Module 2 – Introduction to Programming

1. Overview of C 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.

→ C programming is a powerful and widely used programming language known for its simplicity and versatility. Developed in the early 1970s by Dennis Ritchie at Bell Labs, C has since become one of the most influential programming languages in the world.

→ C became famous because it was used to build the **UNIX operating system**. Before that, operating systems were written in assembly language, which was very hard. C made it easier, faster, and more powerful.

→Evolution of C: -

In **1978**, **Brian Kernighan and Dennis Ritchie** wrote a book called *The C Programming Language*. This book made C even more popular.

Later, C was given official standards to keep it the same everywhere:

- **ANSI C (1989)**
- **C99 (1999)**
- **C11 (2011)**
- **C18 (2018)**

→ Importance of C

C is called the **mother of programming languages** because many languages like **C++, Java, and Python** came from it.

C is important because:

- It is **fast** and uses **less memory**.
- It is **portable** (same code works on different computers).
- It helps programmers understand how computers really work.
- It is used in **operating systems** (Windows, Linux, etc.).
- It is used in **embedded systems** (mobiles, cars, machines).

→ Why C is Still Used

Even after 50 years, C is still used today because it is simple, powerful, and close to hardware. It is perfect for building software that needs **speed and control**.

2. Setting Up Environment

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

→ Step 1: Install GCC Compiler

- Download **MinGW** (Minimalist GNU for Windows) from its official website.
- Run the installer.
- In the setup, tick **"mingw32-gcc-g++"** (this is the C/C++ compiler).
- After installation, add MinGW's bin folder path (like C:\MinGW\bin) to **Environment Variables PATH**.

→ This step lets Windows find the compiler.

- To check if it works:
- Open **Command Prompt**
- gcc –version

→ **Step 2: Choose and Install an IDE**

Option 1: **DevC++**

- Download DevC++ from the official site.
- Install it like normal software (Next → Next → Finish).
- Open DevC++ → New Project → Console Application → Choose **C Language**.
- Write your C code → Press **F11** to run.
- DevC++ already comes with a compiler, so it's easiest for beginners.

→ Option 2: **Visual Studio Code (VS Code)**

- Download and install **VS Code**.
- Open VS Code → Go to Extensions → Install **C/C++ by Microsoft**.
- Make sure **MinGW (GCC)** is installed and added to PATH (Step 1).
- Create a new file hello.c → Write your C program.
- Open **Terminal inside VS Code** → Run commands:

→ **Easiest way (for beginners):** Use **DevC++** because it comes with a built-in compiler.

→ **Better for projects:** Use **Code: Blocks** or **VS Code** for more features.

3. Basic Structure of a C Program

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

→ **Basic Structure of a C Program**

Every C program follows a standard structure. The main parts are:

- **Header Files**
- **Main Function**
- **Comments**
- **Data Types**
- **Variables**

1) Header file: -

 These are special files that contain built-in functions (like printf for output, scanf for input).
 Declared at the top using #include.

Example: -

```
#include <stdio.h> // Standard input-output header
```

2) Main Function: -

 Every C program must have a main () function.
 Program execution starts from here.

Example: -

```
int main () {
   // Code goes here
   return 0; // End of program
}
```

3) Comments: -

- Notes for programmers, ignored by the compiler.
- Two types:
  - **Single-line:** // comment here
  - **Multi-line:** /* comment here */
- Example: -

```
#include<stdio.h>
Main()
{
// This is a single-line comment
/* This is
   a multi-line comment */
return 0;
}
```

4) **Data Type: -**
   - Data types are used to define the type of data that a variable can store.
   - Basic Data type: -
     - int → integers (10, -5, 100)
     - float → decimal numbers (3.14, -2.5)
     - char → single character ('A', 'b')
     - double → larger decimal numbers

5) **Variables: -**
   - Variable are fundamental elements used to store and manipulate data.
   - They act as named container that hold different types of values, such as integer, floating-point numbers, characters, and pointer.

