# Module 2 – Introduction to Programming (Theory Exercises)

## 1. Overview of C Programming

### Theory Exercise:

The C programming language was developed in the early 1970s by Dennis Ritchie at Bell Labs. It evolved from two earlier languages, BCPL and B, and was designed as a system programming language for developing the UNIX operating system. Its portability, efficiency, and rich set of features made it widely adopted in the computing world.  
  
Over time, C influenced many other popular programming languages, including C++, Java, and C#. It introduced key concepts like structured programming, low-level memory access, and modularity, which remain relevant in modern programming. Despite being over 50 years old, C is still widely used today due to its performance, control over hardware, and simplicity. It is considered the foundation for learning other languages and is crucial in fields like embedded systems, operating systems, and system-level software development.

## 2. Setting Up Environment

### Theory Exercise:

To install a C compiler and set up an Integrated Development Environment (IDE), follow these steps:  
1. Install GCC (GNU Compiler Collection):  
 - On Windows: Install MinGW or TDM-GCC.  
 - On Linux: GCC is usually pre-installed; otherwise, install it via package managers (e.g., apt-get install gcc).  
 - On macOS: Install Xcode Command Line Tools.  
2. Install an IDE:  
 - DevC++: Lightweight IDE for C/C++.  
 - Code::Blocks: Open-source IDE with debugging and project management features.  
 - Visual Studio Code: Modern, extensible editor with support for C through extensions.  
3. Configure IDE:  
 - Link the IDE to the installed C compiler.  
 - Set the compiler path in the IDE settings.  
 - Test the setup by writing and running a simple 'Hello, World!' program.

## 3. Basic Structure of a C Program

### Theory Exercise:

A C program generally consists of the following components:  
1. Header Files: Preprocessor directives that include libraries, e.g., #include <stdio.h>.  
2. main() Function: The entry point of the program.  
3. Comments: Used to explain the code. Single-line (//) and multi-line (/\* ... \*/).  
4. Data Types: Define the type of data (int, float, char, etc.).  
5. Variables: Named storage for data values.  
  
Example:  
#include <stdio.h>  
int main() {  
 int age = 20; // variable declaration  
 float marks = 88.5;  
 char grade = 'A';  
 printf("Age: %d, Marks: %.2f, Grade: %c", age, marks, grade);  
 return 0;  
}

## 4. Operators in C

### Theory Exercise:

Operators in C are symbols that perform operations on variables and values:  
1. Arithmetic Operators: +, -, \*, /, %  
2. Relational Operators: ==, !=, >, <, >=, <=  
3. Logical Operators: &&, ||, !  
4. Assignment Operators: =, +=, -=, \*=, /=  
5. Increment/Decrement: ++, --  
6. Bitwise Operators: &, |, ^, ~, <<, >>  
7. Conditional (Ternary) Operator: ? :

## 5. Control Flow Statements in C

### Theory Exercise:

Decision-making in C allows conditional execution:  
1. if: Executes a block if the condition is true.  
2. if-else: Executes one block if true, another if false.  
3. Nested if-else: Multiple if-else structures inside each other.  
4. switch: Used for multi-way branching based on a variable value.

## 6. Looping in C

### Theory Exercise:

C provides three main types of loops:  
1. for loop: Best when the number of iterations is known.  
2. while loop: Checks condition before execution; used when iterations depend on a condition.  
3. do-while loop: Executes at least once since condition is checked after execution.

## 7. Loop Control Statements

### Theory Exercise:

1. break: Immediately exits the loop or switch.  
2. continue: Skips the current iteration and proceeds to the next.  
3. goto: Transfers control to a labeled statement in the program.

## 8. Functions in C

### Theory Exercise:

Functions are blocks of code that perform a specific task.  
1. Declaration: Tells the compiler about the function (prototype).  
2. Definition: Contains the actual code.  
3. Call: Executes the function from main or other functions.  
Functions improve modularity and code reuse.

## 9. Arrays in C

### Theory Exercise:

Arrays store multiple values of the same type in contiguous memory locations.  
1. One-dimensional array: List of elements (e.g., int arr[5]).  
2. Multi-dimensional array: Arrays of arrays (e.g., int matrix[3][3]).  
Useful for handling collections of data efficiently.

## 10. Pointers in C

### Theory Exercise:

Pointers are variables that store memory addresses.  
Declaration: int \*ptr;  
Initialization: int x = 10; ptr = &x;  
Importance: Allow dynamic memory management, efficient array handling, and manipulation of data structures.

## 11. Strings in C

### Theory Exercise:

Strings in C are arrays of characters ending with a null character '\0'.  
Common string handling functions:  
1. strlen(): Returns the length of a string.  
2. strcpy(): Copies one string into another.  
3. strcat(): Concatenates two strings.  
4. strcmp(): Compares two strings.  
5. strchr(): Finds the first occurrence of a character.

## 12. Structures in C

### Theory Exercise:

Structures are user-defined data types that group variables of different types under a single name.  
Declaration: struct Student { char name[50]; int roll; float marks; };  
Initialization: struct Student s1 = {"John", 1, 85.5};  
Access: s1.roll, s1.marks.

## 13. File Handling in C

### Theory Exercise:

File handling allows permanent storage of data.  
Operations include:  
1. fopen(): Open a file.  
2. fclose(): Close a file.  
3. fprintf()/fputs(): Write to a file.  
4. fscanf()/fgets(): Read from a file.  
It is essential for applications that require data storage beyond program execution.