**Java – Full stack Assignment**

* **Module 2 – Introduction to Programming:**

**Overview of C Programming:-**

• THEORY EXERCISE: C programming was developed in the early 1970s by Dennis Ritchie at Bell Labs as an improvement over the B language. It was designed to create the UNIX operating system, making it portable and efficient. Over time, C became widely adopted due to its simplicity, performance, and control over system hardware. The development of ANSI C in 1989 standardized the language, followed by C99 and C11 updates that introduced modern features.

C remains crucial today because of its efficiency, low-level capabilities, and portability. It is widely used in system programming, embedded systems, and high-performance applications, serving as the foundation for modern languages like C++, Java, and Python.

• LAB EXERCISE:

**Real-World Applications of C Programming:**

1. **Operating Systems** – C is the backbone of OS development, including Linux, Windows, and macOS.
2. **Embedded Systems** – Used in microcontrollers, IoT devices, and automotive software due to its efficiency and hardware control.
3. **Game Development** – Many game engines, like Unreal Engine, utilize C for performance-critical components.

**Setting Up Environment**

**Theory Exercise: Installing a C Compiler and Setting Up an IDE**

1. **Install a C Compiler (GCC)**
   * Windows: Install **MinGW** or **TDM-GCC** from their official websites.
   * macOS: Use **Homebrew** (brew install gcc).
   * Linux: Use the package manager (sudo apt install gcc for Ubuntu).
2. **Choose and Install an IDE**
   * **DevC++**: Download from its official website and install it.
   * **VS Code**: Install VS Code and add the **C/C++ extension** from the marketplace.
   * **CodeBlocks**: Download and install CodeBlocks with the built-in compiler.
3. **Configure the IDE**
   * Set the installed compiler path in the IDE settings.
   * Write and run a simple C program to verify installation.

**Lab Exercise: Writing Your First C Program:-**

1. **Install GCC and an IDE**
2. **Open the IDE and create a new C file**
3. **Write the following code:**
4. #include <stdio.h>
5. int main() {
6. printf("Hello, World!\n");
7. return 0;
8. }

**Basic Structure of a C Program**

**Theory Exercise: Understanding the Structure:-**

A basic C program consists of the following components:

1. **Header Files** – Includes standard libraries using #include.
2. **Main Function (main())** – The entry point of the program.
3. **Comments** – Used for code explanations (// single-line or /\* multi-line \*/).
4. **Data Types & Variables** – Variables store data of types like int, char, float, etc.
5. **Statements & Functions** – The program logic written inside {}.

**Example: Basic C Program Structure**

#include <stdio.h>  functions

 Main function

int main() {

    int number = 10;

    printf("Number: %d\n", number);

    return 0;

}

**Lab Exercise: Writing a C Program with Variables and Constants:-**

#include <stdio.h>

int main() {

    int age = 25;

    float pi = 3.14;

    char grade = 'A';

    printf("Age: %d\n", age);

    printf("Value of Pi: %.2f\n", pi);

    printf("Grade: %c\n", grade);

    printf("Current Year: %d\n", YEAR);

    return 0;

}

Operators in C

Theory Exercise: Types of Operators in C

Arithmetic Operators – Used for basic mathematical operations.

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

Relational Operators – Compare two values and return true (1) or false (0).

== (Equal), != (Not equal), > (Greater than), < (Less than), >=, <=.

Logical Operators – Used for logical conditions.

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

Assignment Operators – Assign values to variables.

= (Simple assignment), +=, -=, \*=, /=, %=.

Increment & Decrement Operators – Increase or decrease value by 1.

++ (Increment), -- (Decrement).

Bitwise Operators – Perform operations at the binary level.

& (AND), | (OR), ^ (XOR), << (Left shift), >> (Right shift).

Conditional (Ternary) Operator – A shorthand for if-else statements.