     Example: -
     int age = 20; // integer variable
     float pi = 3.14; // floating point variable
     char grade = 'A'; // character variable

Example: -

```
#include <stdio.h>   // Header file


// This program shows the basic structure of C
int main () {
    // Variable declaration
    int age = 20;
    float height = 5.8;
    char grade = 'A';

    // Printing values
    printf("Age: %d\n", age);
    printf("Height: %.1f\n", height);
    printf("Grade: %c\n", grade);
    return 0;  // End of program
}
```

4. Operators in C

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

- **Operator: -**
  Operators are **symbols** used to perform operations on values and variables.
  Example: +, -, *, /

1. Arithmetic Operators

Used for **mathematical calculations**.

- + → addition
- - → subtraction
- * → multiplication
- / → division
- % → remainder (modulus)

Example: -

```
#include<stdio.h>

Main() {

Int a=10, b=5;

Printf (“%d”, a+b); //15

Printf (“%d”, a-b); //5

Printf (“%d”, a*b); //50

Printf (“%d”, a/b); //2

Printf (“%d”, a%b); //0

Return 0;

}
```

2. Relational Operators

Used to **compare two values**.

- == → equal to
- != → not equal to
- > → greater than
- < → less than
- >= → greater or equal
- <= → less or equal

Example: -

```
#include<stdio.h>

Main() {

Int a=10, b=5;

Printf (“%d”, a==b); //false

Printf (“%d”, a! =b); //true

Printf (“%d”, a>b); //true

Printf (“%d”, a<b); //false

Printf (“%d”, a>=b); //true

Printf (“%d”, a<=b); //false

Return 0;

}
```

3. Logical Operator

Used to combine **conditions**.

- && → AND (true if both are true)
- || → OR (true if one is true)
- ! → NOT (reverse the result)

Example: -

```
#include<stdio.h>
Main()
{
int a = 5, b = 10;
printf("%d", (a < b && b > 0)); // 1 (true)
printf("%d", !(a < b));         // 0 (false)
printf(“%d”, (a==5 || b==5); // 1 (true)
}
```

4. Assignment Operators

Used to **store values** in variables.

- = → assign
- += → add and assign
- -= → subtract and assign
- *= → multiply and assign
- /= → divide and assign

Example: -

```
#include <stdio.h>

int main() {

    int a;

    // = (assign)

    a = 10;

    printf("a = %d\n", a);   // Output: 10
```

```
    // += (add and assign)
    a += 5;   // a = a + 5 → 10 + 5 = 15
    printf("a += 5 → %d\n", a);   // Output: 15

    // -= (subtract and assign)
    a -= 3;   // a = a - 3 → 15 - 3 = 12
    printf("a -= 3 → %d\n", a);   // Output: 12

    // *= (multiply and assign)
    a *= 2;   // a = a * 2 → 12 * 2 = 24
    printf("a *= 2 → %d\n", a);   // Output: 24

    // /= (divide and assign)
    a /= 4;   // a = a / 4 → 24 / 4 = 6
    printf("a /= 4 → %d\n", a);   // Output: 6

    return 0;
}
```

5. Increment / Decrement Operators

- Used to **increase or decrease** value by 1.
  - **++** → increment
  - -- → decrement

There are **two types**:

1. **Pre-increment / Pre-decrement** → Changes the value first, then uses it.
2. **Post-increment / Post-decrement** → Uses the value first, then changes it.

Example: -

```
#include <stdio.h>


int main() {
    int a = 5, b;

    // Pre-increment (++a)
    b = ++a;   // a is increased first, then assigned
    printf("Pre-increment: a = %d, b = %d\n", a, b);

    // Post-increment (a++)
    b = a++;   // value of a is assigned first, then increased
    printf("Post-increment: a = %d, b = %d\n", a, b);

    // Pre-decrement (--a)
    b = --a;   // a is decreased first, then assigned
    printf("Pre-decrement: a = %d, b = %d\n", a, b);

    // Post-decrement (a--)
```

```
    b = a--;   // value of a is assigned first, then decreased
    printf("Post-decrement: a = %d, b = %d\n", a, b);

    return 0;
}
```

Output: -

Pre-increment: a = 6, b = 6

Post-increment: a = 7, b = 6

Pre-decrement: a = 6, b = 6

Post-decrement: a = 5, b = 6

6. Bitwise Operators

Work at the **bit (0/1) level**.

