Module 2 – Introduction to Programming

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

C programming is one of the most important and widely used programming languages in computer science. It was developed in **1972** by **Dennis Ritchie** at **Bell Laboratories**. The main purpose of C was to create the **UNIX operating system**. It was derived from the **B language**, which itself came from **BCPL**. The goal was to design a structured and efficient programming language that could offer **low-level memory access** and also replace assembly language for systems programming.

Evolution of C

Over the years, C has been improved and standardized several times:

- **1978**: First edition of *"The C Programming Language"* by Kernighan and Ritchie (also called K&R C).
- **1989**: ANSI C (C89) was standardized by the **American National Standards Institute**.
- **1999**: C99 introduced new features such as inline functions and new data types.
- **2011**: C11 added support for multi-threading and better Unicode handling.
- **2017 & 2023**: C17 and C23 refined the language with bug fixes and minor improvements.

Importance and Continued Use

C has remained important for decades because of the following reasons:

- **Performance and efficiency** – Programs in C run very fast.
- **Low-level access** – It allows direct control of memory and hardware.
- **Portability** – C programs can run on different platforms with little or no modification.

- **Foundation for other languages** – Many modern languages like **C++**, **Java**, and **Python** are built on the concepts of C.
- **Wide usage** – C is still used in **embedded systems, operating systems, compilers, and system-level programming**.

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

Installing GCC (via MinGW on Windows)

1. Download **MinGW** from the official website.
2. Run the installer and select:
    - gcc-g++
    - binutils
    - mingw32-base
3. Add the path to MinGW's **bin folder** (e.g., C:\MinGW\bin) to the system **PATH environment variable**.

Setting Up IDEs

DevC++

1. Download and install **DevC++**.
2. Create a new project or source file.
3. Write and compile C code.

Visual Studio Code (VS Code)

1. Install **Visual Studio Code**.
2. Install the **"C/C++" extension by Microsoft**.
3. Set up **tasks.json** and **launch.json** for build and debug configuration.

Code::Blocks

1. Download the version of **Code::Blocks** that includes the compiler.
2. Install and open Code::Blocks.
3. Create a new project and write C code.

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

```
#include <stdio.h>    // Header file
// This program demonstrates the structure of C
int main() {
   int age = 20;          // integer variable
   float marks = 89.5;     // float variable
   char grade = 'A';       // character variable
   printf("Age: %d\n", age);
   printf("Marks: %.2f\n", marks);
   printf("Grade: %c\n", grade);
   return 0;
}
```

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

1. Arithmetic Operators

- Used for mathematical calculations.
- Operators: + , - , * , / , %
- Example:
- int a = 10, b = 3;
- printf("%d", a + b);   // Output: 13

2. Relational Operators

- Used to compare two values.
- Operators: == , != , > , < , >= , <=
- Example:
- int x = 5, y = 10;
- printf("%d", x < y); // Output: 1 (true)

3. Logical Operators

- Used for combining conditions.
- Operators: && , || , !
- Example:
- int a = 5, b = 10;
- printf("%d", (a < b) && (b > 0)); // Output: 1

4. Assignment Operators

- Used to assign values.
- Operators: = , += , -= , *= , /=
- Example:
- int num = 10;
- num += 5; // num = num + 5
- printf("%d", num); // Output: 15

5. Increment and Decrement Operators

- Used to increase or decrease value by 1.
- Operators: ++ , --
- Example:
- int x = 5;
- x++; // x = 6

- printf("%d", x);

6. Bitwise Operators

- Work at the bit level.
- Operators: & , | , ^ , ~ , << , >>
- Example:
- int a = 5, b = 3; // Binary: 5=0101, 3=0011
- printf("%d", a & b); // Output: 1

7. Conditional (Ternary) Operator

- Short form of if-else.
- Syntax: condition ? value_if_true : value_if_false;
- Example:
- int age = 18;
- printf("%s", (age >= 18) ? "Adult" : "Minor");
- // Output: Adult

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

1. if Statement

- Executes a block of code **only if** the condition is true.
- Syntax:
- if (condition) {
- 	// code to run if condition is true
- }
- Example:
- int age = 20;
- if (age >= 18) {

- printf("You are an adult.");
- }

2. if-else Statement

- Provides two paths: one if the condition is true, another if false.
- Example:
- int num = 5;
- if (num % 2 == 0) {
- printf("Even number");
- } else {
- printf("Odd number");
- }

3. Nested if-else Statement

- An **if-else inside another if-else**.
- Used when there are multiple conditions.
- Example:
- int marks = 75;
- if (marks >= 90) {
- printf("Grade A");
- } else if (marks >= 60) {
- printf("Grade B");
- } else {
- printf("Grade C");
- }

4. switch Statement

- Used to choose one option from **multiple cases**.

- Syntax:
- switch (expression) {
- case value1:
- // code
- break;
- case value2:
- // code
- break;
- default:
- // code if no cases match
- }
- Example:
- int day = 3;
- switch (day) {
- case 1: printf("Monday"); break;
- case 2: printf("Tuesday"); break;
- case 3: printf("Wednesday"); break;
- default: printf("Invalid day");

}

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

1. While Loop

Definition:
The while loop executes a block of code repeatedly as long as the condition is true.