Syntax: condition ? expr1 : expr2;

Lab Exercise: C Program Demonstrating Operators:-

#include <stdio.h>

int main() {

    int a, b;

    // Taking input from user

    printf(“Enter two integers: “);

    scanf(“%d %d”, &a, &b);

    // Arithmetic operations

    printf(“\nArithmetic Operations:\n”);

    printf(“Sum: %d\n”, a + b);

    printf(“Difference: %d\n”, a – b);

    printf(“Product: %d\n”, a \* b);

    printf(“Quotient: %d\n”, a / b);

    printf(“Remainder: %d\n”, a % b);

    // 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);

    // Logical operations

    printf(“\nLogical Operations:\n”);

    printf(“(a > 0 && b > 0): %d\n”, (a > 0 && b > 0));

    printf(“(a > 0 || b > 0): %d\n”, (a > 0 || b > 0));

    printf(“!(a > b): %d\n”, !(a > b));

    return 0;

}

**Control Flow Statements in C**

**Theory Exercise: Decision-Making Statements in C:-**

1. **if Statement** – Executes a block of code if a condition is true.
2. if (x > 0) {
3. printf("Positive number");
4. }
5. **if-else Statement** – Executes one block if the condition is true, another if false.
6. if (x % 2 == 0) {
7. printf("Even number");
8. } else {
9. printf("Odd number");
10. }
11. **Nested if-else** – An if-else inside another if-else statement.
12. if (age >= 18) {
13. if (age >= 65) {
14. printf("Senior Citizen");
15. } else {
16. printf("Adult");
17. }
18. } else {
19. printf("Minor");
20. }
21. **switch Statement** – Evaluates a variable and executes the corresponding case.
22. switch (day) {
23. case 1: printf("Monday"); break;
24. case 2: printf("Tuesday"); break;
25. default: printf("Invalid choice");
26. }

**Lab Exercise: Check Even/Odd and Display Month Name:-**

#include <stdio.h>

int main() {

    int num, month;

    printf("Enter a number: ");

    scanf("%d", &num);

    if (num % 2 == 0) {

        printf("%d is Even\n", num);

    } else {

        printf("%d is Odd\n", num);

    }

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

    scanf("%d", &month);

    switch (month) {

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

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

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

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

        case 5: printf("May"); break;

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

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

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

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

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

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

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

        default: printf("Invalid month number");

    }

    return 0;

}

**Looping in C**

**Theory Exercise: Comparing Loop Types:-**

1. **while Loop**
   * Checks the condition first; executes the loop body if the condition is true.
   * Used when the number of iterations is **not known in advance**.
2. int i = 1;
3. while (i <= 10) {
4. printf("%d ", i);
5. i++;
6. }
7. **for Loop**
   * Initializes, checks the condition, and increments in a single line.
8. for (int i = 1; i <= 10; i++) {
9. printf("%d ", i);
10. }
11. **do-while Loop**
    * Executes at least once, then checks the condition.
    * Useful when the loop **must run at least once**, even if the condition is false.

int i = 1;

do {

    printf("%d ", i);

    i++;

} while (i <= 10);

| **Loop Type** | **Condition Checked** | **Guaranteed Execution** | **Best Use Case** |
| --- | --- | --- | --- |
| while | Before the loop | No | When the number of iterations is unknown |
| for | Before the loop | No | When the number of iterations is known |
| do-while | After the loop | Yes (at least once) | When the loop must execute at least once |

**Lab Exercise: Printing Numbers 1 to 10 Using All Loops:-**

#include <stdio.h>

int main() {

    int i;

    printf("Using while loop:\n");

    i = 1;

    while (i <= 10) {

        printf("%d ", i);

        i++;

    }

    printf("\n\nUsing for loop:\n");

    for (i = 1; i <= 10; i++) {

        printf("%d ", i);

    }

    printf("\n\nUsing do-while loop:\n");

    i = 1;

    do {

        printf("%d ", i);

        i++;

    } while (i <= 10);

    return 0;

