Module 2 – Introduction to Programming

- **Overview of C Programming**

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

Ans. = **The C programming language is one of the most influential and enduring programming languages in the history of computer science. Developed in the early 1970s, C has played a pivotal role in shaping modern software development and continues to be widely used today. This essay explores the history and evolution of C, its importance, and the reasons for its continued relevance.**

- Why C is Still Used Today
    - ❖ Performance
    - ❖ Embedded Systems
    - ❖ **Operating Systems and Compilers**
    - ❖ Portability and Reliability
    - ❖ Educational Value

- **Setting Up Environment**

2. 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.**

- **Basic Structure of a C Program**

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

Ans. =

1. **Header Files**
   - **#include <stdio.h>**
2. **Main Function**
   - **int main() {**
   - **// code goes here**
   - **return 0;**
   - **}**
3. **Comments**
   - **// Single-line comment**
   - **/* Multi-line**
   - **comment */**
4. **Data Types**
   - **int → integers (whole numbers)**
   - **float → decimal numbers**
   - **char → single characters**
   - **double → large decimal numbers**
5. **Variables**
   - **int age = 20;**
   - **float salary = 3500.5;**
   - **char grade = 'A';**

- **Operators in C**

4. 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 – ++, --.**

Control Flow Statements in C

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

Ans. =

if statement

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

if-else statement

```
if (condition) {
   // code if condition is true
} else {
   // code if condition is false
}
```

Nested if-else statement

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

- **switch statement**

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

- **Looping in C**

6. 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.
    - Condition is checked after the loop body.

Example:

```
 int i = 0;

do {

printf("%d ", i);

i++;

}

while (i < 5);
```

- **Loop Control Statements**

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

Ans. =

- 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);
 }
```

- ## 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);

}
```

- ## 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);
++;
if (i < 3)
goto start
```

- ## Functions in C

8. 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.

- ### Function Declaration:

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

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

- ### Function Definition:

Contains the actual body/code of the function.

```
int add(int a, int b) {

        return a + b;

}
```

- ### Function Call:

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

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

- **Arrays in C**

9. 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.

- Each element in an array can be accessed using an index**.**
- Array indexing in C starts from **0**.
- Arrays help store multiple values under a single variable name instead of declaring multiple variables.

Feature	One-Dimensional Array	Multi-Dimensional Array
Structure	Linear list of elements	Table-like structure (rows & columns)
Syntax	int arr[5];	int arr[2][3];
Access Method	arr[index]	arr[row][column]
Example	marks[3] gives 4th element	matrix[1][2] gives element at 2nd row, 3rd column
Visualization	10 20 30 40 50	10 20 30 40 50 60

- **Pointers in C**

10. 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.

- **Strings in C**

11. 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, "java 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'
```

- ## Structures in C

12. 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
```

- ## File Handling in C

13. Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing file

14. Ans. =

- **Opening a File:**
  FILE *fp = fopen("data.txt", "r");
- **Writing to a File:**
  fprintf(fp, "Hello, World!");
- **Reading from a File:**
  fscanf(fp, "%s", buffer);
- **Closing a File:**
  fclose(fp);