**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.

:- C was developed by **Dennis Ritchie** at **Bell Labs** in **1972** as an improvement over the B programming language, which itself was derived from BCPL. The need for C arose during the development of the Unix operating system. Unix was initially written in assembly language, but the developers wanted a more portable and efficient language that could still access low-level system components. C met this requirement.

C is one of the most influential and widely used programming languages in the history of computing. Developed in the early 1970s, it revolutionized software development with its balance of low-level functionality and high-level abstraction. Today, despite the emergence of modern programming languages, C continues to be foundational in systems programming, embedded systems, and education.

**Importance of C Programming**

* **Efficiency**: C programs execute quickly and use minimal system resources.
* **Hardware Interaction**: Ideal for programming microcontrollers and embedded systems.
* **Stability**: Mature and stable with a vast ecosystem and community.
* **Legacy Code**: A large amount of legacy code exists in C, especially in infrastructure and industrial systems.
* **Compatibility**: Acts as an intermediate language between hardware and higher-level applications.

1. Research and provide three real-world applications where C programming is

Extensively used, such as in embedded systems, operating systems, or game

Development.

: - **Embedded Systems**: - Example: Automotive Control Systems (e.g., Airbag Control, Engine **Management)**

**Operating Systems: -** Example: Linux Kernel

**Game Development: -** Doom (1993) and Quake (1996) Game Engines

**Setting Up Environment**

1. 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

**:- Download Dev-C++:**

* + Go to: <https://sourceforge.net/projects/orwelldevcpp/>

**Install and Run**:

* + Follow the installation steps.
  + GCC is already bundled with Dev-C++, so no separate setup needed.
  + Open Dev-C++, click **File > New > Source File** to start coding.
* **Dev-C++**: Lightweight and beginner-friendly.

**Basic Structure of a C Program**

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

* **Header Files**
* These are used to include libraries that provide built-in functions.
* Declared at the **top** of the program using #include.
* For example #include <stdio.h> // Standard Input Output header
* **Main Function**
* This is the **entry point** of every C program.
* Syntax: int main() { ... }
* It usually returns an integer to indicate success or failure.
* For Examle :-

int main() {

return 0;

}

* **Comments**
* Used to explain code. Ignored by the compiler.
* **Single-line:** // comment here
* **Multi-line:** /\*

// - This is a single-line comment

* **Data Types**
* Tell the compiler what kind of data a variable will store.

|  |  |  |
| --- | --- | --- |
| * int | * Integer numbers | * 10 |

|  |  |  |
| --- | --- | --- |
| * float | * Decimal numbers | * 3.14 |

|  |  |  |
| --- | --- | --- |
| * char | * Single characters | * 'A' |

|  |  |  |
| --- | --- | --- |
| * double | * Double-precision float | * 3.14159 |

* **Variables**
* Containers used to store data in memory.
* Declared using: data\_type variable\_name;
* For example :-

int age = 25;

float height = 5.9;

char grade = 'A';

**Operators in C**

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

## :- **Arithmetic Operators**

Used to perform basic mathematical operations.

## **Relational Operators**

Used to compare two values; result is either **true (1)** or **false (0)**.

## **Logical Operators**

Used to combine multiple conditions.

## **Assignment Operators**

Used to assign values to variables.

## **Increment/Decrement Operators**

Used to increase or decrease the value of a variable by 1.

## **Bitwise Operators**

Used for **bit-level operations** on integers.

## **Conditional (Ternary) Operator**

A shorthand for if-else.

| **Type** | **Common Operators** |
| --- | --- |

|  |  |
| --- | --- |
| Arithmetic | +, -, \*, /, % |

|  |  |
| --- | --- |
| Relational | ==, !=, >, <, >=, <= |

|  |  |
| --- | --- |
| Logical | &&, ` |

|  |  |
| --- | --- |
| Assignment | =, +=, -=, \*=, /=, %= |

|  |  |
| --- | --- |
| Increment/Decrement | ++, -- |

|  |  |
| --- | --- |
| Bitwise | &, ` |

|  |  |
| --- | --- |
| Conditional | ?: |

**Control Flow Statements in C**

* Explain decision-making statements in C (if, else, nested if-else, switch)

Provide examples of each

## IF **Statement**

Used to execute a block of code only **if a condition is true**.

int num = 10;

if (num > 0) {

printf("Number is positive.\n");

}

## if-else **Statement**

Executes one block if the condition is true, another if it's false.

int num = -5;

if (num >= 0) {

printf("Positive number\n");

} else {

printf("Negative number\n"); }

## **Nested** if-else **Statement**

if-else statements **inside another if or else block** for multiple conditions.

int num = 0;

if (num > 0) {

printf("Positive\n");

} else {

if (num < 0) {

printf("Negative\n");

} else {

printf("Zero\n");

}

}

## else if **Ladder**

Used for checking **multiple conditions** in a cleaner way than nested if-else.

int marks = 75;

if (marks >= 90) {

printf("Grade A\n");

