Module2–Introduction to Programming

1.Overview of C Programming:

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

- C language was made in 1972 by Dennis Ritchie at Bell Labs to build the Unix operating system.

- It is the base of many other languages like C++, Java, and Python.

- C is fast, powerful, and gives full control over the computer's memory.

- It’s still used today in operating systems, embedded systems, and hardware programming.

- C is popular because it’s simple, portable, and helps learn programming basics easily.

In short: C is old but gold! Still useful, still powerful.

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.

1. Embedded Systems –

Like in TV remotes, washing machines, ACs – the logic inside is written in C language.

2. Operating Systems –

Operating systems like Windows and Linux are mostly built using C.

3. Game Development –

Old high-performance games and their engines were made in C.

Why is it still used?

C is fast, works close to the hardware, and gives full control over the system. That’s why it’s still important today.

2.SettingUpEnvironment:

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 Code Blocks.

Step 1: Install a C Compiler (GCC)

- Windows:

- Download MinGW or TDM-GCC.

- Install it and add the path to system environment variables if needed.

- Linux (Ubuntu):

- Open Terminal and run: sudo apt install gcc

- Mac:

- Open Terminal and run: xcode-select --install

---

Step 2: Install an IDE

Choose one IDE from below:

➤ DevC++:

- Download and install it from its official website.

- Comes with GCC compiler already included.

➤ VS Code:

- Install Visual Studio Code.

- Add C/C++ extension from Microsoft.

- Install GCC/MinGW separately.

- Configure tasks and launch files for compiling and running.

➤ Code::Blocks:

- Download the version that includes MinGW.

- Install and start coding right away.

LAB EXERCISE: Install a C compiler on your system and configure the IDE. Write your firstprogram to print "Hello, World!" and run it.

Step 1: Install C Compiler and IDE

- Download and install Code::Blocks (with MinGW included) or DevC++

- OR install VS Code + GCC (MinGW) + C/C++ extension

Step 2: Open IDE and create a new file

- Make a new C file and save it with .c extension

Step 3: Write your first program

c

#include<stdio.h>

main()

{

printf("hello world");

}

Output: Hello World

3. 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.

**🔹 1. Header Files**

Header files contain definitions of functions and macros. They are included at the top using #include.

c

#include <stdio.h> // Standard I/O functions

**🔹 2. Main Function**

Every C program must have a main() function. It is the **entry point** of the program.

c

int main() {

// code goes here

return 0;

}

* int indicates the return type.
* return 0; signals successful program execution.

**🔹 3. Comments**

Comments help make the code readable and are ignored by the compiler.

* **Single-line comment**:

c

// This is a single-line comment

* **Multi-line comment**:

c

/\* This is a

multi-line comment \*/

**🔹 4. Data Types**

Data types tell the compiler what kind of data is being used.

| **Data Type** | **Description** | **Example** |
| --- | --- | --- |
| int | Integer numbers | 1, 2, -5 |
| float | Floating point numbers | 3.14, -0.5 |
| char | Single characters | 'A', 'z' |
| double | Double precision float | 2.718281828 |

**🔹 5. Variables**

Variables are used to store data. You must declare them with a data type.

c

int age = 25;

float weight = 65.5;

char grade = 'A';

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

#include <stdio.h> // Header file

int main() {

// Constant declaration

const float PI = 3.14;

// Variable declarations

int age = 20;

float height = 5.8;

char grade = 'A';

// Output the values

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

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

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

printf("PI value: %.2f\n", PI); // Constant

return 0;

}

Output:

Age: 20

Height: 5.8

Grade: A

PI value: 3.14

4. 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

Used for basic math operations.

- + (Addition)

- - (Subtraction)

- \* (Multiplication)

- / (Division)

- % (Modulus - remainder)

Example: a + b

---

2. Relational Operators

Used to compare two values.

- == (Equal to)

- != (Not equal to)

- > (Greater than)

- < (Less than)

- >= (Greater than or equal to)

- <= (Less than or equal to)

Example: a > b

---

3. Logical Operators

Used to check multiple conditions.

- && (Logical AND)

- || (Logical OR)

- ! (Logical NOT)

Example: a > 5 && b < 10

---

4. Assignment Operators

Used to assign values.

- = (Assign)

- +=, -=, \*=, /=, %= (Short-hand operations)

Example: a += 5 is same as a = a + 5

---

5. Increment / Decrement Operators

Used to increase or decrease value by 1.