}

**Loop Control Statements in C**

**Theory Exercise: Understanding break, continue, and goto Statements:-**

1. **break Statement**
   * Exits the loop immediately when encountered.
   * Used to **stop execution** when a specific condition is met.
2. for (int i = 1; i <= 10; i++) {
3. if (i == 5) {
4. break;  // Stops loop when i is 5
5. }
6. printf("%d ", i);
7. }
8. **continue Statement**
   * Skips the current iteration and moves to the next one.
   * Used to **skip specific values** while continuing the loop.
9. **goto Statement**
   * Transfers control to a labeled part of the program.
10. for (int i = 1; i <= 5; i++) {
11. if (i == 3) {
12. continue;  // Skips printing 3
13. }
14. printf("%d ", i);
15. }
    * Avoided in structured programming but can be useful in certain cases.
16. int i = 1;
17. loop:
18. if (i <= 5) {
19. printf("%d ", i);
20. i++;
21. goto loop;  // Jumps back to the loop label
22. }

**Lab Exercise: Using break and continue in a Loop:-**

#include <stdio.h>

int main() {

    int i;

    printf("Using break statement:\n");

    for (i = 1; i <= 10; i++) {

        if (i == 5) {

            break;

        }

        printf("%d ", i);

    }

    printf("\n\nUsing continue statement:\n");

    for (i = 1; i <= 5; i++) {

        if (i == 3) {

            continue;

        }

        printf("%d ", i);

    }

    return 0;

}

**Functions in C**

**THEORY EXERCISE:**

What are Functions in C?

A function in C is a block of code that performs a specific task. Functions help in code modularity, reusability, and better readability.

Function Components:

1. Function Declaration (Prototype)  
   This informs the compiler about the function’s name, return type, and parameters before its actual definition.
2. Function Definition  
   This contains the actual implementation of the function.
3. Function Call  
   This executes the function by passing arguments (if any).

**Example:**

#include <stdio.h>

// Function Declaration

int add(int, int);

int main() {

    int result = add(5, 3);  // Function Call

    printf("Sum: %d\n", result);

    return 0;

}

// Function Definition

int add(int a, int b) {

    return a + b;

}

**LAB EXERCISE:**

**C Program to Calculate Factorial Using a Function**

#include <stdio.h>

// Function Declaration

long factorial(int);

int main() {

    int num;

    printf("Enter a number: ");

    scanf("%d", &num);

    if (num < 0) {

        printf("Factorial is not defined for negative numbers.\n");

    } else {

        printf("Factorial of %d is %ld\n", num, factorial(num));

    }

    return 0;

}

// Function Definition

long factorial(int n) {

    if (n == 0 || n == 1)

        return 1;

    return n \* factorial(n - 1); // Recursive call

}

**Arrays in C**

**Theory Exercise: Understanding Arrays**

An **array** in C is a collection of elements of the same data type stored in contiguous memory locations. Arrays help store multiple values efficiently instead of using multiple variables.

**Types of Arrays:**

1. **One-Dimensional Array**
   * Stores a list of values.
   * Example:

int arr[5] = {10, 20, 30, 40, 50};

1. **Multi-Dimensional Array (2D, 3D, etc.)**
   * Stores tabular or matrix-like data.
   * Example (2D array – 3x3 matrix):
2. int matrix[3][3] = {
3. {1, 2, 3},
4. {4, 5, 6},
5. {7, 8, 9}
6. };

| **Feature** | **One-Dimensional Array** | **Multi-Dimensional Array** |
| --- | --- | --- |
| Storage | Linear sequence | Table-like structure |
| Access | arr[i] | arr[row][col] |
| Usage | List of numbers, marks, etc. | Matrices, grids, images |

**Lab Exercise: One-Dimensional & Two-Dimensional Arrays:-**

#include <stdio.h>

int main() {

    int arr[5] = {10, 20, 30, 40, 50};

    printf("One-Dimensional Array Elements:\n");

    for (int i = 0; i < 5; i++) {

        printf("%d ", arr[i]);

    }

    int matrix[3][3] = {

        {1, 2, 3},

        {4, 5, 6},

        {7, 8, 9}

    };

    int sum = 0;

    printf("\n\nTwo-Dimensional Array (3x3 Matrix):\n");

    for (int i = 0; i < 3; i++) {

        for (int j = 0; j < 3; j++) {

            printf("%d ", matrix[i][j]);

            sum += matrix[i][j];

        }

        printf("\n");

    }

    printf("\nSum of all matrix elements: %d", sum);

    return 0;

}

**Pointers in C**

**Theory Exercise: Understanding Pointers:-**

A **pointer** is a variable that stores the memory address of another variable. Instead of storing a direct value, it holds the address where the value is stored.

