**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) |

* **LAB EXERCISE**: Write a C program to print numbers from 1 to 10 using all three types of loops (while, for, do-while).

Code:

#include <stdio.h>

int main() {

    int i;

    // Using for loop

    printf("Using for loop:\n");

    for(i = 1; i <= 10; i++) {

        printf("%d ", i);

    }

    printf("\n");

    // Using while loop

    printf("Using while loop:\n");

    i = 1;  // Reset i

    while(i <= 10) {

        printf("%d ", i);

        i++;

    }

    printf("\n");

    // Using do-while loop

    printf("Using do-while loop:\n");

    i = 1;  // Reset i

    do {

        printf("%d ", i);

        i++;

    } while(i <= 10);

    printf("\n");

    return 0;

}

Output:

Using for loop:

1 2 3 4 5 6 7 8 9 10

Using while loop:

1 2 3 4 5 6 7 8 9 10

Using do-while loop:

1 2 3 4 5 6 7 8 9 10

1. **Loop Control Statements**

* **THEORY EXERCISE:** Explain the use of break, continue, and goto statements in C. Provide examples of each.

**1. break Statement**

Used to exit a loop or switch case immediately.

**Code:**

for (int i = 1; i <= 10; i++) {

    if (i == 5) break;

    printf("%d ", i);

}

**Output:** 1 2 3 4

**2. continue Statement**

Skips the current iteration and moves to the next.

**Code:**

for (int i = 1; i <= 10; i++) {

    if (i == 5) continue;

    printf("%d ", i);

}

**Output:** 1 2 3 4 6 7 8 9 10

**3. goto Statement**

Jumps to a **label** unconditionally (not recommended for structured programming).

**Code:**

int i = 1;

while (i <= 10) {

    if (i == 6) goto end;

    printf("%d ", i);

    i++;

}

end:

printf("\nLoop exited using goto.");

**Output:**

1 2 3 4 5

Loop exited using goto.

* **LAB EXERCISE:** Write a C program that uses the break statement to stop printing numbers when it reaches 5. Modify the program to skip printing the number 3 using the continue statement.

**1. Using break (Stops at 5)**

Code:

#include <stdio.h>

int main() {

    for (int i = 1; i <= 10; i++) {

        if (i == 5)

            break;  // Stop loop when i is 5

        printf("%d ", i);

    }

    return 0;

}

Output:

1 2 3 4

**2. Using continue (Skips 3 but prints all numbers)**

Code:

#include <stdio.h>

int main() {

    for (int i = 1; i <= 10; i++) {

        if (i == 3)

            continue;  // Skip number 3

        printf("%d ", i);

    }

    return 0;

}

Output:

1 2 4 5 6 7 8 9 10

1. **Functions in C**

* **THEORY EXERCISE**: What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.

A **function** is a reusable block of code that performs a specific task.

**1. Function Components**

* + **Declaration (Prototype) :** Tells the compiler about the function.
  + **Definition:** Contains the actual code.
  + **Call:** Executes the function.

**2. Function Syntax**

returnType functionName(parameterList);  // Declaration

returnType functionName(parameterList) {  // Definition

    // Function body

    return value;

}

functionName(arguments);  // Function Call

**3. Example: Function to Add Two Numbers**

Code**:**

#include <stdio.h>

// Function Declaration

int add(int, int);

int main() {

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

    printf("Sum: %d\n", sum);

    return 0;

}

// Function Definition

int add(int a, int b) {

    return a + b;

}

Output:

Sum: 8

* **LAB EXERCISE:** Write a C program that calculates the factorial of a number using a function. Include function declaration, definition, and call.

Code:

#include <stdio.h>

// Function Declaration (Prototype)

long long factorial(int n);

int main() {

    int num;

    // Input from user

    printf("Enter a number: ");

    scanf("%d", &num);

    // Function Call

    if (num < 0)

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

    else

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

    return 0;

}

// Function Definition

long long factorial(int n) {

    long long fact = 1;

    for (int i = 1; i <= n; i++) {

        fact \*= i;

    }

    return fact;

}

Output:

Enter a number: 5

Factorial of 5 is 120

1. **Arrays in C**

* **THEORY EXERCISE:** Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.

An **array** is a collection of elements of the same type, stored in contiguous memory locations. It allows efficient data handling using indices.

**1. One-Dimensional Array (1D Array)**

A **1D array** is a simple list of elements accessed using a single index.

Example:

int arr[5] = {10, 20, 30, 40, 50}; // 1D Array

printf("%d", arr[2]); // Output: 30

**2. Multi-Dimensional Array**

A **multi-dimensional array** consists of multiple rows and columns.