- ++ (Increment)

- -- (Decrement)

Example: a++ means a = a + 1

---

6. Bitwise Operators

Work on bits.

- & (AND)

- | (OR)

- ^ (XOR)

- ~ (NOT)

- << (Left shift)

- >> (Right shift)

Example: a & b

---

7. Conditional (Ternary) Operator

Shortcut for if-else.

- ? :

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

Example: a > b ? a : b

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.

#include <stdio.h>

int main() {

int num1, num2;

// Input two integers

printf("Enter two integers: ");

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

// Arithmetic Operations

printf("Addition: %d\n", num1 + num2);

printf("Subtraction: %d\n", num1 - num2);

printf("Multiplication: %d\n", num1 \* num2);

printf("Division: %d\n", num1 / num2);

printf("Division: Cannot divide by zero\n");

printf("\n");

// Relational Operations

printf("Is num1 equal to num2? %d\n", num1 == num2);

printf("Is num1 not equal to num2? %d\n", num1 != num2);

printf("Is num1 greater than num2? %d\n", num1 > num2);

printf("Is num1 less than num2? %d\n", num1 < num2);

printf("\n");

// Logical Operations

printf("num1 > 0 AND num2 > 0: %d\n", (num1 > 0) && (num2 > 0));

printf("num1 > 0 OR num2 > 0: %d\n", (num1 > 0) || (num2 > 0));

printf("NOT (num1 > 0): %d\n", !(num1 > 0));

return 0;

}

5. Control Flow Statements in C :

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

- if: Run code if condition true.

- if-else: Run one code if true, another if false.

- nested if: if-else inside another if-else for multiple checks.

- switch: Choose code to run based on exact value.

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 for January, 2 for February, etc.).

#include <stdio.h>

main()

{

int num, month;

printf("Enter a number: ");

scanf("%d", &num);

if (num % 2 == 0) {

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

} else {

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

}

}

#include <stdio.h>

main()

{

// Display month name using switch

int month;

printf("Enter month number (1 to 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.\n");

}

}

6. 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.

- While loop: Checks first, then runs. Use when you don’t know how many times it will run.

- For loop: Counts from start to end. Use when you know exactly how many times to repeat.

- Do-while loop: Runs first, then checks. Use when you want to run the code at least once.

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

#include <stdio.h>

int main()

{

int i;

// Using while loop

i = 1;

printf("While loop:\n");

while (i <= 10)

{

printf("%d ", i);

i++;

}

// Using for loop

printf("\n\nFor loop:\n");

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

{

printf("%d ", i);

}

// Using do-while loop

i = 1;

printf("\n\nDo-while loop:\n");

do

{

printf("%d ", i);

i++;

} while (i <= 10);

return 0;

}

7. Loop Control Statements:

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

- break: Stops the loop or switch immediately and jumps out.

Example: Stop a loop when a number is found.

- continue: Skips the current loop cycle and moves to the next one.

Example: Skip even numbers in a loop.

- goto: Jumps to a labeled part in the code (use carefully!).

Example: Jump to error handling.

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.

#include <stdio.h>

int main() {

int i;

// Using break to stop at 5

printf("Using break:\n");

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

{

if (i == 5)

{

break; // Stop the loop 5

}

printf("%d ", i);

}

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

// Using continue to skip 3

for (i = 1; i <= 5; i++)

{

if (i == 3)

{

continue; // Skip print 3

}

printf("%d ", i);

}

return 0;

}

8. Functions in C :

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

1. Function Declaration (Prototype):

Tells the compiler about the function name, return type, and parameters.

Example:

int add(int, int);

2. Function Definition:

Where we write the actual code of the function.

Example:

c

int add(int a, int b) {

return a + b;

}

3. Function Call:

Using the function in your code to run its task.

Example:

c

int result = add(5, 3);

printf("%d", result); // Output: 8

---

Summary:

- Declare function first (optional but good practice)

- Define function (write code)

- Call function whenever you want to use it

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

#include <stdio.h>

int factorial(int n);

int main()

{

int num;

printf("Enter a number: ");

scanf("%d", &num);

// Function

int result = factorial(num);

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

return 0;

}

// Function Definition

int factorial(int n)

{

int fact = 1;

for(int i = 1; i <= n; i++)

{

fact \*= i;

}

return fact;

}

9. Arrays in C :

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

🔹 1. One-Dimensional Array (1D):

Like a list or a line of elements.

