Module 2 - Introduction to Programming

1. Overview of C Programming.

- THEORY EXERCISE:

Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.

→ History and evolution of C –

The C language was created in the early 1970s by Dennis M. Ritchie at Bell Laboratories. The language grew out of efforts to develop the UNIX operating system, initially written in assembly language. Before C, programmers lacked a language that could offer both low-level hardware control and high-level programming capabilities. Ritchie designed C as a successor to the B and BCPL languages, aiming to create a more efficient tool for system-level development while retaining portability across machines.

As usage grew through the 1970s and 1980s, different organizations implemented their own variations of C, leading to inconsistencies. To address this, the American National Standards Institute (ANSI) standardized C in 1989, creating ANSI C (also known as C89). This standard ensured uniformity across compilers and systems, allowing programs to run predictably.

C's Importance in Computer Science:

C has remained critical for several reasons. First, it provides low-level control, allowing direct manipulation of memory through pointers and efficient interaction with hardware. This makes it ideal for systems programming, where performance and precision are essential.

C is portable. Programs written in C can easily be adapted to new hardware platforms, helping it become a dominant language in an era where computers varied widely in architecture. This portability helped spread UNIX systems globally, influencing networking, operating systems, and academic computing.

Why C Is Still Used Today:

i. **Performance and Efficiency**: C generates compact, fast machine code with minimal runtime overhead. This is critical in domains where system resources are limited.

ii. **System-Level Programming:** Operating systems such as UNIX, Linux, Windows, and macOS rely heavily on C. Many device drivers, compilers, and interpreters are also written in C.

iii. **Embedded Systems:** C dominates in firmware and embedded devices—from microcontrollers in household appliances to automotive systems—due to its closeness to hardware and deterministic behaviour.

iv. **Educational Value:** C teaches core concepts including memory management, data representation, and system interfaces. As a result, it remains part of the curriculum in computer science and engineering programs worldwide

v. **Longevity and Community Support:** With decades of libraries, tools, and documentation, C benefits from an extensive ecosystem that is difficult to replace.

- LAB EXERCISE:
  Research and provide three real-world applications where C programming is extensively used, such as in embedded systems, operating systems, or game development.

→ **1. Embedded Systems**

C is widely used in embedded devices such as microcontrollers, IoT devices, automotive control systems, medical equipment, and consumer electronics.

Reason:

C provides direct access to hardware, efficient memory usage, and fast execution, which are crucial in resource-limited embedded environments.

2. Operating Systems

Many major operating systems are written partially or entirely in C, including Unix, Linux, Windows, and macOS kernels.

Reason:

C allows low-level interaction with memory and hardware while still offering portability across platforms—making it ideal for system-level programming.

3. Game Development (Game Engines)

Many game engines and performance-critical game modules are written in C or C++. For example, early game engines like Doom and Quake used C extensively.

2. Setting Up Environment

- THEORY EXERCISE:

Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like Dev C++, VS Code, or Code Blocks.

→ **1. Dev-C++ (Windows Only)**

Steps:

1. Download and install Dev-C++.
2. Open Dev-C++, go to **File → New → Project**.
3. Select **Console Application → C Project**.
4. Write your code in the editor.
5. Click **Compile & Run** to execute the program.

Dev-C++ includes a compiler, so you don't need to install GCC separately.

2. Code::Blocks (Windows/Linux/macOS)

Steps:

1. Download **Code::Blocks (with ming w setup)** so GCC is included.
2. Install the software.
3. Open Code::Blocks and create a new project using:
   File → New → Project → Console Application (C language).
4. Write your C code in the editor.
5. Use **Build → Build and Run** to compile and execute.

3. Visual Studio Code (VS Code) — Requires GCC Installed

Steps:

1. Install VS Code from the official site.
2. Install the GCC compiler using MinGW or the steps above.
3. Open VS Code and install extensions:
    - **C/C++ (Microsoft)**
    - **Code Runner** (optional)
4. Create a folder for your C programs and open it in VS Code.
5. Create a new file with .c extension (e.g., main.c).
6. Write your code and compile it manually using the terminal:
7.

3. Basic Structure of a C Program

- **THEORY EXERCISE**:

  Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.

→ **Headers:** Headers are files that contain declarations of functions and macros used in a program. They are included at the top of the file using the #include directive.

Main Function: Every C program must have a main() function. It is the starting point of program execution.

Comments: Comments are non-executable text used to explain code.
There are two types:

Single-line comment: // This is a single-line comment

Multi-line comment: /* This is

a multi-line

comment */

Data Types: Data types specify the type of data a variable can hold.
Common built-in data types in C:

Data Type	Description	Example
int	integers	5, 120
float	decimal numbers	3.14
double	larger decimals	12.34567
char	single character	'A', 'z'

Variables: Variables are named storage locations used to store data values.
A variable must be declared before use.

4. Operators in C

- **THEORY EXERCISE:**

  Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.

→ Operators in C are special symbols used to perform operations on variables and values. C provides several types of operators, each serving a different purpose.