Example:

int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}}; // 2D Array

printf("%d", matrix[1][2]); // Output: 6

Difference Between 1D and Multi-Dimensional Arrays

|  |  |  |
| --- | --- | --- |
| **Feature** | **One-Dimensional Array** | **Multi-Dimensional Array** |
| **Definition** | Stores a list of elements in a single row | Stores elements in multiple rows and columns (or more) |
| **Access Method** | Accessed using a single index arr[i] | Accessed using multiple indices arr[i][j], arr[i][j][k], etc. |
| **Storage Format** | Linear memory allocation | Row-major order (C stores elements row by row) |
| **Use Cases** | Storing lists (e.g., student marks, prices) | Representing tables, matrices, and grids |

* **LAB EXERCISE :** Write a C program that stores 5 integers in a one-dimensional array and prints them.

Code:

#include <stdio.h>

int main() {

    int numbers[5]; // Declare an array of 5 integers

    // Input: Getting 5 integers from the user

    printf("Enter 5 integers:\n");

    for (int i = 0; i < 5; i++) {

        scanf("%d", &numbers[i]);

    }

    // Output: Printing the stored integers

    printf("The stored integers are:\n");

    for (int i = 0; i < 5; i++) {

        printf("%d ", numbers[i]);

    }

    printf("\n");

    return 0;

}

Output:

Enter 5 integers:

10 20 30 40 50

The stored integers are:

10 20 30 40 50

* **LAB EXERCISE :** Extend this to handle a two-dimensional array (3x3 matrix) and calculate the sum of all elements.

Code:

#include <stdio.h>

int main() {

    int matrix[3][3]; // Declare a 3x3 matrix

    int sum = 0;      // Variable to store the sum of elements

    // Input: Getting 3x3 matrix elements from the user

    printf("Enter 9 integers for the 3x3 matrix:\n");

    for (int i = 0; i < 3; i++) {

        for (int j = 0; j < 3; j++) {

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

            sum += matrix[i][j]; // Adding each element to sum

        }

    }

    // Output: Printing the stored matrix

    printf("The entered 3x3 matrix is:\n");

    for (int i = 0; i < 3; i++) {

        for (int j = 0; j < 3; j++) {

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

        }

        printf("\n");

    }

    // Printing the sum of all elements

    printf("The sum of all elements in the matrix is: %d\n", sum);

    return 0;

}

Output:

Enter 9 integers for the 3x3 matrix:

1 2 3

4 5 6

7 8 9

The entered 3x3 matrix is:

1 2 3

4 5 6

7 8 9

The sum of all elements in the matrix is: 45

1. **Pointers in C**

* **THEORY EXERCISE**: Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

A **pointer** in C is a variable that stores the memory address of another variable.

**Declaration & Initialization:**

int \*ptr;   // Declares a pointer to an integer

int num = 10;

ptr = &num;  // Stores the address of num in ptr

**Why Are Pointers Important?**

* Efficient Memory Management
* Dynamic Memory Allocation (malloc, free)
* Array & String Manipulation
* Pass-by-Reference in Functions
* Building Data Structures (Linked Lists, Trees, etc.)
* Pointer Arithmetic for Fast Iteration
* **LAB EXERCISE:** Write a C program to demonstrate pointer usage.

Code:

#include <stdio.h>

// Function to modify value using pointer

void modifyValue(int \*ptr) {

    \*ptr = 20; // Changing the value at the memory address

}

int main() {

    int num = 10;

    int \*ptr = &num; // Pointer storing the address of num

    // Displaying pointer details

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

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

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

    printf("Value at ptr (dereferencing): %d\n\n", \*ptr);

    // Modifying variable using pointer

    \*ptr = 15;

    printf("After modifying via pointer, num = %d\n\n", num);

    // Passing pointer to function

    modifyValue(&num);

    printf("After function call, num = %d\n", num);

    return 0;

}

Output:

Value of num: 10

Address of num: 0x7ffde2b6e4ac

Pointer ptr stores: 0x7ffde2b6e4ac

Value at ptr (dereferencing): 10

After modifying via pointer, num = 15

After function call, num = 20

* **LAB EXERCISE :** Use a pointer to modify the value of a variable and print the result.

