**Module 2 – Introduction to Programming**

1. **Overview of C Programming**

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

**The History and Evolution of C Programming**

C programming language, developed in the early 1970s by Dennis Ritchie at Bell Labs, was designed to provide low-level access to memory while maintaining high-level programming capabilities. It evolved from the B language, which itself was derived from BCPL, and was created to develop the UNIX operating system. C quickly became popular due to its efficiency, portability, and flexibility.

In the 1980s, C was standardized by ANSI (American National Standards Institute), leading to the development of ANSI C (C89). Later revisions, such as C99, C11, and C18, introduced features like better memory management, multithreading support, and enhanced security.

**Importance and Continued Usage**

C remains one of the most influential programming languages today. Its structured approach, rich library support, and direct hardware interaction make it ideal for system programming, embedded systems, and performance-critical applications. It serves as the foundation for many modern languages like C++, Java, and Python.

Despite the emergence of higher-level languages, C is still widely used in operating systems, game development, and firmware due to its speed and control over system resources. Its legacy and adaptability ensure its relevance in the ever-evolving tech industry.

* **LAB EXERCISE:** Research and provide three real-world applications where C programming is extensively used, such as in embedded systems, operating systems, or game development.

**Real-World Applications of C**

**Operating Systems** – Used in Windows, Linux, and macOS for efficient system performance.

**Embedded Systems** – Powers microcontrollers, IoT devices, and automotive systems.

**Game Development** – Essential for game engines like Unreal Engine for high-performance graphics and real-time processing.

1. **Setting Up Environment**

* **THEORY EXERCISE:** 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.

**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.
* **Linux/macOS:** Open the terminal and run:

sudo apt install gcc    # Ubuntu/Debian

sudo yum install gcc    # CentOS/RHEL

xcode-select --install  # macOS

**2. Install and Set Up an IDE**

* **Dev-C++**
  1. Download **Dev-C++** from SourceForge.
  2. Install and open it.
  3. Go to **Tools > Compiler Options** to ensure GCC is set up.
* **VS Code**
  1. Download and install **VS Code** from its official site.
  2. Install the **C/C++ Extension** from the Extensions Marketplace.
  3. Add GCC to the system path and configure tasks.json for compilation.
* **Code::Blocks**
  1. Download and install **Code::Blocks** with MinGW.
  2. Set **GNU GCC Compiler** in **Settings > Compiler**.
  3. Create a new C project and start coding.
* **LAB EXERCISE:** Install a C compiler on your system and configure the IDE. Write your first program to print "Hello, World!" and run it.

**Step 1: Install a C Compiler**

* **Windows:** Install **MinGW** or **TDM-GCC.**

**Step 2: Set Up an IDE**

* Install **Dev-C++**, **VS Code**, or **Code::Blocks** and configure the compiler.

**Step 3: Write and Run Your First Program**

Create a new C file (hello.c) and enter the following code:

#include <stdio.h>

int main() {

printf("Hello, World!\n");

return 0;

}

**Step 4: Compile and Run**

* Using Terminal/Command Prompt:

gcc hello.c -o hello

./hello

You should see:

Hello, World!

1. **Basic Structure of a C Program**

* **THEORY EXERCISE:** Explain the basic structure of a C program , including headers , main function, comments, datatypes, and variables. Provide examples.

Code:

*#include* <stdio.h>  *// Standard Input-Output Library*

*// Main function: Entry point of the program*

int main() {

*// Variable declaration*

    int age = 25; *// Integer type*

    float price = 99.99; *// Floating-point type*

    char grade = 'A'; *// Character type*

*// Printing values*

    printf("Age: %d\n", age); *// %d for integers*

    printf("Price: %.2f\n", price); *// %.2f for floating-point numbers with 2 decimal places*

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

*return* 0; *// Indicate successful execution*

}

**Explanation of Structure:**

1. **#include <stdio.h>** → Adds standard functions like printf().
2. **int main()** → The starting point of the program.
3. **Variables (int, float, char)** → Store values of different types.
4. **printf()** → Displays output on the screen.
5. **return 0;** → Indicates successful program execution.

Output:

Age: 25

Price: 99.99

Grade: A

* **LAB EXERCISE:** Write a C program that includes variables, constants ,and comments. Declare and use different datatypes(int,char,float) and display their values.

Code:

*#include* <stdio.h>  *// Standard Input-Output Library*

*#define* PI 3.1416 *// Defining a constant*

int main() {

*// Variable declaration*

    int age = 25; *// Integer type variable*

    float price = 99.99; *// Floating-point variable*

    char grade = 'A'; *// Character variable*

    const int year = 2024; *// Constant integer (cannot be changed)*

*// Printing values*

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

    printf("Price: %.2f\n", price); *// %.2f for floating-point*

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

    printf("Year: %d\n", year); *// Printing constant*

    printf("Value of PI: %.4f\n", PI); *// Printing defined constant*

*return* 0; *// Successful execution*

}

Output:

Age: 25

Price: 99.99

Grade: A

Year: 2024

Value of PI: 3.1416

1. **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** → Perform basic math operations.