**Pointer Declaration and Initialization**

int \*ptr;

int x = 10;

ptr = &x; // Storing the address of x in ptr

Here, ptr stores the memory address of x.

**Importance of Pointers in C**

* **Efficient Memory Management** – Helps in dynamic memory allocation.
* **Faster Processing** – Direct access to memory speeds up operations.
* **Used in Data Structures** – Essential for linked lists, trees, and graphs.
* **Function Arguments** – Allows passing large data structures efficiently by reference.

**Lab Exercise: Demonstrating Pointer Usage:-**

#include <stdio.h>

int main() {

    int num = 10;

    int \*ptr;  // Pointer declaration

    ptr = &num;  // Storing address of num

    printf("Value of num: %d\n", num);

    printf("Address of num: %p\n", &num);

    printf("Pointer ptr stores: %p\n", ptr);

    printf("Value at address stored in ptr: %d\n", \*ptr);

    // Modifying value using pointer

    \*ptr = 20;

    printf("\nAfter modifying through pointer:\n");

    printf("Value of num: %d\n", num);

    return 0;

}

**Strings in C**

**Theory Exercise: String Handling Functions:-**

A **string** in C is an array of characters ending with a null character \0. C provides several functions in the <string.h> library to manipulate strings efficiently.

**Common String Functions:**

| **Function** | **Description** | **Example Usage** |
| --- | --- | --- |
| strlen(str) | Returns the length of the string (excluding \0). | int len = strlen("Hello"); // len = 5 |
| strcpy(dest, src) | Copies src string into dest. | strcpy(dest, "World"); |
| strcat(dest, src) | Appends src to the end of dest. | strcat(str1, str2); |
| strcmp(str1, str2) | Compares two strings (0 if equal, <0 if str1 < str2, >0 if str1 > str2). | strcmp("abc", "abd") returns <0 |
| strchr(str, ch) | Finds first occurrence of ch in str. | strchr("Hello", 'l'); returns pointer to 'l' |

**Lab Exercise: Concatenating Strings and Finding Length**

#include <stdio.h>

#include <string.h>

int main() {

    char str1[100], str2[100];

    printf("Enter first string: ");

    gets(str1);

    printf("Enter second string: ");

    gets(str2);

    strcat(str1, str2);

    printf("\nConcatenated String: %s\n", str1);

    printf("Length of Concatenated String: %lu\n", strlen(str1));

    return 0;

}

**Structures in C**

**Theory Exercise: Understanding Structures**

A **structure** in C is a user-defined data type that allows grouping different types of variables under one name. It is useful for representing complex data.

**Declaring a Structure:**

struct Student {

    char name[50];

    int roll;

    float marks;

};

**Initializing and Accessing Structure Members:**

struct Student s1 = {"John", 101, 85.5

    printf("Name: %s, Roll No: %d, Marks: %.2f", s1.name, s1.roll, s1.marks);

**Why Use Structures?**

* Helps **organize related data** (e.g., student records, employee details).
* Makes **complex programs more manageable**.
* Used in **file handling, databases, and system programming**.

**Lab Exercise: Storing Student Details Using an Array of Structures**

#include <stdio.h>

struct Student {

    char name[50];

    int roll;

    float marks;

};

int main() {

    struct Student students[3];

    for (int i = 0; i < 3; i++) {

        printf("Enter details for student %d:\n", i + 1);

        printf("Name: ");

        scanf(" %[^\n]", students[i].name);

        printf("Roll No: ");

        scanf("%d", &students[i].roll);

        printf("Marks: ");

        scanf("%f", &students[i].marks);

    }

    printf("\nStudent Details:\n");

    for (int i = 0; i < 3; i++) {

        printf("Name: %s, Roll No: %d, Marks: %.2f\n", students[i].name, students[i].roll, students[i].marks);

    }

    return 0;

}

**File Handling in C**

**Theory Exercise: Understanding File Handling**

**File handling** in C allows programs to store and retrieve data from external files instead of using standard input/output. This is crucial for **data persistence** beyond program execution.