Code:

#include <stdio.h>

int main() {

    int num = 10;     // Declare an integer variable

    int \*ptr = &num;  // Declare a pointer and store the address of num

    printf("Before modification:\n");

    printf("num = %d\n", num);

    // Modifying the value using the pointer

    \*ptr = 20;

    printf("\nAfter modification using pointer:\n");

    printf("num = %d\n", num);

    return 0;

}

Output:

Before modification:

num = 10

After modification using pointer:

num = 20

1. **Strings in C**

* **THEORY EXERCISE:** Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr().
  1. **strlen(str) –** Returns the length of a string (excluding \0).

printf("%lu", strlen("Hello")); // Output: 5

* 1. **strcpy(dest, src) –** Copies src into dest

strcpy(dest, "World"); // dest = "World"

* 1. **strcat(dest, src) –** Appends src to dest

strcat(str1, " World"); // str1 = "Hello World"

* 1. **strcmp(str1, str2)** – Compares two strings.
* 0 → Equal
* <0 → str1 < str2
* >0 → str1 > str2

strcmp("Apple", "Banana"); // Returns negative

* 1. **strchr(str, ch)** – Finds first occurrence of ch in str.

strchr("Programming", 'g'); // Returns "gamming"

* **THEORY EXERCISE:** Provide examples of when these functions are useful.

Code:

#include <stdio.h>

#include <string.h>

int main() {

    char username[20], password[20], email[50];

    // strcpy() - Setting default username

    strcpy(username, "Guest");

    // User Input

    printf("Enter your name: ");

    scanf("%s", username);

    printf("Enter your password: ");

    scanf("%s", password);

    printf("Enter your email: ");

    scanf("%s", email);

    // strlen() - Checking password length

    if (strlen(password) < 8) {

        printf("Error: Password too short!\n");

        return 1;

    }

    // strcat() - Creating a welcome message

    char greeting[50] = "Welcome, ";

    strcat(greeting, username);

    printf("%s\n", greeting);

    // strcmp() - Checking admin access

    if (strcmp(username, "admin") == 0) {

        printf("Admin access granted!\n");

    } else {

        printf("Regular user access.\n");

    }

    // strchr() - Checking for '@' in email

    if (strchr(email, '@') == NULL) {

        printf("Invalid email address!\n");

    } else {

        printf("Email verified!\n");

    }

    return 0;

}

Output:

Enter your name: Arsh

Enter your password: secret12

Enter your email: info@patelarsh.com

Welcome, Arsh

Regular user access.

Email verified!

* **LAB EXERCISE:** Write a C program that takes two strings from the user and concatenates them using strcat().

Code:

#include <stdio.h>

#include <string.h>

int main() {

    char str1[100], str2[100];  // Arrays to store input strings

    // Taking input from the user

    printf("Enter the first string: ");

    scanf("%s", str1);

    printf("Enter the second string: ");

    scanf("%s", str2);

    // Concatenating strings

    strcat(str1, str2);

    // Displaying the concatenated string

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

    return 0;

}

Output:

Enter the first string: Hello

Enter the second string: World

Concatenated String: HelloWorld

* **LAB EXERCISE:** Display the concatenated string and its length using strlen().

Code:

#include <stdio.h>

#include <string.h>

int main() {

    char str1[200], str2[100];  // Ensure str1 has enough space for concatenation

    // Taking input from the user

    printf("Enter the first string: ");

    scanf("%s", str1);

    printf("Enter the second string: ");

    scanf("%s", str2);

    // Concatenating strings

    strcat(str1, str2);

    // Displaying concatenated string and its length

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

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

    return 0;

}

Output:

Enter the first string: Hello

Enter the second string: World

Concatenated String: HelloWorld

Length of Concatenated String: 10

1. **Structures in C**

* **THEORY EXERCISE:** Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.

A structure groups related variables of different data types into a single unit.

Code:

This program demonstrates **declaration, initialization, and access** of structure members, including pointer usage.

#include <stdio.h>

#include <string.h>

// Define a structure

struct Student {

    char name[50];

    int age;

    float marks;

};

int main() {

    // Initializing structure

    struct Student s1 = {"John", 20, 85.5};

    struct Student s2;

    // Assigning values to structure members

    s2.age = 22;

    s2.marks = 90.0;

    strcpy(s2.name, "Alice");

    // Accessing structure members

    printf("Student 1: Name = %s, Age = %d, Marks = %.2f\n", s1.name, s1.age, s1.marks);

    printf("Student 2: Name = %s, Age = %d, Marks = %.2f\n", s2.name, s2.age, s2.marks);

    // Using pointer to access structure members

    struct Student \*ptr = &s1;

    printf("Using Pointer - Name: %s, Age: %d, Marks: %.2f\n", ptr->name, ptr->age, ptr->marks);

    return 0;

}

Output:

Student 1: Name = John, Age = 20, Marks = 85.50

Student 2: Name = Alice, Age = 22, Marks = 90.00

Using Pointer - Name: John, Age: 20, Marks: 85.50

* **LAB EXERCISE :** Write a C program that defines a structure to store a student's details (name, roll number, and marks).

