**Module 2**

Introduction to Programming

* Overview of C Programming.
* THEORY EXERCISE:Write an essay covering the history and evolution of C programming. Explainits importance and why it is still used today.
* The History and Importance of C Programming
* **Introduction**
* C is a powerful programming language created in the early 1970s by Dennis Ritchie at Bell Labs. It was designed to be efficient and easy to use while still offering great control over computer hardware.
* **History and Evolution**
* C was developed to build the Unix operating system, which made it popular. In 1978, the book The C Programming Language by Kernighan and Ritchie helped standardize it. Later, ANSI C was introduced in the 1980s to ensure compatibility across different systems. Over time, C has remained relevant due to its efficiency and adaptability.
* **Why C Is Still Used Today**
* Speed & Efficiency – C is fast and ideal for system programming.
* Foundation for Other Languages – Languages like C++, Java, and Python are based on C.
* Used in Critical Systems – Operating systems, databases, and embedded systems rely on C.
* Portability – C programs work across multiple platforms.
* Control & Flexibility – C allows direct memory management, making it powerful for high-performance applications.
* **Conclusion**
* Even after 50 years, C remains one of the most important programming languages. It is the foundation of many modern technologies and is still widely used in system development, embedded programming, and more. Learning C helps programmers understand the core concepts of computing.
* LAB EXERCISE: Research and provide three real-world applications where C programmingisextensively used, such as in embedded systems, operating systems, or gamedevelopment.
* **Three Real-World Applications of C Programming.**
* **Operating Systems.**  
  C is the backbone of many operating systems, including Windows, Linux, and macOS. It allows developers to write efficient and fast system code that manages hardware and software interactions.
* **Embedded Systems.**  
  C is widely used in embedded systems like smart TVs, microwave ovens, medical devices, and car control systems. Its efficiency and direct hardware access make it perfect for small devices with limited resources.
* **Game Development.**Many game engines, such as Unity and Unreal Engine, use C and C++ for their core functionality. C helps in creating fast and responsive game mechanics, physics, and graphics rendering.
* Setting Up Environment.
* THEORY EXERCISE:Describe the steps to install a C compiler (e.g., GCC) and set up an IntegratedDevelopment Environment (IDE) like DevC++, VS Code, or CodeBlocks.
* **Steps to Install a C Compiler and Set Up an IDE.**

1. **Install a C Compiler (GCC)**

* Windows: Download and install MinGW or TDM-GCC from their official websites.
* Mac: Install Xcode Command Line Tools using the terminal: xcode-select –install.
* **Linux**: Install GCC using the terminal: sudo apt install gcc .

1. **Set Up an IDE**

* (A) Dev-C++
* Download Dev-C++ from its official website.
* Install it by following on-screen instructions.
* Open Dev-C++, create a new file, and start coding!
* (B) VS Code
* Download and install VS Code from the official website.
* Install the C/C++ extension from the Extensions menu.
* Set up GCC for compiling and running C programs.
* (C) Code Blocks
* Download Code::Blocks with MinGW from the official site.
* Install it and choose the default compiler settings.
* Start coding in the editor and compile your programs.
* Basic Structure of a C Program
* THEORY EXERCISE: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**

1. Headers (#include) – Used to include libraries (e.g., #include <stdio.h> for input/output).
2. Main Function (main()) – The starting point of the program.
3. Comments (// or /\* \*/) – Used to add notes that don’t affect the program.
4. Data Types – Define the type of data (e.g., int, float, char).
5. Variables – Store values in memory (e.g., int num = 10;).

* Example Program.

