Module 2 – Introduction to Programming – THEORY EXERCISE

---

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

   Ans. C programming language was developed in 1972 by Dennis Ritchie at Bell Labs. It was created for developing the UNIX operating system and is known for its efficiency and control. C is a procedural language that supports structured programming. Its popularity lies in its ability to provide low-level memory access and fast performance. Even today, C is widely used in embedded systems, operating systems, and compilers because of its simplicity, portability, and ability to interact with hardware directly. It has influenced many modern languages like C++, Java, and Python.

---

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

   Ans. To install and set up a C programming environment:

1. **Install Compiler**: Download and install GCC (MinGW for Windows).
2. **Choose IDE**: Download an IDE such as DevC++, Code::Blocks, or Visual Studio Code.
3. **Configure Path**: Set the environment path variable for GCC in system settings.
4. **Write Program**: Open the IDE and write your first program (e.g., "Hello World").
5. **Compile and Run**: Use the compile and run buttons or terminal commands to test the setup.

3. **Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.**
   Ans.
   A C program is made up of:

- **Header files** (e.g., #include<stdio.h>) – to include standard functions.
- **Main function** – Entry point of the program.
- **Comments** – To describe code (e.g., // This is a comment)
- **Variables** – Used to store data (e.g., int a = 5;)
- **Data Types** – Such as int, float, char

Example:

```
#include<stdio.h>
int main() {
   // declaring variables
   int num = 10;
   float price = 12.5;
   char grade = 'A';
   printf("%d %.2f %c", num, price, grade);
   return 0;
}
```

---

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 mathematical operations.
- **Relational Operators**: ==,!=, >, <, >=, <=
  Used to compare values.
- **Logical Operators**: &&, ||, !
  Used in conditional expressions.
- **Assignment Operators**: =, +=, -=, etc.
  Used to assign values.
- **Increment/Decrement Operators**: ++, --
  Increase or decrease value by 1.
- **Bitwise Operators**: &, |, ^, ~, <<, >>
  Used for bit-level operations.
- **Conditional Operator**: condition ? true: false
  Short form of if-else.

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

- **if**: Checks a condition and runs code if it's true.
- **else**: Runs code if the if condition is false.
- **nested if-else**: if-else inside another if or else block.
- **switch**: Selects code to run based on the value of a variable.

Example:

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

---

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

- **for loop**: Used when the number of iterations is known.

```
for(int i = 0; i < 5; i++) { }
```

- **while loop**: Checks condition before execution, used when iterations are unknown.

while(condition) { }

- **do-while loop**: Executes at least once, checks condition after execution.

do { } while(condition);

---

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

- **break**: Exits loop or switch early.
- **continue**: Skips current loop iteration.
- **goto**: Jumps to a labeled section of code (not recommended).

Example:

```
for(int i=1; i<=5; i++) {
   if(i == 3) continue;
   if(i == 5) break;
   printf("%d ", i);
}
```

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

Ans.
A function is a block of code designed to perform a specific task.

- **Declaration**: Tells the compiler about the function's name and return type.
- **Definition**: The actual code.
- **Call**: Executes the function.

Example:

```
int add(int a, int b); // Declaration
int add(int a, int b) { return a + b; } // Definition
int result = add(5, 3); // Call
```

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

Ans .
An array is a collection of variables of the same data type stored in contiguous memory.

- **1D Array**: int a[5] = {1, 2, 3, 4, 5};
- **2D Array**: int b[2][2] = {{1,2}, {3,4}};

Use: Arrays make it easier to store and process multiple values using loops.

---

10. Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

Ans.
Pointers store the memory address of another variable.

Declaration:

int a = 10;

int *ptr = &a;

Importance:

- Dynamic memory allocation
- Function argument passing by reference
- Efficient array and string handling

---

11. Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.

Ans.

- **strlen()**: Returns the length of a string.
- **strcpy()**: Copies one string to another.
- **strcat()**: Appends one string to another.
- **strcmp()**: Compares two strings.
- **strchr()**: Finds first occurrence of a character in a string.

Example:

char str1[20] = "Hello";

char str2[10] = "World";

strcat(str1, str2); // str1 becomes "HelloWorld"

---

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

Ans.
A structure groups variables of different types.

Declaration:

struct student {

  char name[20];

```
    int roll;
    float marks;
};
```

Initialization and access:

```
struct student s1 = {"John", 101, 85.5};
printf("%s %d %.2f", s1.name, s1.roll, s1.marks);
```

---

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

Ans.
File handling allows reading/writing data from/to files. Useful for storing data persistently.

Common functions:

- fopen(), fclose() – open and close files.
- fprintf(), fscanf() – write/read formatted data.
- fread(), fwrite() – binary file handling.

Example:

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