Module 2 – Introduction to Programming

1. Overview of C Programming

• THEORY EXERCISE:

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

Answer:

C programming was developed in the early 1970s by Dennis Ritchie at Bell Labs. It evolved from earlier languages like B and BCPL. C was designed to provide low-level access to memory, offer simple language constructs, and support structured programming.

C became widely popular after being used to rewrite the UNIX operating system, making it one of the first operating systems written in a high-level language. Over the decades, it became the foundation for many modern programming languages like C++, C#, Java, and even Python to some extent.

Its importance today lies in its speed, portability, and control over system-level resources. C is still used extensively in embedded systems, operating systems (like Linux), game engines, and IoT devices due to its performance and efficiency.

2. Setting Up Environment

THEORY EXERCISE:

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

Answer:

1. Download Compiler:

o For Windows: Download and install MinGW or TDM-GCC for the GCC compiler.

o For Linux: Use terminal command sudo apt install build-essential.

2. Choose and Install IDE:

o Download and install an IDE like Code::Blocks, DevC++, or VS Code.

3. Configure the IDE:

o For Code::Blocks: Ensure it detects the GCC compiler during installation.

o For VS Code: Install the C/C++ extension by Microsoft and configure the tasks.json and launch.json for build and run settings.

4. Test Setup:

o Write a simple C program (Hello World) and compile/run it to check everything is set up correctly.

3. Basic Structure of a C Program

THEORY EXERCISE:

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

Answer:

A basic C program structure includes:

#include <stdio.h>

#include<conio.h>

// This is a single-line comment

Void main() {

int a = 5;

float b = 3.14;

printf("Value of a: %d, b: %.2f\n", a, b);

getch();

}

4. Operators in C

THEORY EXERCISE:

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

Answer:

• Arithmetic Operators: +, -, \*, /, % — used for mathematical operations.

• Relational Operators: ==, !=, <, >, <=, >= — compare values.

• Logical Operators: && (AND), || (OR), ! (NOT) — used in conditional expressions.

• Assignment Operators: =, +=, -=, \*=, /=, %= — assign values to variables.

• Increment/Decrement: ++, -- — increase/decrease value by 1.

• Bitwise Operators: &, |, ^, ~, <<, >> — perform bit-level operations.

• Conditional (Ternary) Operator: condition ? true\_value : false\_value;

Example:

int a = 10, b = 20;

int max = (a > b) ? a : b;

5. Control Flow Statements in C

THEORY EXERCISE:

Q: Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples.

Answer:

• if Statement:

if (a > b) {

printf("a is greater");

}

• if-else Statement:

if (a > b) {

printf("a is greater");

} else {

printf("b is greater");

}

• Nested if-else:

if (a > b) {

if (a > c)

printf("a is greatest");

else

printf("c is greatest");

}

• switch Statement:

int choice = 2;

switch (choice) {

case 1: printf("Option 1"); break;

case 2: printf("Option 2"); break;

default: printf("Invalid");

}

6. Looping in C

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

Answer:

• while loop: Checks condition before executing. Best when the number of iterations is unknown in advance.

while (i < 10) {

}

• for loop: Used when the number of iterations is known. Initialization, condition, and increment are in one line.

for (int i = 0; i < 10; i++) {

}

• do-while loop: Executes the loop body at least once. Condition is checked after execution.

do {

} while (i < 10);

Loop Type Condition Checked Best Use Case

While Before Unknown iterations

For Before Known, fixed number of iterations

do-while After At least one guaranteed execution

7. Loop Control Statements

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

Answer:

• break: Immediately exits the loop or switch.

for (int i = 0; i < 10; i++) {

if (i == 5) break;

}

• continue: Skips the rest of the current loop iteration.

for (int i = 0; i < 10; i++) {

if (i == 3) continue;

printf("%d\n", i);

}

• goto: Jumps to a labeled statement.

goto label;

...

label:

printf("Jumped here");

8. Functions in C

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

Answer:

A function is a block of code that performs a specific task and can be reused.

1. Declaration (also called prototype):

int add(int, int);

2. Definition:

int add(int a, int b) {

return a + b;

}

3. Call:

int result = add(3, 4);

Functions improve code readability, modularity, and reusability.

9. Arrays in C

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

Answer:

An array is a collection of elements of the same data type stored in contiguous memory locations.

• One-Dimensional Array:

int arr[5] = {1, 2, 3, 4, 5};

• Multi-Dimensional Array (2D):

•

int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};

Difference:

Feature 1D Array 2D Array

Declaration int arr[5]; int matrix[2][3];

Access arr[2] matrix[1][2]

Use Case List of items Tabular data (matrix, table)

10. Pointers in C

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

Answer:

A pointer is a variable that stores the memory address of another variable.

Declaration and Initialization:

int x = 10;

int \*ptr = &x; // ptr holds the address of x

Why Important:

• Allow dynamic memory allocation

• Used in arrays and strings

• Required for function arguments by reference

• Enable efficient handling of data structures (e.g., linked lists)

Example:

printf("Value: %d, Address: %p", \*ptr, ptr);

11. Strings in C

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

Answer:

Function Description & Example

strlen() Returns the length of a string.

strlen("hello") → 5

strcpy() Copies one string into another.

strcpy(dest, src);

strcat() Appends one string to another.

strcat(str1, str2);

strcmp() Compares two strings. Returns 0 if equal.

strcmp("abc", "abc") → 0

strchr() Finds the first occurrence of a character.

strchr("hello", 'e') → pointer to 'e'

These are used for basic string manipulation in C (e.g., user input processing, string formatting, and searching).

12. Structures in C

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

Answer:

A structure in C is a user-defined data type that groups variables of different types under one name.

Declaration:

struct Student {

int roll;

char name[20];

float marks;

};

Initialization:

struct Student s1 = {1, "Rahul", 85.5};

Access:

printf("%s", s1.name);

Structures are useful when dealing with grouped data, such as storing student records, employee info, etc.

13. File Handling in C

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

Answer:

File handling allows a program to read from and write to files stored on a disk, enabling permanent data storage.

Operations:

1.Opening:

FILE \*fp = fopen("file.txt", "w");

2.Writing:

fprintf(fp, "Hello World");

3.Reading:

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

4.Closing:

fclose(fp);

File handling is used in data processing, report generation, logging, etc.