Theory Exercise

A-1: C programming is a high-level, general-purpose programming language developed in the 1970s. It is widely used for system and application software development due to its efficiency and portability. C provides low-level access to memory and allows direct manipulation of hardware, making it ideal for developing operating systems, embedded systems, and other performance critical applications.

A-2: To install a C compiler, first, download and install a compiler like GCC On Linux, you can install GCC using the command sudo apt-get install gcc. On Windows, you can use MinGW to install GCC. After installation, ensure the compiler is added to the system's PATH variable to compile C programs from the command line.

A-3: C program typically consists of **three main parts**:
1. **Preprocessor Directives**: Instructions like **#include** to include libraries, which are processed before compilation.
2. **Main Function**: The entry point of the program, written as **int main(),** where execution begins.
3. **Functions and Statements**: The body of the program containing logic and functions to perform specific tasks, with the main function returning an integer value (return 0;).

Comments in C : In C, comments are used to add explanatory notes within the code, which are ignored by the compiler. They are two types:
1. **Single-line comments:** Use // to comment out a single line.
2. **Multi-line comments**: Enclosed between /* and */, used for comments spanning multiple lines.
Comments help improve code readability and provide explanations for complex sections.

Data Types in C : A data type specifies what type of data a Variable can store such as integer, float, character, double

A-4: Operator can be defined as a symbol that takes one or more operands such as variables, expressions or values and operates on them to give an output.

1. Arithmetic Operators : Used to perform basic mathematical operations.

Operator	Meaning	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus (remainder)	a % b

2. Relational Operators : Used to compare two values. The result is either true or false.

Operator	Meaning	Example
==	Equal to	a == b
!=	Not equal to	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater or equal	a >= b
<=	Less or equal	a <= b

3. Logical Operator : Used to combine multiple conditions (usually with if, while, etc.).

Operator	Meaning	Example
&&	Logical AND	(a > 0 && b > 0)
!	Logical NOT	!(a > b)

4. Assignment Operators : Used to assign values to variables.

Operator	Meaning	Example
=	Assign	a = 10
+=	Add and assign	a += 5
-=	Subtract and assign	a -= 3
*=	Multiply and assign	a *= 2
/=	Divide and assign	a /= 2
%=	Modulus and assign	a %= 2

5. Increment and Decrement Operators : Used to increase or decrease a value by 1.

Operator	Meaning	Example
++	Increment (add 1)	a++ or ++a
--	Decrement (subtract 1)	a-- or --a

6. Bitwise Operators : Used to perform operations on bits (binary level).

Operator	Meaning	Example
&	Bitwise AND	a & b
^	Bitwise XOR	a ^ b
~	Bitwise NOT	~a
<<	Left shift	a << 2
>>	Right shift	a >> 2

7. Conditional (Ternary) Operator

A shortcut for if-else. It uses **? :** and works with three parts.

Syntax	Meaning	Example
condition ? x : y	If condition is true, return x; else return y	a > b ? a : b

A-5:

1. **if Statement** : Used to execute code only if a condition is true.

Syntax:

```
if (condition) {
    // code to run if condition is true
}
```

2. **if-else Statement** : Used to choose between two options: if the condition is true, do one thing; else, do another.

Syntax:

```
if (condition) {
    // true part
} else {
    // false part
}
```

3. **Nested if-else :** An if or if-else statement inside another if or else. Used when there are multiple conditions to check.

Syntax:

```
if (condition1) {
    if (condition2) {
        // both conditions are true
    } else {
        // only condition1 is true
    }
} else {
    // condition1 is false
}
```

4. **switch Statement :** Used when you have multiple values to check for one variable. It's a cleaner alternative to many if-else statements.

Syntax:

```
switch (expression) {
    case value1:
        // code
        break;
    case value2:
        // code
        break;
    default:
        // code if no case matches
}
```

A-6:

For Loops: It is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

While Loops: It repeatedly executes a target statement as long as the given condition is True.

Do-while Loops: It is similar to while loop, except the fact that it executes once even, while condition is false.