} else if (marks >= 75) {

printf("Grade B\n");

} else {

printf("Grade C\n");

}

## switch **Statement**

Used when you want to **compare a variable to many constant values**.

int day = 3;

switch (day) {

case 1: printf("Monday\n"); break;

case 2: printf("Tuesday\n"); break;

case 3: printf("Wednesday\n"); break;

default: printf("Invalid day\n");

}

**Looping in C**

* Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.
* **While Loop**
* Entry-controlled loop.
* **Condition is checked before** executing the loop body.
* The loop body **may not run** if the condition is false initially.
* **For Loop**
* Entry-controlled loop.
* All loop control statements (unit, test, update) are in one line.
* Best when the **number of iterations is known**.
* **do-while Loop**
* **Exit-controlled loop** (condition is checked **after** the loop body).
* The loop body **executes at least once**, even if the condition is false.

**Loop Control Statements**

* **Explain the use of break, continue, and goto statements in C. Provide**

**examples of each.**

## break Statement

### 🔹 ****Purpose:****

* Used to **exit** a loop (for, while, do-while) or a switch statement **immediately**, even if the condition is still true.

### ✅ ****Common Uses:****

* Exit loop on a specific condition.
* End a switch case.

## continue Statement

### 🔹 ****Purpose:****

* **Skips** the current iteration of the loop and **continues** with the next iteration.
* Useful when you want to skip specific values without exiting the loop.

## goto Statement

### 🔹 ****Purpose:****

* Transfers program control **to a labeled statement**.
* Generally **discouraged** due to risk of creating unreadable ("spaghetti") code.

**Functions in C**

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

:- 1)Function defination : Function is a block of code which only Runs when it is called.

2)Function Call

There are two type

1) Built in : printf(); , scanf();

2) user defined -> there are 4 types in user define

1) Function without parameters and without return types

default function

2) Function with parameter without return type:- Para ;user input

3) Function without Parameters with return type

4) Function With parameters and returns types

**Arrays in C**

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

:- Array is a collection of similar data type like int,float,char etc.

There are Two type of Array

* 1. **One dimension array :-**

A **linear collection** of elements (like a list).

For example :- int a[5] = {10, 20, 30, 40, 50};

* 1. **Two dimensions array :-**

An array with **more than one dimension**, like a table or matrix.

For Example :- int matrix[2][3] = { {1, 2, 3},{4, 5, 6} };

**Pointers in C**

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

:- A **pointer** is a **variable that stores the memory address** of another variable.  
Instead of holding a value like 10, a pointer holds a **location in memory** where a value is stored.

| **Reason** | **Explanation** |
| --- | --- |
| Direct memory access | Enables low-level memory manipulation. |
| Efficient function arguments | Enables passing large data (arrays, structs) by reference, not by value. |
| Dynamic memory allocation | Required for functions like malloc(), calloc() in stdlib.h. |
| Data structures | Essential for creating linked lists, trees, graphs, etc. |
| Array and string manipulation | Pointers and arrays are closely related; strings are handled via pointers. |

### Declaration:

### data\_type \*pointer\_name;

### 🔹 Initialization:

### int a = 10;

### int \*ptr = &a; // ptr stores the address of variable 'a'

### Strings in C

* Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr().

| **Function** | **Use** |
| --- | --- |
| strlen() | Get length of string |
| strcpy() | Copy one string to another |
| strcat() | Append one string to another |
| strcmp() | Compare two strings |
| strchr() | Find the first occurrence of a character |

### Structures in C

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

:- A **structure** in C is a **user-defined data type** that allows grouping **different types of variables** under a single name.  
It is used to represent a **record** (like a student, employee, etc.).

## **🔹 Why Use Structures?**

* To combine different data types (e.g., int, char[], float) into a single logical unit.
* Useful for organizing complex data in a meaningful way.
* Common in building **databases**, **linked lists**, **file records**, etc.

| **Component** | **Syntax/Use** |
| --- | --- |
| Declare struct | struct Student { ... }; |
| Create variable | struct Student s1; |
| Access members | s1.id, s1.name, s1.marks |
| Initialize struct | struct Student s = {101, "John", 90} |

## 

**File Handling in C**

* **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 allows programs to **store, retrieve, and manipulate data** permanently on disk (e.g., .txt, .dat files), instead of just temporarily in memory.

**Why File Handling Is Important:**

| **Reason** | **Explanation** |
| --- | --- |
| **Permanent Storage** | Data is preserved after the program ends |
| **Large Data Management** | Handles more data than memory-limited variables |
| **Input/Output Efficiency** | Reads from or writes to files instead of manual input |
| **Data Sharing** | Files allow data to be shared between programs or systems |

**How Perform File Operation :-**

| **Operation** | **Function** | **Purpose** |
| --- | --- | --- |
| Open | fopen() | Opens a file |
| Close | fclose() | Closes the file |
| Write | fprintf()/fputs()/fputc() | Writes to file |
| Read | fscanf()/fgets()/fgetc() | Reads from file |