. **Closing a File:**
   - Description: After performing operations on a file, it is important to close it. Closing a file ensures that all the changes made are saved, and system resources are freed up.
   - Function/Method: close()
5. **Seeking a Specific Position in a File:**
   - Description: This operation allows the program to move the file cursor to a specific position within the file. It is useful for repositioning before reading or writing data.
   - Function/Method: seek()
6. **Renaming or Moving a File:**
   - Description: This operation involves changing the name or location of a file on the disk.

- Function/Method: os.rename(), os.replace()

7. **Deleting a File:**
   - Description: This operation involves removing a file from the disk.
   - Function/Method: os.remove()

(b) Opening Modes in Standard I/O: Standard I/O (Input/Output) modes define how a file should be opened, indicating whether it should be read from, written to, or both. In Python, the open() function is used with a mode parameter to specify the file's intended use. Here are some common opening modes:

1) **'r' - Read Mode:**
   - Description: Opens the file for reading. Raises an error if the file does not exist.
2) **'w' - Write Mode:**
   - Description: Opens the file for writing. If the file already exists, it truncates the file to zero length. If the file does not exist, it creates a new file.
3) **'a' - Append Mode:**
   - Description: Opens the file for writing, but appends to the end of the file instead of truncating it. Creates a new file if it doesn't exist.
4) **'b' - Binary Mode:**
   - Description: Opens the file in binary mode, indicating that the file should be treated as a binary file. This is often used in conjunction with other modes (e.g., 'rb' or 'wb').
5) **'x' - Exclusive Creation:**
   - Description: Opens the file for exclusive creation. If the file already exists, the operation will fail.
6) **'t' - Text Mode (Default):**
   - Description: Opens the file in text mode. This is the default mode if no mode is specified. It is used to indicate that the file should be treated as a text file.

16. (a) What is Token? What are the different types of token available in C language.

(b) What is an identifier (variable)? What are the rules to construct identifier (varible)?Classify the following as valid/invalid Identifiers: i) num2 ii) $num1 iii) +add iv) a_2 v) 199_space

Ans:- **(a) Token in C language:**

In C programming language, a token is the smallest unit in the source code. The compiler breaks down the source code into tokens to understand and process it. Tokens can be keywords, identifiers, constants, operators, and punctuation symbols.

- **Types of tokens in C:**

I. Keywords: Words reserved by the C language (e.g., int, for, if).
II. Identifiers: Names given to various program elements (e.g., variables, functions).
III. Constants: Numeric or string constants (e.g., 42, 3.14, "hello").

IV. String literals: Sequences of characters enclosed in double quotes.

V. Operators: Symbols representing computations or operations (e.g., +, -, *).

VI. Punctuation symbols: Special symbols like braces, commas, semicolons, etc.

(b) Identifier (Variable) in C: An identifier is a name given to a program element such as a variable, function, array, etc. It is used to identify and refer to a specific memory location.

- **Rules for constructing identifiers (variables) in C:**

a) Must begin with a letter (uppercase or lowercase) or an underscore (_).

b) Can be followed by letters, digits, or underscores.

c) No spaces or special characters (except underscore) are allowed.

d) Case-sensitive (e.g., num and Num are different identifiers).

- Valid/Invalid Identifiers:

i) num2: Valid (starts with a letter and followed by letters/digits).

ii) $num1: Invalid (contains a special character $).

iii) +add: Invalid (starts with a special character +).

iv) a_2: Valid (contains letters, digits, and underscores).

v) 199_space: Invalid (starts with a digit).

17. (a) Write a C program that takes three coefficients (a,b,and c) of a quadtatic equation ; (ax2+bx+c) as input and compute all possible roots and print them with appropriate messages.

(b) Write a C program to find GCD of two numbers using ternary operator and for loop.

Ans:- (a) C Program to Compute Roots of a Quadratic Equation:

Source code:-

```
#include <stdio.h>
#include <math.h>
int main()
{
   // Input coefficients
   float a, b, c;
   printf("Enter coefficients a, b, and c: ");
   scanf("%f %f %f", &a, &b, &c);
   // Calculate discriminant
   float discriminant = b * b - 4 * a * c;
   // Check for real or complex roots
   if (discriminant > 0)
   {
      float root1 = (-b + sqrt(discriminant)) / (2 * a);
      float root2 = (-b - sqrt(discriminant)) / (2 * a);
      printf("Roots are real and different:\n");
```

```
        printf("Root 1 = %.2f\n", root1);
        printf("Root 2 = %.2f\n", root2);
    }
    else if (discriminant == 0)
    {
        float root = -b / (2 * a);
        printf("Roots are real and same:\n");
        printf("Root = %.2f\n", root);
    }
    else
    {
        float realPart = -b / (2 * a);
        float imaginaryPart = sqrt(-discriminant) / (2 * a);
        printf("Roots are complex and different:\n");
        printf("Root 1 = %.2f + %.2fi\n", realPart, imaginaryPart);
        printf("Root 2 = %.2f - %.2fi\n", realPart, imaginaryPart);
    }
    return 0;
}
```