Syntax:

```
while(condition) {
   // code
}
```

Example:

```
int i = 1;
while(i <= 5) {
   printf("%d ", i);
   i++;
}
```

---

2. For Loop

Definition:
The for loop is used when you know exactly how many times you want to execute the loop.

Syntax:

```
for(initialization; condition; update) {
   // code
}
```

Example:

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

---

3. Do-While Loop

Definition:
The do-while loop executes the body at least once, then checks the condition.

Syntax:

```
do {
   // code
} while(condition);
```

Example:

```
int i = 1;
do {
   printf("%d ", i);
   i++;
} while(i <= 5);
```

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

1. break Statement

Used to **exit** from a loop or switch immediately.

```
for(int i=1;i<=5;i++){
   if(i==3) break;
   printf("%d ", i);
}
// Output: 1 2
```

---

2. continue Statement

Used to **skip the current iteration** and move to the next one.

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

// Output: 1 2 4 5

---

3. goto Statement

Used to **jump** to a labeled statement in the program.

```
int i=1;

start:

printf("%d ", i);

i++;

if(i<=5) goto start;

// Output: 1 2 3 4 5
```

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

Functions in C

Definition:
A function in C is a block of code that performs a specific task and can be reused multiple times.

---

1. Function Declaration (Prototype)

Tells the compiler about the function's name, return type, and parameters **before using it**.

Syntax:

```
return_type function_name(parameters);
```

Example:

```
int add(int a, int b);   // function declaration
```

---

2. Function Definition

Contains the **actual code** (body) of the function.

Syntax:

```
return_type function_name(parameters) {
   // function body
}
```

Example:

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

---

3. Function Call

Executes the function from main() or another function.

Syntax:

```
function_name(arguments);
```

Example:

```
#include <stdio.h>

int add(int a, int b);  // declaration

int main() {
   int result = add(5, 3);   // function call
   printf("Sum = %d", result);
   return 0;
}
```

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

Output:

Sum = 8

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

Arrays in C

An array is a collection of elements of the **same data type** stored in **continuous memory** and accessed using an **index**.

One-Dimensional Array

- Linear form (single row).
- Uses **one index**.

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

Multi-Dimensional Array

- Table form (rows & columns).
- Uses **two or more indices**.

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

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

1. strlen(str)

Definition: Returns the **length of a string** (number of characters).

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

```
#include <string.h>
int main() {
   char str[] = "Hello";
   printf("%d", strlen(str));  // Output: 5
   return 0;
}
```

---

2. strcpy(dest, src)

Definition: Copies one string into another.

```
char str1[20], str2[] = "World";
strcpy(str1, str2);
printf("%s", str1);  // Output: World
```

---

3. strcat(dest, src)

Definition: Concatenates (joins) two strings.

```
char str1[20] = "Hello ";
char str2[] = "World";
strcat(str1, str2);
printf("%s", str1);  // Output: Hello World
```

---

4. strcmp(str1, str2)

Definition: Compares two strings.

- Returns **0** if equal
- Returns **>0 or <0** if different

```
printf("%d", strcmp("abc", "abc"));  // Output: 0
```

printf("%d", strcmp("abc", "xyz")); // Output: negative value

---

5. strchr(str, ch)

Definition: Finds the **first occurrence of a character** in a string.

char str[] = "Hello";

printf("%s", strchr(str, 'l')); // Output:

12. Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.Structures: User-defined data types to group different data types.

Definition:
A **structure** in C is a **user-defined data type** that allows grouping of variables of **different data types** under one name.

Declaring a Structure :

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

Initializing a Structure :

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

Accessing Structure Members

Use the **dot (.) operator**.

printf("ID: %d\n", s1.id);

```
printf("Name: %s\n", s1.name);
printf("Marks: %.2f", s1.marks);
```

Example Program :

```
#include <stdio.h>
struct Student {
    int id;
    char name[20];
    float marks;
};

int main() {
    struct Student s1 = {1, "Hardik", 85.5};

    printf("ID: %d\n", s1.id);
    printf("Name: %s\n", s1.name);
    printf("Marks: %.2f\n", s1.marks);

    return 0;
}
```

Output:

```
ID: 1
Name: Hardik
Marks: 85.50
```

13. Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing filesImportance: File handling allows programs to store data permanently.

Importance of File Handling :

File handling allows programs to **store data permanently** on disk, so the data is not lost after the program ends.

Basic File Operations in C :

1. Opening a File

Use fopen() with mode (r, w, a, etc.).

```
FILE *fp;
fp = fopen("data.txt", "w");   // open file for writing
```

---

2. Writing to a File

Use fprintf() or fputs().

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

---

3. Reading from a File

Use fscanf() or fgets().

```
char str[50];
fgets(str, 50, fp);   // read string
```

---

4. Closing a File

Always close with fclose().

```
fclose(fp);
```

Example Program

```
#include <stdio.h>
int main() {
   FILE *fp;
   char str[50];

   // Write
   fp = fopen("data.txt", "w");
   fprintf(fp, "Hello World!");
   fclose(fp);

   // Read
   fp = fopen("data.txt", "r");
   fgets(str, 50, fp);
   printf("File content: %s", str);
   fclose(fp);
   return 0;
}
```

Output:

```
File content: Hello World!
```