- & → AND
- | → OR
- ^ → XOR (exclusive OR)
- ~ → NOT (flip bits)
- << → left shift
- >> → right shift

Example: -

```
int a = 5, b = 3;

// 5 = 101, 3 = 011 (binary)

printf("%d", a & b); // 1 (001)

printf("%d", a | b); // 7 (111)
```

7. Conditional (Ternary) Operator

Shortcut for **if-else**.

- Syntax: condition? value_if_true: value_if_false

Example: -

```
#include <stdio.h>
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    // Ternary operator (? :)
    (num % 2 == 0)
        ? printf("%d is Even\n", num)
        : printf("%d is Odd\n", num);

    return 0;
}
```

5. Control Flow Statements in C

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

1. **if statement**

- Used when you want to run some code **only if a condition is true**.

  Syntax: -

  if (condition) {

  // code runs if condition is true

  }

Example: -

```
#include <stdio.h>

int main() {

    int age = 20;

    if (age >= 18) {   // condition

        printf("You are an Adult.\n");  // runs only if condition true

    }

    return 0;

}
```

2. if-else statement

- Used when you want to choose between **two options**.

  Syntax: -

```
if (condition) {

    // code runs if condition is true

} else {
```

```
    // code runs if condition is false
}
```

Example: -

```
#include <stdio.h>
int main() {
    int num = 5;

    if (num % 2 == 0) {
        printf("Even number\n");
    } else {
        printf("Odd number\n");
    }
    return 0;
}
```

3. nested if-else

- Means using **if inside another if**. Used when there are **multiple conditions**.
  Syntax: -

```
if (condition1) {
    // code if condition1 is true
    if (condition2) {
        // code if condition2 is also true
    } else {
        // code if condition2 is false
    }
} else {
    // code if condition1 is false
}
```

Example: -

```
#include <stdio.h>
int main() {
    int a = 10, b = 20, c = 15;

    if (a > b) {
        if (a > c) {
            printf("a is the biggest\n");
        } else {
            printf("c is the biggest\n");
        }
    } else {
        if (b > c) {
            printf("b is the biggest\n");
        } else {
            printf("c is the biggest\n");
        }
    }

    return 0;
}
```

4. switch statement

- Used when you want to compare **one variable** with **many possible values** (instead of writing many if-else).
  Syntax: -

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

Example: -

```
#include <stdio.h>
int main() {
    int choice;

    printf("Enter a number (1-3): ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("You chose ONE\n");
            break;
        case 2:
            printf("You chose TWO\n");
            break;
        case 3:
            printf("You chose THREE\n");
            break;
        default:
            printf("Invalid choice!\n");
    }

    return 0;
}
```

- if → check 1 condition.
- if-**else** → choose between 2 conditions.
- nested **if-else** → check multiple conditions step by step.
- switch → better when one variable has many possible values.

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.

1. **while loop**

- Condition is checked **before** the loop runs.
- If condition is **true**, loop runs. If false, it stops immediately.

Syntax: -

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

Example: -

```
#include <stdio.h>
int main() {
    int i = 1;
    while (i <= 5) {
        printf("%d\n", i);
        i++;
    }
    return 0;
}
```

2. for loop

- Used when you **know exactly how many times** you want to repeat.
- Initialization, condition, and update are written in one line.

Syntax: -

```
for (initialization; condition; modification) {
    // code to repeat
}
```

Example: -

```
#include <stdio.h>
int main() {
    for (int i = 1; i <= 5; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

3. do-while loop

- Runs the loop **at least once**, even if condition is false.
- Condition is checked **after** running the loop.

Syntax: -

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

Example: -

```
#include <stdio.h>
int main() {
    int i = 1;
    do {
        printf("%d\n", i);
        i++;
    } while (i <= 5);
    return 0;
}
```

Feature	while loop	for loop	do-while loop
Condition check	Before loop body	Before loop body	After loop body
Guaranteed execution	✗ Not guaranteed	✗ Not guaranteed	☑ At least once
Best for	Unknown iterations	Known/finite iterations	Must run at least once
Example use case	Reading file until EOF	Iterating through array indexes	Menu system, input prompt