#include <stdio.h>  // Header file for input/output

int main() {  // Main function

    // This is a single-line comment

    /\* This is a

       multi-line comment \*/

    int age = 25;       // Integer variable

    float height = 5.9; // Floating-point variable

    char grade = 'A';   // Character variable

    printf("Age: %d\n", age);        // Print integer

    printf("Height: %.1f\n", height); // Print float

    printf("Grade: %c\n", grade);    // Print character

    return 0; // End of the program

}

* LAB EXERCISE: Write a C program that includes variables, constants, and comments. Declareand use different data types (int, char, float) and display their values.
* **C Program Example**

#include <stdio.h>  // Standard input/output library

#define PI 3.14  // Constant definition

int main() {

    // Variable declarations

    int age = 25;       // Integer variable

    float height = 5.9; // Floating-point variable

    char grade = 'A';   // Character variable

    // Display values

    printf("Age: %d\n", age);

    printf("Height: %.1f\n", height);

    printf("Grade: %c\n", grade);

    printf("Value of PI: %.2f\n", PI);  // Printing the constant

    return 0;  // End of program

}

* **Output of the Program**

Age: 25

Height: 5.9

Grade: A

Value of PI: 3.14

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

1. **Arithmetic Operators** (For Math Operations)

Used to perform basic mathematical calculations.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| + | Addition | 5 + 3 = 8 |
| - | Subtraction | 10 - 4 = 6 |
| \* | Multiplication | 6 \* 2 = 12 |
| / | Division | 8 / 2 = 4 |
| % | Modulus (Remainder) | 10 % 3 = 1 |

1. **Relational Operators (For Comparing Values)**

Used to compare two values and return true (1) or false (0).

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Meaning** | **Example (a = 5, b = 3)** | **Result** |
| **==** | **Equal to** | **a == b** | **0 (false)** |
| **!=** | **Not equal to** | **a != b** | **1 (true)** |
| **>** | **Greater than** | **a > b** | **1 (true)** |
| **<** | **Less than** | **a < b** | **0 (false)** |
| **>=** | **Greater or equal** | **a >= b** | **1 (true)** |
| **<=** | **Less or equal** | **a <= b** | **0 (false)** |

1. **Logical Operators (For Decision Making)**

Used to combine multiple conditions in if statements.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Meaning** | **Example (x = 5, y = 10)** | **Result** |
| && | AND (Both must be true) | (x > 2 && y < 15) | 1 (true) |
| ` |  | ` | OR (At least one must be true) |
| ! | NOT (Reverses true/false) | !(x > 10) | 1 (true) |

1. **Assignment Operators (For Assigning Values)**

Used to assign values to variables.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Meaning** | **Example (x = 10)** | **Same as** |
| = | Assign value | x = 5 | x = 5 |
| += | Add and assign | x += 2 | x = x + 2 |
| -= | Subtract and assign | x -= 3 | x = x - 3 |
| \*= | Multiply and assign | x \*= 4 | x = x \* 4 |
| /= | Divide and assign | x /= 2 | x = x / 2 |
| %= | Modulus and assign | x %= 3 | x = x % 3 |

1. **Increment and Decrement Operators (For Increasing/Decreasing Values)**

Used to increase or decrease a variable by 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Meaning** | **Example (x = 5)** | **Result** |
| ++ | Increment by 1 | x++ | x = 6 |
| -- | Decrement by 1 | x-- | x = 4 |

1. **Bitwise Operators (For Binary Calculations)**

Used for operations at the bit-level (0s and 1s).

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Meaning** | **Example (a = 5, b = 3)** | **Binary Result** |
| & | AND | a & b | 0101 & 0011 = 0001 (1) |
| ` | ` | OR | `a |
| ^ | XOR (Exclusive OR) | a ^ b | 0101 ^ 0011 = 0110 (6) |
| ~ | NOT (Complement) | ~a | ~0101 = 1010 |
| << | Left shift | a << 1 | 0101 << 1 = 1010 (10) |
| >> | Right shift | a >> 1 | 0101 >> 1 = 0010 (2) |

1. **Conditional (Ternary) Operator (Shortcut for if-else)**

Used to make decisions in one line.