1. Arithmetic Operators : Arithmetic operators are used to perform mathematical calculations.

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

2. Relational Operators : Relational operators are used to compare two values. They return true (1) or false (0).

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

3. Logical Operators : Logical operators are used to combine conditions, mainly in decision-making statements.

Operator	Meaning
&&	Logical AND
`	
!	Logical NOT

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

Operator	Description
=	Assign
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulus and assign

5. Increment and Decrement Operators : These operators increase or decrease a variable's value by 1.

Operator	Description
++	Increment
--	Decrement

Types:

- Pre-increment: ++a
- Post-increment: a++

6. Bitwise Operators : Bitwise operators work on binary representations of integers.

Operator	Meaning
&	Bitwise AND
`	`
^	Bitwise XOR
~	Bitwise NOT
<<	Left shift
>>	Right shift

5. Control Flow Statements in C

- **THEORY EXERCISE**: Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.

→ Decision-making statements in C are used to control the flow of program execution based on conditions. These statements evaluate expressions and execute different blocks of code depending on whether the condition is true or false.

1. if Statement : The if statement executes a block of code only when the given condition is true.

2. if–else Statement : The if–else statement executes one block if the condition is true, and another block if the condition is false.

3. Nested if–else Statement : A nested if–else is an if–else statement inside another if or else block. It is used to check multiple conditions.

4. switch Statement : The switch statement is used to execute one block of code among many choices based on the value of a variable or expression.

6. Looping in C

- **THEORY EXERCISE**: Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.

→ Loops are used to execute a block of code repeatedly as long as a given condition is true. C mainly provides three types of loops:

- while loop
- for loop
- do-while loop

1. while Loop

The while loop checks the condition before executing the loop body. If the condition is false initially, the loop may not execute at all.

When to Use:

- When the number of iterations is not known in advance
- When looping depends on user input or a condition
- Example: Reading input until a specific value is entered

2. for Loop

The for loop is used when the number of iterations is known. Initialization, condition, and update are written in a single line.

When to Use:

- When the loop runs a fixed number of times
- Best for counting loops
- Example: Printing numbers, traversing arrays

3. do-while Loop

The do-while loop executes the loop body at least once, because the condition is checked after execution.

When to Use:

- When the loop must run at least once
- Common in menu-driven programs
- Example: Displaying a menu until the user chooses exit

7. Loop Control Statements

- **THEORY EXERCISE:** Explain the use of break, continue, and go-to statements in C. Provide examples of each.

→ **Jump Statements in C**

Jump statements are used to alter the normal sequential flow of a program by transferring control to another part of the program.

1. break Statement

The break statement is used to immediately terminate the execution of a loop or a switch statement. When a break is encountered, control is transferred to the statement that follows the loop or switch. It is commonly used to exit from loops early or to prevent fall-through in switch cases.

2. continue Statement

The continue statement is used to skip the remaining statements of the current iteration of a loop and proceed directly to the next iteration. Unlike break, it does not terminate the loop; it only skips the current cycle and continues with the loop condition check.

3**. go-to Statement**

The go-to statement transfers program control unconditionally to a labelled, statement within the same function. It can be used to jump out of deeply nested structures or for error handling. However, excessive use of go-to makes programs difficult to read, understand, and maintain, so its use is generally discouraged.

8. Functions in C •

- **THEORY EXERCISE:** What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.

→ **Functions in C**

A function in C is a block of code designed to perform a specific task. Functions help break a large program into smaller, manageable units, making the program modular, reusable, and easy to understand.

Function Declaration (Prototype)

A function declaration informs the compiler about:

- Function name
- Return type
- Parameters

Function Definition

A function definition contains the actual code that performs the task. It specifies how the function works.

Function Call

A function call is used to execute the function. Control is transferred to the called function, and after execution, control returns to the calling function.

Complete Example Program

```
#include <stdio.h>


// Function declaration
int add(int, int);


// Function definition
int add(int a, int b)
{
   return a + b;
```

```
}


// Main function (function call)
int main()
{
   int result = add(10, 20);
   printf("Result = %d", result);
   return 0;
}
```

9. Arrays in C

- **THEORY EXERCISE:** Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples

→ **Arrays in C** : An array in C is a collection of elements of the same data type stored in contiguous memory locations. Each element can be accessed using a common name and an index (subscript). Arrays are useful for storing and managing multiple values efficiently using a single variable name.

Key Features of Arrays

- All elements are of the same data type
- Stored in continuous memory
- Index starts from 0
- Allows easy access and manipulation of data

One-Dimensional Array

A one-dimensional array stores data in a single linear list. It is similar to a row of elements.

Uses:

- Storing lists (marks, salaries, ages)
- Simple data processing

Multi-Dimensional Array

A multi-dimensional array contains more than one dimension. The most common type is the two-dimensional array, which is similar to a table with rows and columns.