A-7:

The Break Statement:
The break statement is used inside loop or switch statement. When compiler finds the break statement inside a loop, compiler will abort the loop and continue to execute statements followed by loop.
Syntax: break ;

Continue statement:
The continue statement is also used inside loop. When compiler finds the continue statement inside a loop, compiler will skip all the following statements in the loop and resume the next loop iteration.
Syntax: continue ;

The GOTO statement: By using this goto statements we can transfer the control from current location to anywhere in the program.
To do all this we have to specify a label with goto and the control will transfer to the location where the label is specified.

A-8:

A **function** is a set of statements that take inputs, do some specific computation and produce output.

Syntax :
return_type function_name(parameter list)
{
body of the function
}

How to call a function:
When a program calls a function, the compiler gets redirected towards the function definition
Function Call simply pass the required parameters along with the function name.

A-9:

An **array** is used to store a collection of data, and it is often used as a collection of variables of the same data type

In declaration we specify the type of element and size of the array Element.

Syntax :
data_type array_name [size] ;

Ex: int roll[20];

There are two types of Array:

1. Single/ One Dimensional Array
2. Multi Dimensional Array

1. One-Dimensional Array : A one-dimensional array is a **list of values** of the same type, stored in a single row or column.

Example:

```
int marks[4] = {80, 85, 90, 95};
```

2. Multi-Dimensional Array: A multi-dimensional array is an **array of arrays**. The most common is a **two-dimensional array**, which looks like a **table** with rows and columns.

Example:

```
int matrix[2][3] = {
                        {1, 2, 3},
                        {4, 5, 6}
                    };
```

A-10:

Pointers in C are variables that store the memory address of another variable.

You can access the value stored at the memory address using the dereference **operator `*`**.

The address of a variable is obtained using the address-of **operator `&`**.

Pointer Declaration: int *ptr = &x;

Pointers are important in C because they allow **direct access to memory** using memory addresses.

They are used to **store the address of a variable**, which helps in efficient data handling, especially in large programs.

A-11:

String Handling Functions in C

These functions are defined in the <string.h> header file and are used to **work with strings**.

1. **strlen() – String Length :** Returns the **number of characters** in a string (excluding the null `\0` character).

   ```
   int len = strlen("Hello");  // len = 5
   ```

2. **`strcpy()` – String Copy :** Copies the contents of one string into another.

```
char str1[20];
strcpy(str1, "World");  // str1 now contains "World"
```

3. `strcat()` – String Concatenation

Appends (adds) one string to the end of another.

```
char str1[20] = "Hello ";
char str2[] = "World";
strcat(str1, str2);  // str1 now contains "Hello World"
```

4. `strcmp()` – String Comparison

Compares two strings:

- Returns `0` if both are equal
- Returns `<0` if first string is smaller
- Returns `>0` if first string is greater

```
strcmp("apple", "banana");  // returns a negative value
```

A-12:

Structures are used to represent complex data entities, such as records in databases. They are defined using the `**struct**` keyword and can be accessed using the dot operator (`.`).

Structure variables can be **initialized** separately (s1.age = 20;), directly (struct Student s2 = {"Alice", 21, 90.0};), using typedef, or as an array (struct Student students[2] = {{"Mike", 19, 78.5}, {"Sara", 20, 92.0}};).

A-13:

Importance of File Handling in C

File handling is important because it allows a program to **store data permanently** on a disk.
Without file handling, all data would be lost when the program ends.
It helps in saving, reading, modifying, and processing data stored in files for future use.

1. Opening a File: `fopen()` to open a file.

```
FILE *fp;
fp = fopen("data.txt", "r");  // open for reading
```

Modes:

- `"r"` – Read
- `"w"` – Write (creates new file or clears existing)
- `"a"` – Append
- `"r+"`, `"w+"`, `"a+"` – Read/write versions

2. Writing to a File : `fprintf()` or `fputs()` to write data.

```
FILE *fp = fopen("data.txt", "w");
fprintf(fp, "Hello, File!");
fclose(fp);
```

3. Reading from a File : `fscanf()`, `fgets()`, or `fgetc()` to read data.

```
FILE *fp = fopen("data.txt", "r");
char str[100];
fgets(str, 100, fp);  // reads a line from the file
fclose(fp);
```

4. Closing a File : Always use `fclose()` to close an opened file.

```
fclose(fp);
```