Example:

c

int marks[5] = {90, 85, 70, 88, 95};

printf("%d", marks[2]); // Output: 70

Here, marks[2] gives the 3rd value (index starts at 0).

---

🔹 2. Multi-Dimensional Array (2D or more):

Like a table or matrix (rows & columns).

Example (2D Array):

c

int matrix[2][3] = {

{1, 2, 3},

{4, 5, 6}

};

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

LAB EXERCISE: Write a C program that stores 5 integers in a one-dimensional array and prints them. Extend this to handle a two-dimensional array (3x3 matrix) and calculate the sum of all elements.

#include <stdio.h>

int main()

{

// 1D Array

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

printf("1D Array elements:\n");

for(int i = 0; i < 5; i++)

{

printf("%d ", arr[i]);

}

// 2D Array

int matrix[3][3], sum = 0;

printf("\n\nEnter elements for 3x3 matrix:\n");

for(int i = 0; i < 3; i++)

{

for(int j = 0; j < 3; j++)

{

printf("Element %d %d: ", i, j);

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

sum += matrix[i][j];

}

}

printf("\n2D Array (Matrix):\n");

for(int i = 0; i < 3; i++)

{

for(int j = 0; j < 3; j++)

{

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

}

printf("\n");

}

printf("\nSum of all elements in matrix = %d\n", sum);

return 0;

}

10. Pointers in C :

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

✅ What is a Pointer?

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

---

✅ How to Declare a Pointer:

c

int \*ptr;

- \* means it’s a pointer.

- ptr will store the address of an integer variable.

---

✅ How to Initialize a Pointer:

c

int x = 10;

int \*ptr = &x;

- &x gives the address of x, which is stored in ptr.

✅ Why Are Pointers Important?

1. Memory Efficiency – Direct access to memory.

2. Dynamic Memory – Used in malloc(), calloc().

3. Arrays & Strings – Easy to handle with pointers.

4. Functions – Pass-by-reference using pointers.

5. Data Structures – Used in linked lists, trees, etc.

LAB EXERCISE: Write a C program to demonstrate pointer usage. Use a pointer to modify the value of a variable and print the result.

#include <stdio.h>

int main()

{

int num = 10;

int \*ptr;

ptr = &num;

printf("Before change: num = %d\n", num);

\*ptr = 20;

printf("After change: num = %d\n", num);

return 0;

}

11. Strings in C :

THEORY EXERCISE: Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.

1. \*strlen() – String Length\*

- Use: Finds the length of a string (without counting \0).

c

#include <stdio.h>

#include <string.h>

int main() {

char str[] = "Hello";

printf("Length = %lu", strlen(str)); // Output: 5

}

---

2. \*strcpy() – Copy String\*

- Use: Copies one string into another.

c

char src[] = "C Language";

char dest[20];

strcpy(dest, src); // dest now contains "C Language"

---

3. \*strcat() – Concatenate Strings\*

- Use: Adds (appends) one string to the end of another.

c

char str1[20] = "Hello ";

char str2[] = "World";

strcat(str1, str2); // str1 becomes "Hello World"

---

4. \*strcmp() – Compare Strings\*

- Use: Compares two strings.

- Returns:

- 0 if equal

- <0 if str1 < str2

- >0 if str1 > str2

c

strcmp("abc", "abc"); // returns 0

strcmp("abc", "def"); // returns negative

---

5. \*strchr() – Find Character\*

- Use: Finds first occurrence of a character in a string.

c

char \*ptr = strchr("Hello", 'l'); // points to first 'l'

LAB EXERCISE: Write a C program that takes two strings from the user and concatenates the musing strcat(). Display the concatenated string and its length using strlen().

#include <stdio.h>

#include <string.h>

int main()

{

char str1[100], str2[100];

printf("Enter first string: ");

fgets(str1, sizeof(str1), stdin);

str1[strcspn(str1, "\n")] = '\0'; // Remove newline

printf("Enter second string: ");

fgets(str2, sizeof(str2), stdin);

str2[strcspn(str2, "\n")] = '\0'; // Remove newline

strcat(str1, str2); // Concatenate strings

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

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

return 0;

}

12. Structures in C :

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

1. Declare a Structure

c

struct Student {

char name[50];

int age;

float marks;

};

---

2. Create & Initialize a Structure Variable

c

struct Student s1 = {"Rahul", 20, 88.5};

You can also assign values separately:

c

struct Student s2;

strcpy(s2.name, "Priya");

s2.age = 22;

s2.marks = 92.0;

---

3. Access Members

Use dot . operator:

c

printf("Name: %s", s1.name);

printf("Age: %d", s1.age);

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

LAB EXERCISE: Write a C program that defines a structure to store a student's details (name, roll number, and marks). Use an array of structures to store details of 3students and print them.