Code:

#include <stdio.h>

#include <string.h>

// Define a structure for student details

struct Student {

    char name[50];

    int rollNumber;

    float marks;

};

int main() {

    struct Student s1;

    // Taking user input

    printf("Enter student name: ");

    scanf("%s", s1.name);

    printf("Enter roll number: ");

    scanf("%d", &s1.rollNumber);

    printf("Enter marks: ");

    scanf("%f", &s1.marks);

    // Displaying student details

    printf("\nStudent Details:\n");

    printf("Name: %s\n", s1.name);

    printf("Roll Number: %d\n", s1.rollNumber);

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

    return 0;

}

Output:

Enter student name: Alice

Enter roll number: 101

Enter marks: 89.5

Student Details:

Name: Alice

Roll Number: 101

Marks: 89.50

* **LAB EXERCISE:** Use an array of structures to store details of 3 students and print them.

Code:

#include <stdio.h>

#include <string.h>

// Define a structure for student details

struct Student {

    char name[50];

    int rollNumber;

    float marks;

};

int main() {

    struct Student students[3]; // Array of structures for 3 students

    // Taking input for each student

    for (int i = 0; i < 3; i++) {

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

        printf("Enter name: ");

        scanf("%s", students[i].name);

        printf("Enter roll number: ");

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

        printf("Enter marks: ");

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

        printf("\n");

    }

    // Displaying student details

    printf("Student Details:\n");

    for (int i = 0; i < 3; i++) {

        printf("Student %d - Name: %s, Roll Number: %d, Marks: %.2f\n",

               i + 1, students[i].name, students[i].rollNumber, students[i].marks);

    }

    return 0;

}

Output:

Enter details for Student 1:

Enter name: Alice

Enter roll number: 101

Enter marks: 89.5

Enter details for Student 2:

Enter name: Bob

Enter roll number: 102

Enter marks: 76.8

Enter details for Student 3:

Enter name: Charlie

Enter roll number: 103

Enter marks: 92.3

Student Details:

Student 1 - Name: Alice, Roll Number: 101, Marks: 89.50

Student 2 - Name: Bob, Roll Number: 102, Marks: 76.80

Student 3 - Name: Charlie, Roll Number: 103, Marks: 92.30

1. **File Handling in C**

* **THEORY EXERCISE:** Explain the importance of file handling in C.

**Importance of File Handling in C**

File handling enables permanent data storage, unlike temporary RAM storage.

**Why It’s Important?**

* + **Persistent Storage –** Saves data after program execution.
  + **Efficient Data Management –** Handles large data efficiently.
  + **Data Sharing –** Enables multiple programs to access data.
  + **Logging & Debugging –** Stores logs, reports, and errors.

**Common File Operations**

* + **fopen() –** Open file
  + **fprintf()/fscanf() –** Write/Read text data
  + **fwrite()/fread() –** Write/Read binary data
  + **fclose() –** Close file
* **THEORY EXERCISE:** Discusshow to perform file operations like opening, closing, reading, and writing files.

Code:

This program demonstrates **opening, writing, reading, and closing a file** in C.

#include <stdio.h>

int main() {

    FILE \*fp;

    char str[100];

    // Opening file in write mode and writing to it

    fp = fopen("file.txt", "w");

    if (fp == NULL) {

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

        return 1;

    }

    fprintf(fp, "Hello, this is a file handling example!");

    fclose(fp);  // Closing file after writing

    // Opening file in read mode and reading from it

    fp = fopen("file.txt", "r");

    if (fp == NULL) {

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

        return 1;

    }

    fgets(str, 100, fp);

    printf("File Content: %s\n", str);

    fclose(fp);  // Closing file after reading

    return 0;

}

Output:

File Content: Hello, this is a file handling example!

* **LAB EXERCISE:** Write a C program to create a file, write a string into it, close the file, then open the file again to read and display its contents.

Code:

#include <stdio.h>

int main() {

    FILE \*fp;

    char str[100];

    // Step 1: Create and open file in write mode

    fp = fopen("example.txt", "w");

    if (fp == NULL) {

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

        return 1;

    }

    // Step 2: Write a string to the file

    fprintf(fp, "Hello, this is a file handling example in C!");

    fclose(fp);  // Step 3: Close the file

    // Step 4: Open file in read mode

    fp = fopen("example.txt", "r");

    if (fp == NULL) {

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

        return 1;

    }

    // Step 5: Read and display file contents

    fgets(str, 100, fp);

    printf("File Content: %s\n", str);

    fclose(fp);  // Close the file

    return 0;

}

Output:

File Content: Hello, this is a file handling example in C!