Uses:

- Matrices
- Tables
- Storing structured data like marks of students in multiple subjects

---

Difference Between One-Dimensional and Multi-Dimensional Arrays

Feature	One-Dimensional Array	Multi-Dimensional Array
Structure	Linear list	Table or matrix
Dimensions	One	Two or more
Indexing	Single index	Multiple indices
Example	a[5]	a[3][4]
Memory representation	Single row	Rows and columns

10. Pointers in C

- **THEORY EXERCISE:** Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

→ **Pointers in C**

A pointer in C is a variable that stores the memory address of another variable instead of storing a direct value. Pointers allow programs to access and manipulate data directly in memory.

Declaration of Pointers

A pointer is declared by using the asterisk (*) before the pointer name. The pointer's data type must match the type of variable whose address it will store.

Syntax:

data_type *pointer_name;

Example:

int *ptr;

Initialization of Pointers

A pointer is initialized by assigning it the address of a variable, using the address-of operator (&).

Syntax:

pointer_name = &variable_name;

Example:

int num = 10;

int *ptr = #

- ptr stores the address of num
- *ptr gives the value stored at that address (10)

Dereferencing a Pointer

Dereferencing means accessing the value stored at the memory location pointed to by the pointer, using the * operator.

printf("%d", *ptr);

Importance of Pointers in C

Pointers are important in C for the following reasons:

1. Efficient memory management – Pointers allow direct access to memory.
2. Function call by reference – Enables functions to modify actual variables.
3. Dynamic memory allocation – Used with malloc(), calloc(), and free().
4. Array and string handling – Arrays are accessed using pointers internally.
5. Data structures – Essential for linked lists, stacks, queues, trees, etc.
6. Improved performance – Reduces copying of large data.

11. Strings in C

- **THEORY EXERCISE:** Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.

→ **Strings in C**

In C, a string is a sequence of characters stored in a character array and terminated by a null character ('\0'). C provides several string handling functions in the <string.h> header to perform common operations on strings.

1. strlen()

strlen() is used to find the length of a string, excluding the null character.

Syntax :

strlen(string);

When It Is Useful

- Checking the length of user input
- Validating passwords or usernames
- Looping through characters in a string

2. strcpy()

strcpy() is used to copy one string into another.

Syntax :

strcpy(destination, source);

When It Is Useful

- Copying user input into another variable
- Assigning one string value to another
- Storing temporary or backup strings

3. strcat()

strcat() is used to append (join) one string to the end of another.

Syntax :

strcat(destination, source);

When It Is Useful

- Joining first name and last name
- Creating full file paths or messages
- Combining multiple strings

4. strcmp()

strcmp() is used to compare two strings.

Syntax :

strcmp(string1, string2);

Return Values

- 0 → Strings are equal
- < 0 → First string is smaller
- > 0 → First string is greater

When It Is Useful

- Checking login credentials
- Comparing user input with predefined values
- Sorting strings alphabetically

5. strchr()

strchr() is used to find the first occurrence of a character in a string.

Syntax :

strchr(string, character);

When It Is Useful

- Searching for a specific character
- Validating email addresses (@)
- Parsing text data

12. Structures in C

- **THEORY EXERCISE:** Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.

→ **Structures in C**

A structure in C is a user-defined data type that allows grouping of variables of different data types under a single name. Structures are used to represent real-world entities such as a student, employee, or book, where each entity has multiple attributes of different types.

Declaration of a Structure

Structure declaration defines the structure type and its members. It does not allocate memory.

Declaring Structure Variables

After defining a structure, variables of that structure type can be declared.

struct Student s1;

Initialization of a Structure

A structure can be initialized at the time of variable declaration by providing values in the same order as members.

13. File Handling in C

- **THEORY EXERCISE:** Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.

→ **File Handling in C**

File handling in C allows a program to store data permanently on secondary storage (such as hard disks). Unlike variables, which lose data when the program ends, files help retain data for future use.

C provides file handling support through functions defined in the <stdio.h> header.

Importance of File Handling in C

File handling is important because it:

1. Enables permanent storage of data
2. Allows programs to read and write large amounts of data

3. Helps in data sharing between programs
4. Is essential for real-world applications like databases, logs, and reports
5. Reduces dependency on user input every time the program runs

File Operations in C

1. Opening a File

To perform any file operation, a file must be opened using the fopen() function.

Syntax:

```
FILE *fp;
fp = fopen("filename", "mode");
```

2. Closing a File

After completing file operations, the file should be closed using fclose() to free resources.

Syntax:

```
fclose(fp);
```

3. Writing to a File

Data can be written to a file using functions like fprintf(), fputs(), or fputc().

```
FILE *fp;
fp = fopen("data.txt", "w");
fprintf(fp, "Welcome to C programming");
fclose(fp);
```

4. Reading from a File

Data can be read from a file using functions like fscanf(), fgets(), or fgetc().

```
FILE *fp;
char str[50];
fp = fopen("data.txt", "r");
fgets(str, 50, fp);
printf("%s", str);fclose(fp);
```