Output:-

```
Enter coefficients a, b, and c: 4  5  6
Roots are complex and different:
Root 1 = -0.63+ 1.05i
Root 2 = -0.63- 1.05i
```

(b) C Program to Find GCD of Two Numbers using Ternary Operator and For Loop:

Source code:-

```
#include <stdio.h>
// Function to find GCD using Euclidean algorithm
int findGCD(int a, int b)
{
    return b == 0 ? a : findGCD(b, a % b);
}
int main()
{
    // Input two numbers
    int num1, num2;
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);
    // Find GCD using the findGCD function
    int gcd = findGCD(num1, num2);
```

```
    // Output the result
    printf("GCD of %d and %d is: %d\n", num1, num2, gcd);
    return 0;
}
```

Output:-

```
Enter two numbers : 4  6
GCD of 4 and 6 is :  2
```

18. (a) Write an algorithm and develop a C program to search an integer from N numbers in ascending order using binary searching technique.

(b) Write a C program to find the transpose of a given matrix.

Ans:- (a) Algorithm and C Program for Binary Search on Sorted Array:

- **Alogrithm:**

1) Input the size N of the array.
2) Input N integers in ascending order.
3) Input the target integer to search.
4) Initialize low to 0 and high to N-1.
5) Repeat until low is less than or equal to high:
    a. Calculate the middle index as (low + high) / 2.
    b. If the middle element is equal to the target, print the index and break.
    c. If the middle element is less than the target, update low to middle + 1.
    d. If the middle element is greater than the target, update high to middle - 1.
6) If the loop exits without finding the target, print "Integer not found."

- C program:

Source code:-

```
#include <stdio.h>
// Function to perform binary search on a sorted array
int binarySearch(int arr[], int size, int target)
{
    int low = 0, high = size - 1;
    while (low <= high)
    {
        int mid = (low + high) / 2;
        if (arr[mid] == target)
        {
            return mid; // Target found
        }
        else if (arr[mid] < target)
        {
            low = mid + 1;
```

```
        }
        else
        {
            high = mid - 1;
        }
    }
    return -1; // Target not found
}
int main()
{
    int N;
    // Input the size of the array
    printf("Enter the size of the array: ");
    scanf("%d", &N);
    int arr[N];
    // Input N integers in ascending order
    printf("Enter %d integers in ascending order:\n", N);
    for (int i = 0; i < N; i++)
    {
        scanf("%d", &arr[i]);
    }
    int target;
    // Input the target integer to search
    printf("Enter the integer to search: ");
    scanf("%d", &target);
    // Perform binary search
    int result = binarySearch(arr, N, target);
    // Output the result
    if (result != -1)
    {
        printf("Integer %d found at index %d.\n", target, result);
    }
    else
    {
        printf("Integer not found.\n");
    }
    return 0;
}
```

Output:-

```
Enter the size of the array: 2
```

```
Enter 2 integers in ascending order:
4
3
Enter the integer to search: 4
Integer 4 found at index 0.
```

(b) C Program to Find Transpose of a Matrix:

Source code:-

```
#include <stdio.h>
int main()
{
  int rows, cols;
  printf("Enter the number of rows: ");
  scanf("%d", &rows);
  printf("Enter the number of columns: ");
  scanf("%d", &cols);
  int matrix[rows][cols];
  printf("Enter the matrix elements:\n");
  for (int i = 0; i < rows; i++)
  {
    for (int j = 0; j < cols; j++)
    {
      printf("Enter element [%d][%d]: ", i, j);
      scanf("%d", &matrix[i][j]);
    }
  }
  printf("\nOriginal Matrix:\n");
  for (int i = 0; i < rows; i++)
  {
    for (int j = 0; j < cols; j++)
    {
      printf("%d\t", matrix[i][j]);
    }
    printf("\n");
  }
  int transpose[cols][rows];
  for (int i = 0; i < cols; i++)
  {
    for (int j = 0; j < rows; j++)
    {
      transpose[i][j] = matrix[j][i];
```

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

Output:-

```
Enter the number of rows: 2
Enter the number of columns: 2
Enter the matrix elements:
Enter element [0][0]: 4 7
Enter element [0][1]: Enter element [1][0]: 3 8
Enter element [1][1]:
Original Matrix:
4       7
3       8
Transposed Matrix:
4       3
7       8
```

19. (a) Write a C program to maintain a record of "n" students details using an array of structures with four fields(roll no,name,marks,and grade). Assume appropriate data type for each field. Print the marks of the student given the student name as input.