**File Operations in C**

1. **Opening a File (fopen())**
   * Syntax: FILE \*fp = fopen("filename.txt", "mode");
   * Modes:
     + "r" – Read
     + "w" – Write (creates a new file or overwrites existing content)
     + "a" – Append
     + "r+" / "w+" – Read & Write
2. **Writing to a File (fprintf() / fputs())**
   * Example: fprintf(fp, "Hello, File!");
3. **Reading from a File (fscanf() / fgets())**
   * Example: fgets(buffer, 100, fp);
4. **Closing a File (fclose())**
   * Always close a file using fclose(fp); to avoid memory leaks.

**Lab Exercise: File Creation, Writing, and Reading**

#include <stdio.h>

int main() {

    FILE \*fp;

    char data[100];

    fp = fopen("sample.txt", "w");

    if (fp == NULL) {

        printf("Error opening file!\n");

        return 1;

    }

    fprintf(fp, "Hello, this is a sample file.\n");

    fclose(fp);

    fp = fopen("sample.txt", "r");

    if (fp == NULL) {

        printf("Error opening file!\n");

        return 1;

    }

    printf("File Contents:\n");

    while (fgets(data, sizeof(data), fp) != NULL) {

        printf("%s", data);

    }

    fclose(fp);

    return 0;

}

C Program: Simple Calculator:

#include <stdio.h>

int main() {

    char op;

    double num1, num2, result;

    printf("Enter first number: ");

    scanf("%lf", &num1);

    printf("Enter an operator (+, -, \*, /, %%): ");

    scanf(" %c", &op);

    printf("Enter second number: ");

    scanf("%lf", &num2);

    switch (op) {

        case '+':

            result = num1 + num2;

            printf("Result: %.2lf\n", result);

            break;

        case '-':

            result = num1 - num2;

            printf("Result: %.2lf\n", result);

            break;

        case '\*':

            result = num1 \* num2;

            printf("Result: %.2lf\n", result);

            break;

        case '/':

            if (num2 != 0) {

                result = num1 / num2;

                printf("Result: %.2lf\n", result);

            } else {

                printf("Error! Division by zero is not allowed.\n");

            }

            break;

        case '%':

            if ((int)num2 != 0) {

                printf("Result: %d\n", (int)num1 % (int)num2);

            } else {

                printf("Error! Modulus by zero is not allowed.\n");

            }

            break;

        default:

            printf("Invalid operator! Please use +, -, \*, /, or %%.\n");

    }

    return 0;

}

C Program: Check Number Properties:

#include <stdio.h>

int main() {

    int num;

    printf("Enter an integer: ");

    scanf("%d", &num);

    if (num % 2 == 0)

        printf("The number is Even.\n");

    else

        printf("The number is Odd.\n");

    if (num > 0)

        printf("The number is Positive.\n");

    else if (num < 0)

        printf("The number is Negative.\n");

    else

        printf("The number is Zero.\n");

    if (num % 3 == 0 && num % 5 == 0)

        printf("The number is a multiple of both 3 and 5.\n");

    else

        printf("The number is NOT a multiple of both 3 and 5.\n");

    return 0;

}

EXTRA LAB EXERCISES FOR IMPROVING PROGRAMMING LOGIC:

LAB EXERCISE 1: Simple Calculator:

#include <stdio.h>

int main() {

    double num1, num2, result;

    char operator;

    printf("Enter first number: ");

    scanf("%lf", &num1);

    printf("Enter an operator (+, -, \*, /, %): ");

    scanf(" %c", &operator);

    printf("Enter second number: ");

    scanf("%lf", &num2);

    switch (operator) {

        case '+':

            result = num1 + num2;

            printf("Result: %.2lf\n", result);

            break;

        case '-':

            result = num1 - num2;

            printf("Result: %.2lf\n", result);

            break;

        case '\*':

            result = num1 \* num2;

            printf("Result: %.2lf\n", result);

            break;

        case '/':

            if (num2 != 0)

                printf("Result: %.2lf\n", num1 / num2);

            else

                printf("Error: Division by zero is not allowed.\n");

            break;

        case '%':

            if ((int)num2 != 0)

                printf("Result: %d\n", (int)num1 % (int)num2);

            else

                printf("Error: Modulus by zero is not allowed.\n");

            break;

        default:

            printf("Error: Invalid operator.\n");