Syntax: condition ? value\_if\_true : value\_if\_false;

Example: int a = 10, b = 20;

int min = (a < b) ? a : b;

printf("%d", min); // Output: 10

* LAB EXERCISE: Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results.
* **C Program:**

#include <stdio.h>  // Standard input/output library

* int main() {
* int a, b;  // Declare two integer variables
* // Accept input from the user
* printf("Enter first integer: ");
* scanf("%d", &a);
* printf("Enter second integer: ");
* scanf("%d", &b);
* // Arithmetic Operations
* printf("\nArithmetic Operations:\n");
* printf("Addition (a + b): %d\n", a + b);
* printf("Subtraction (a - b): %d\n", a - b);
* printf("Multiplication (a \* b): %d\n", a \* b);
* printf("Division (a / b): %d\n", (b != 0) ? (a / b) : 0);  // Avoid division by zero
* printf("Modulus (a %% b): %d\n", (b != 0) ? (a % b) : 0);
* // Relational Operations
* printf("\nRelational Operations:\n");
* printf("a == b: %d\n", a == b);
* printf("a != b: %d\n", a != b);
* printf("a > b: %d\n", a > b);
* printf("a < b: %d\n", a < b);
* printf("a >= b: %d\n", a >= b);
* printf("a <= b: %d\n", a <= b);
* // Logical Operations
* printf("\nLogical Operations:\n");
* printf("(a > 0 && b > 0) (Both positive): %d\n", (a > 0 && b > 0));
* printf("(a > 0 || b > 0) (At least one positive): %d\n", (a > 0 || b > 0));
* printf("!(a == b) (Not equal): %d\n", !(a == b));
* return 0;  // End of the program
* }
* **Output:**

Enter first integer: 10

Enter second integer: 5

Arithmetic Operations:

Addition (a + b): 15

Subtraction (a - b): 5

Multiplication (a \* b): 50

Division (a / b): 2

Modulus (a % b): 0

Relational Operations:

a == b: 0

a != b: 1

a > b: 1

a < b: 0

a >= b: 1

a <= b: 0

Logical Operations:

(a > 0 && b > 0) (Both positive): 1

(a > 0 || b > 0) (At least one positive): 1

!(a == b) (Not equal): 1

* Control Flow Statements in C
* THEORY EXERCISE: Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.
* Decision-Making Statements in C
* In C, decision-making statements allow the program to choose different actions based on conditions. The main types are:

1. **if Statement**
2. **if-else Statement**
3. **Nested if-else Statement**
4. **switch Statement**
5. **if Statement** (Executes code if a condition is true)

* Example:

int age = 20;

if (age >= 18) {

    printf("You are an adult.\n");

}

* Output:

If age is **18 or more**, it prints "You are an adult."  
 If age is **less than 18**, nothing happens.

1. **if-else Statement** (Chooses between two options)

int num = 10;

if (num % 2 == 0) {

    printf("Even number.\n");

} else {

    printf("Odd number.\n");

}

* Output:

If num is **even**, it prints "Even number."  
If num is **odd**, it prints "Odd number."

1. **Nested if-else Statement (if inside another if)**

* Example:

int marks = 85;

if (marks >= 50) {

    if (marks >= 80) {

        printf("Excellent!\n");

    } else {

        printf("Good job!\n");

    }

} else {

    printf("You failed. Try again!\n");

}}

* Output:

If marks is **80 or more**, prints "Excellent!"  
If marks is **50 to 79**, prints "Good job!"  
If marks is **below 50**, prints "You failed."

1. **switch Statement (Checks multiple conditions easily)**

* Example:

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");

}

* Output:

**If day = 3, it prints "Wednesday."**

**If day is not 1, 2, or 3, it prints "Invalid day."**

* LAB EXERCISE: Write a C program to check if a number is even or odd using an if-else statement. Extend the program using a switch statement to display the monthname based on the user’s input (1 for January, 2 for February, etc.)
* Here is a simple C program that:

1. Checks if a number is even or odd using an if-else statement.
2. Displays the month name using a switch statement based on the user’s input.

