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.

Ans: C was developed by Dennis Ritchie in 1972 to rewrite the UNIX operating system. It provided low-level access and portability, making it ideal for system programming. C became standardized as ANSI C in 1989, with updates like C99 and C11. It remains crucial in operating systems, embedded systems, and performance-critical applications. C is still widely taught and used due to its speed, control, and foundational role in modern programming.

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.

Ans: • Download and install a C compiler like **GCC** (via MinGW or TDM-GCC).

• Add the compiler's `bin` folder to the system **PATH**.

• Choose an IDE: **Dev C++**,**Code : :Blocks**, or **VS Code**.

• Install the IDE and ensure it detects the GCC compiler.

• Create a new C file or project, write code, and run/compile it

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.

Ans:

1. **Header files** include libraries: `#include <stdio.h>`
2. The program starts with the **main function**: `int main() { ... }`
3. **Comments** explain the code: `// single-line` or `/* multi-line */`
4. **Data types** define variable types: `int`, `float`, `char`, etc.
5. **Variables** store data: `int age = 20;`

Example:

```
#include <stdio.h>            // Header

int main() {                  // Main function
```

```
int age = 20;              // Variable with data type
printf("Age: %d", age);    // Output

return 0;
}
```

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.

Ans:

- **Arithmetic Operators**: Used for basic math – `+`, `-`, `*`, `/`, `%` (modulus).
- **Relational Operators**: Compare values – `==`, `!=`, `>`, `<`, `>=`, `<=`.
- **Logical Operators**: Combine conditions – `&&` (AND), `||` (OR), `!` (NOT).
- **Assignment Operators**: Assign values – `=`, `+=`, `-=`, `*=`, `/=`, etc.
- **Increment/Decrement**: Increase or decrease by 1 – `++`, `--`.

5. Control Flow Statements in C

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

Ans:

if – Executes code if the condition is true.

```
if (x > 0) {
printf("Positive");
 }
```

if-else – Chooses between two blocks based on condition.

```
if (x > 0) {
 printf("Positive");
}
else {
 printf("Not positive");
 }
```

nested if-else – Multiple conditions inside each other.

```
if (x > 0) {
    if (x < 10) {
 printf("Between 1 and 9");
```

```
 }
}
```

switch – Selects code block based on variable value.

```
switch (x) {
  case 1: printf("One"); break;
  case 2: printf("Two"); break;
  default: printf("error");
}
```

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.

Ans:

while Loop

- Checks condition **before** executing the loop body.
- Used when the number of iterations is **not known in advance**.
- May **never run** if the condition is false at the start.

Example:

```
int i = 0;
while (i < 5)
{
    printf("%d ", i);
    i++;
}
```

---

for Loop

- Includes initialization, condition, and increment in one line.
- Best when the number of iterations is **known**.
- Cleaner and more readable for counter-based loops.

Example:

```
for (int i = 0; i < 5; i++)
{
    printf("%d ", i);
}
```

---

do-while Loop

- Executes the loop body **at least once**, even if the condition is false.
- Condition is checked **after** the loop body.

- Useful for menus or input validation.

Example:

```
int i = 0;
do {
    printf("%d ", i);
    i++;
} while (i < 5);
```

7. Loop Control Statements

THEORY EXERCISE: Explain the use of break, continue, and goto statements in C. Provide examples of each.

Ans:

1. `break` Statement:
Used to exit a loop or `switch` statement prematurely when a condition is met.

```
for (int i = 0; i < 10; i++) {
    if (i == 5)
        break;
    printf("%d ", i);
}
```

2. `continue` Statement:
Skips the remaining code in the current loop iteration and proceeds with the next iteration.

```
for (int i = 0; i < 5; i++) {
    if (i == 2)
        continue;
    printf("%d ", i);
}
```

3. `goto` Statement:
Transfers control to a labeled statement. It should be used sparingly as it can make code harder to read.

```
int i = 0;
start:
    printf("%d ", i);
    i++;
    if (i < 3)
        goto start
```

8. Functions in C

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

Ans:

Functions in C are blocks of reusable code that perform a specific task. They help in modular programming and code reusability.

1. Function Declaration:

Tells the compiler about the function name, return type, and parameters.

```
int add(int a, int b);  // Declaration
```

2. Function Definition:

Contains the actual body/code of the function.

```
int add(int a, int b) {
    return a + b;
}
```

3. Function Call:

Used to invoke the function in the `main()` or another function

```
int result = add(3, 4);
```

9. Arrays in C

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

Ans: An **array** in C is a collection of elements of the same data type stored in **contiguous memory locations**. Arrays allow easy access and manipulation of data using **indices**.

feature	One-Dimensional Array (1D)	Multi-Dimensional Array (2D or more)
Structure	Linear (single row)	Tabular (rows and columns)
Syntax Example	`int arr[5];`	`int arr[3][4];`
Accessing Elements	`arr[0], arr[1], ..., arr[4]`	`arr[0][0], arr[1][2], ..., arr[2][3]`
Use Case	List of values (e.g., marks, ages)	Matrix/grid of values (e.g., game board, table)

feature	One-Dimensional Array (1D)	Multi-Dimensional Array (2D or more)
Memory Layout	Contiguous in one dimension	Contiguous in row-major order for each dimension

Example:

1D Array :

int `scores[3] = {10, 20, 30};`

2D Array :

int `table[2][2] = {{1, 2}, {3, 4}};`

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?

Ans: Pointers in C are variables that store the memory address of another variable. They are declared using the `*` symbol before the pointer name. For example, `int *ptr;` declares a pointer to an integer. Pointers are initialized by assigning them the address of a variable using the `&` operator, like `ptr = &x;`. They are important because they allow direct memory access, efficient array handling, dynamic memory allocation, and the ability to modify function arguments. Pointers are essential for advanced programming tasks like managing data structures (linked lists, trees) and interacting with hardware.

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.

Ans:

- **`strlen(str)`**
  - Returns the length of a string (excluding the null terminator).
  - Example:

```
strlen("hello");  // Returns 5
```

  - **Use:** Measure user input length, e.g., passwords or names.

- **`strcpy(dest, src)`**
  - Copies `src` string into `dest`.
  - Example:

```
strcpy(dest, "C programming");
```

- **Use:** Duplicate a string into another variable.

- **`strcat(dest, src)`**
  - Appends `src` string to the end of `dest`.
  - Example:

```
strcat(greeting, " World!");
```

  - **Use:** Combine first name and last name, build messages.

- **`strcmp(str1, str2)`**
  - Compares two strings. Returns 0 if equal.
  - Example:

```
strcmp("apple", "apple");  // Returns 0
```

  - **Use:** Validating user input, passwords, etc.

- **`strchr(str, ch)`**
  - Finds first occurrence of character `ch` in `str`.
  - Example:
  - strchr("hello", 'e'); // Returns pointer to 'e'

12.Structures in C

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

Ans: Structures allow grouping of variables of different data types under one name. They are used to model real-world entities like students, books, etc.

Declaration:

```
struct Student {
    int id;
    char name[20];
    float marks;
};
```

Initialization:

```
struct Student s1 = {101, "Ali", 87.5};
```

Accessing Members:

Use the dot . operator:

```
printf("%d", s1.id);        // Access id
printf("%s", s1.name);      // Access name
```

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 file

Ans:

1. **Opening a File:**

```
FILE *fp = fopen("data.txt", "r");
```

2. **Writing to a File:**

```
fprintf(fp, "Hello, World!");
```

3. **Reading from a File:**

```
fscanf(fp, "%s", buffer);
```

4. **Closing a File:**

```
fclose(fp);
```