    }

    return 0;

}

#include <stdio.h>

int main() {

#include <stdio.h>

int main() {

    int num;

    printf("Enter an integer: ");

    scanf("%d", &num);

    if (num % 2 == 0)

        printf("The number is Even.\n");

    else

        printf("The number is Odd.\n");

    if (num > 0)

        printf("The number is Positive.\n");

    else if (num < 0)

        printf("The number is Negative.\n");

    else

        printf("The number is Zero.\n");

    if (num % 3 == 0 && num % 5 == 0)

        printf("The number is a multiple of both 3 and 5.\n");

    else

        printf("The number is NOT a multiple of both 3 and 5.\n");

    return 0;

}

2. Control Statements:

#include <stdio.h>

int main() {

    int marks;

    printf("Enter the student's marks: ");

    scanf("%d", &marks);

    switch (marks / 10) {

        case 10:

        case 9:

            printf("Grade: A\n");

            break;

        case 8:

        case 7:

            printf("Grade: B\n");

            break;

        case 6:

        case 5:

            printf("Grade: C\n");

            break;

        default:

            printf("Grade: D\n");

    }

    return 0;

}

#include <stdio.h>

int main() {

    int num1, num2, num3, largest, smallest;

    printf("Enter three numbers: ");

    scanf("%d %d %d", &num1, &num2, &num3);

    switch (num1 >= num2 && num1 >= num3) {

        case 1:

            largest = num1;

            break;

        default:

            switch (num2 >= num1 && num2 >= num3) {

                case 1:

                    largest = num2;

                    break;

                default:

                    largest = num3;

            }

    }

    switch (num1 <= num2 && num1 <= num3) {

        case 1:

            smallest = num1;

            break;

        default:

            switch (num2 <= num1 && num2 <= num3) {

                case 1:

                    smallest = num2;

                    break;

                default:

                    smallest = num3;

            }

    }

    printf("Largest number: %d\n", largest);

    printf("Smallest number: %d\n", smallest);

    return 0;

}

Prime Number Check (Using for loop)

#include <stdio.h>

int main() {

    int num, i, isPrime = 1;

    // Taking input from the user

    printf("Enter a number: ");

    scanf("%d", &num);

    // Edge cases for numbers <= 1

    if (num <= 1) {

        isPrime = 0;

    } else {

        // Check divisibility from 2 to num/2

        for (i = 2; i <= num / 2; i++) {

            if (num % i == 0) {

                isPrime = 0;

                break;  // No need to check further

            }

        }

    }

    // Output result

    if (isPrime)

        printf("%d is a Prime Number.\n", num);

    else

        printf("%d is NOT a Prime Number.\n", num);

    return 0;

}

2️Print All Prime Numbers up to N (Challenge Solution)

#include <stdio.h>

int main() {

    int num, i, j, isPrime;

    // Taking input from the user

    printf("Enter a number: ");

    scanf("%d", &num);

    printf("Prime numbers between 1 and %d are: ", num);

    // Loop from 2 to num

    for (i = 2; i <= num; i++) {

        isPrime = 1; // Assume i is prime

        // Check divisibility

        for (j = 2; j <= i / 2; j++) {

            if (i % j == 0) {

                isPrime = 0;

                break;

            }

        }

        // Print if prime

        if (isPrime)

            printf("%d ", i);

    }

    printf("\n");

    return 0;

}

1️.Basic Multiplication Table (1 to 10):

#include <stdio.h>

int main() {

    int num, i;

    // Taking input from the user

    printf("Enter a number: ");

    scanf("%d", &num);

    // Printing multiplication table from 1 to 10

    printf("Multiplication Table of %d:\n", num);

    for (i = 1; i <= 10; i++) {

        printf("%d x %d = %d\n", num, i, num \* i);

    }

    return 0;

}