(b) write a program to find nth term of Fibonacci series using function.

Ans:- **(a) C program to maintain a record of "n" students details using an array of structures and print the marks of a student given the student name as input:**

Source code:-

```
  #include <stdio.h>
 #include <string.h>
// Define the structure for student details
struct Student
{
```

```
  int roll_no;
  char name[50];
  float marks;
  char grade;
};
int main()
{
   int n;
   printf("Enter the number of students: ");
   scanf("%d", &n);
  // Declare an array of structures to store student details
  struct Student students[n];
  // Input details for each student
  for (int i = 0; i < n; i++)
  {
     printf("Enter details for student %d:\n", i + 1);
     printf("Roll No: ");
     scanf("%d", &students[i].roll_no);
     printf("Name: ");
     scanf("%s", students[i].name);
     printf("Marks: ");
     scanf("%f", &students[i].marks);
     // Assign grade based on marks
     if (students[i].marks >= 90)
    {
        students[i].grade = 'A';
     }
     else if (students[i].marks >= 80)
    {
        students[i].grade = 'B';
     }
     else if (students[i].marks >= 70)
    {
        students[i].grade = 'C';
     }
     else
     {
        students[i].grade = 'F';
     }
  }
```

```
  // Print marks of a student given the student name as input
  char search_name[50];
  printf("\nEnter the name of the student to find marks: ");
  scanf("%s", search_name);
  // Search for the student by name
  int found = 0;
  for (int i = 0; i < n; i++)
  {
     if (strcmp(students[i].name, search_name) == 0) {
        printf("Marks of %s: %.2f\n", search_name, students[i].marks);
        found = 1;
        break;
     }
  }
  if (!found)
 {
     printf("Student with name %s not found.\n", search_name);
  }
  return 0;
 }
```

Output:-

```
 Enter the number of students: 2
Enter details for student 1:
Roll No: 70
Name: Tridib
Marks: 95
Enter details for student 2:
Roll No: 02
Name: Abhrajyoti
Marks: 45
Enter the name of the student to find marks: Tridib
Marks of Tridib: 95.00
```

(b) Find nth term of Fibonacci series using function:-

Source code:-

```
#include <stdio.h>
// Function to calculate the nth term of the Fibonacci series
int fibonacci(int n)
{
   if (n <= 1)
  {
```

```
    return n;
  }
  else
  {
    return fibonacci(n - 1) + fibonacci(n - 2);
  }
}
int main()
{
  int n;
  // Input: Get the value of n from the user
  printf("Enter the value of n: ");
  scanf("%d", &n);
  // Validate if n is non-negative
  if (n < 0)
  {
    printf("Please enter a non-negative integer.\n");
    return 1; // Return error code
  }
  // Output: Calculate and print the nth term of the Fibonacci series
  printf("The %dth term of the Fibonacci series is: %d\n", n, fibonacci(n));
  return 0; // Return success code
}
```

Output:-

```
Enter the value of n: 10
The 10th term of the Fibonacci series is: 55
```

20. (a) Write the functions for random access file processing.

1. fseek() 2. ftell() 3. rewind()

(b) Write short notes on : i) fprintf() and ii) fscanf()

Ans:- **(a)Random Access File Processing Functions:**

1) **fseek() Function:**

Syntax:

```
int fseek(FILE *stream, long int offset, int origin);
```

Description:

- fseek() is used to set the file position indicator for the given stream.
- stream is a pointer to the FILE object that represents the stream.
- offset is the number of bytes to offset from the specified origin.
- origin specifies the reference point for the offset and can take values like SEEK_SET, SEEK_CUR, or SEEK_END.

2) ftell() Function:

Syntax:

long int ftell(FILE *stream);

Description:

- ftell() returns the current file position indicator of the given stream.
- The return value represents the current offset in bytes from the beginning of the file.

3) rewind() Function:

Syntax:

void rewind(FILE *stream);

Description:

- rewind() sets the file position indicator for the given stream to the beginning of the file.
- It is equivalent to calling fseek(stream, 0L, SEEK_SET).

(b)Short Notes on fprintf() and fscanf():

i) fprintf() Function:

Syntax:

int fprintf(FILE *stream, const char *format, ...);

Description:

- fprintf() is used to write formatted data to the file pointed to by stream.
- It works similarly to printf() but writes the formatted output to a file.
- The format string specifies the format of the output, and additional arguments provide the values to be formatted.

ii) fscanf() Function:

Syntax:

int fscanf(FILE *stream, const char *format, ...);

Description:

- fscanf() is used to read formatted data from the file pointed to by stream.
- It works similarly to scanf() but reads input from a file instead of the standard input.
- The format string specifies the expected format of the input, and additional arguments are pointers to variables where the values will be stored.

These functions are part of the standard C library and are commonly used for file I/O operations in C programming.