* + (Addition), - (Subtraction), \* (Multiplication), / (Division), % (Modulus)

1. **Relational Operators** → Compare values, return true or false.

* == (Equal), != (Not equal), > (Greater), < (Less), >= (Greater or equal), <= (Less or equal)

1. **Logical Operators** → Used in conditional statements.

* && (AND), || (OR), ! (NOT)

1. **Assignment Operators** → Assign values.

* = (Assign), += (Add & assign), -= (Subtract & assign), \*= (Multiply & assign), /= (Divide & assign), %= (Modulus & assign)

1. **Increment/Decrement Operators** → Increase or decrease a value by 1.

* ++ (Increment), -- (Decrement)

1. **Bitwise Operators** → Work on individual bits.

* & (AND), | (OR), ^ (XOR), ~ (NOT), << (Left shift), >> (Right shift)

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

* condition ? value\_if\_true : value\_if\_false
* **LAB EXERCISE**: Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display there results.

Code:

*#include* <stdio.h>

int main() {

    int a, b;

*// Input two integers*

    printf("Enter two integers: ");

    scanf("%d %d", &a, &b);

*// Arithmetic Operations*

    printf("\nArithmetic Operations:\n");

    printf("%d + %d = %d\n", a, b, a + b);

    printf("%d - %d = %d\n", a, b, a - b);

    printf("%d \* %d = %d\n", a, b, a \* b);

*if* (b != 0) {

        printf("%d / %d = %d\n", a, b, a / b);

        printf("%d %% %d = %d\n", a, b, a % b);

    } *else* {

        printf("Division and modulus by zero are not allowed.\n");

    }

*// Relational Operations*

    printf("\nRelational Operations:\n");

    printf("%d == %d: %d\n", a, b, a == b);

    printf("%d != %d: %d\n", a, b, a != b);

    printf("%d > %d: %d\n", a, b, a > b);

    printf("%d < %d: %d\n", a, b, a < b);

    printf("%d >= %d: %d\n", a, b, a >= b);

    printf("%d <= %d: %d\n", a, b, a <= b);

*// Logical Operations*

    printf("\nLogical Operations:\n");

    printf("(%d && %d) = %d\n", a, b, a && b);

    printf("(%d || %d) = %d\n", a, b, a || b);

    printf("!%d = %d\n", a, !a);

    printf("!%d = %d\n", b, !b);

*return* 0;

}

Output:

Enter two integers: 5 3

Arithmetic Operations:

5 + 3 = 8

5 - 3 = 2

5 \* 3 = 15

5 / 3 = 1

5 % 3 = 2

Relational Operations:

5 == 3: 0

5 != 3: 1

5 > 3: 1

5 < 3: 0

5 >= 3: 1

5 <= 3: 0

Logical Operations:

(5 && 3) = 1

(5 || 3) = 1

!5 = 0

!3 = 0

1. **Control Flow Statements in C**

* THEORY EXERCISE: Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.

1. **if Statement** → Executes a block if a condition is true.

*if* (num > 0) {

    printf("Positive number");

}

1. **if-else Statement** → Executes one block if true, another if false

*if* (num % 2 == 0) {

    printf("Even");

} *else* {

    printf("Odd");

}

1. **Nested if-else Statement** → if inside another if.

*if* (num > 0) {

*if* (num % 2 == 0) {

        printf("Positive and even");

    } *else* {

        printf("Positive and odd");

    }

} *else* {

    printf("Non-positive");

}

1. **switch Statement** → Compares a variable with multiple values.

*switch* (day) {

*case* 1: printf("Monday"); *break*;

*case* 2: printf("Tuesday"); *break*;

*default*: printf("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 month name based on the user’s input (1 forJanuary,2 for February, etc.).

Code:

*#include* <stdio.h>

int main() {

    int num, month;

*// Check 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 Month Name*

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

    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 a number between 1 and 12.\n");

    }

*return* 0;

}

Output

Enter a number: 7

7 is an Odd number.

Enter a number (1-12) *for* a month: 3

March

Enter a number: 10

10 is an Even number.

Enter a number (1-12) *for* a month: 15

Invalid month number! Please enter a number between 1 and 12.

1. **Looping in C**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Loop Type** | **Condition Check** | **Executes At Least Once?** | **Best Use Case** |
| **for loop** | Before each iteration | ❌ No | Fixed number of iterations (e.g., counting loops) |
| **while loop** | Before each iteration | ❌ No | When the number of iterations is unknown (e.g., user input) |
| **do-while loop** | After each iteration | ✅ Yes | When execution must happen at least once (e.g., menus) |