#include <stdio.h>  // Standard input/output library

int main() {

    int num, month;

    // Check if the number is even or odd

    printf("Enter a number: ");

    scanf("%d", &num);

    if (num % 2 == 0) {

        printf("%d is an Even number.\n", num);

    } else {

        printf("%d is an Odd number.\n", num);

    }

    // Display the month name using switch

    printf("\nEnter a month number (1-12): ");

    scanf("%d", &month);

    switch (month) {

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

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

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

        case 4: printf("April\n"); break;

        case 5: printf("May\n"); break;

        case 6: printf("June\n"); break;

        case 7: printf("July\n"); break;

        case 8: printf("August\n"); break;

        case 9: printf("September\n"); break;

        case 10: printf("October\n"); break;

        case 11: printf("November\n"); break;

        case 12: printf("December\n"); break;

        default: printf("Invalid month number! Please enter between 1 and 12.\n");

    }

    return 0;  // End of program

}

* **Example Outputs:**
* **Case 1:** User enters an even number and valid month

Enter a number: 10

10 is an Even number.

Enter a month number (1-12): 5

May

* **Case 2: User enters an odd number and valid month**

Enter a number: 7

7 is an Odd number.

Enter a month number (1-12): 11

November

* **Case 3: User enters an invalid month number**

Enter a number: 4

4 is an Even number.

Enter a month number (1-12): 15

Invalid month number! Please enter between 1 and 12.

* Looping in C

THEORY EXERCISE: Compare and contrast while loops, for loops, and do-while loops. Explainthescenarios in which each loop is most appropriate.

* Comparison of Loops in C: while, for, and do-while
* Loops allow us to **repeat a block of code multiple times**. There are three main types of loops in C:

|  |  |  |  |
| --- | --- | --- | --- |
| **Loop Type** | **Syntax** | **When It Runs** | **Best Used For** |
| **while** | while(condition) { code } | Runs **as long as** the condition is true | When the **number of iterations is unknown** |
| **for** | for(init; condition; update) { code } | Runs **a fixed number of times** | When the **number of iterations is known** |
| **do-while** | do { code } while(condition); | Runs **at least once**, then repeats if the condition is true | When the **loop must run at least once** |

1. **while Loop**

#include <stdio.h>

int main() {

    int num;

    printf("Enter a positive number: ");

    scanf("%d", &num);

    while (num <= 0) {  // Loop until a positive number is entered

        printf("Invalid! Enter again: ");

        scanf("%d", &num);

    }

    printf("You entered a valid number: %d\n", num);

    return 0;

}

* **Output:**

Enter a positive number: -3

Invalid! Enter again: 0

Invalid! Enter again: 5

You entered a valid number: 5

1. **for Loop**

#include <stdio.h>

int main() {

    for (int i = 1; i <= 5; i++) {  // Loop 5 times

        printf("%d\n", i);

    }

    return 0;

}

* **Output:**

1

2

3

4

5

1. **do-while Loop**

#include <stdio.h>

int main() {

    int password;

    do {

        printf("Enter password (1234): ");

        scanf("%d", &password);

    } while (password != 1234);  // Keep looping until correct

    printf("Access granted!\n");

    return 0;

}

* **Output:**

Enter password (1234): 5678

Enter password (1234): 4321

Enter password (1234): 1234

Access granted!

* **Key Differences & When to Use Each Loop**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **while Loop** | **for Loop** | **do-while Loop** |
| **Use When...** | Iterations **unknown** | Iterations **known** | Code must **run at least once** |
| **Condition Checked** | Before entering loop | Before entering loop | After executing loop body |
| **Guarantees Execution?** | No | No | **Yes, at least once** |
| **Example Usage** | User input validation | Counting numbers | Menu-based programs |

* **Conclusion**
* **Use while when the number of repetitions depends on a condition.**
* **Use for when you know how many times the loop should run.**
* **Use do-while when you must run the loop at least once before checking the condition.**