#include <stdio.h>

// Define structure

struct Student

{

char name[50];

int roll;

float marks;

};

int main()

{

struct Student s[3];

int i;

// Input data for 3 students

for(i = 0; i < 3; i++)

{

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

printf("Name: ");

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

printf("Roll Number: ");

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

printf("Marks: ");

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

}

// Print data

printf("\nStudent Details:\n");

for(i = 0; i < 3; i++)

{

printf("\nStudent %d\n", i + 1);

printf("Name: %s\n", s[i].name);

printf("Roll Number: %d\n", s[i].roll);

printf("Marks: %f\n", s[i].marks);

}

return 0;

}

13. File Handling in C :

THEORY EXERCISE: Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.

Why File Handling is Important:

- Stores data permanently

- Useful for reading/writing large amounts of data

- Helps in managing logs, reports, records, etc.

---

Basic File Operations:

1. Opening a File

c

FILE \*fp;

fp = fopen("data.txt", "r"); // modes: "r", "w", "a", etc.

2. Writing to a File

c

fp = fopen("data.txt", "w");

fprintf(fp, "Hello, file!");

fclose(fp);

3. Reading from a File

c

char ch;

fp = fopen("data.txt", "r");

while ((ch = fgetc(fp)) != EOF) {

putchar(ch);

}

fclose(fp);

4. Closing a File

c

fclose(fp);

---

Common File Modes:

- "r" – read

- "w" – write (overwrite)

- "a" – append

- "r+" – read/write

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.

#include <stdio.h>

int main()

{

FILE \*fptr;

char str[100];

// Write to file

fptr = fopen("example.txt", "w"); // open in write mode

if (fptr == NULL)

{

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

return 1;

}

printf("Enter a string to write into file: ");

fgets(str, sizeof(str), stdin);

fputs(str, fptr);

fclose(fptr);

printf("Data written and file closed.\n");

// Read from file

fptr = fopen("example.txt", "r"); // open in read mode

if (fptr == NULL)

{

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

return 1;

}

printf("Reading from file:\n");

while (fgets(str, sizeof(str), fptr) != NULL)

{

printf("%s", str);

}

fclose(fptr);

return 0;

}

EXTRA LAB EXERCISES FOR

IMPROVING PROGRAMMING LOGIC

1. Operators:

LAB EXERCISE 1: Simple Calculator:

• Write a C program that acts as a simple calculator. The program should take two numbers and an operator as input from the user and perform the respective operation (addition, subtraction, multiplication, division, or modulus) using operators.

#include <stdio.h>

int main() {

float num1, num2;

char op;

printf("Enter first number: ");

scanf("%f", &num1);

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

scanf(" %c", &op); // space before %c to handle newline

printf("Enter second number: ");

scanf("%f", &num2);

switch(op) {

case '+':

printf("Result = %.2f\n", num1 + num2);

break;

case '-':

printf("Result = %.2f\n", num1 - num2);

break;

case '\*':

printf("Result = %.2f\n", num1 \* num2);

break;

case '/':

if(num2 != 0)

printf("Result = %.2f\n", num1 / num2);

else

printf("Error: Cannot divide by zero.\n");

break;

case '%':

if((int)num2 != 0)

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

else

printf("Error: Cannot take modulus by zero.\n");

break;

default:

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

}

return 0;

}

LAB EXERCISE 2: Check Number Properties :

Write a C program that takes an integer from the user and checks the following using different operators:

Whether the number is even or odd.

Whether the number is positive, negative, or zero.

Whether the number is a multiple of both 3 and 5

#include <stdio.h>

int main()

{

int num;

printf("Enter an integer: ");

scanf("%d", &num);

// Even or Odd

if (num % 2 == 0)

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

else

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

// Positive, Negative or Zero

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");

// Multiple of both 3 and 5

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;

}

1. Control Statements LAB EXERCISE:

1: Grade Calculator:

Write a C program that takes the marks of a student as input and displays the corresponding grade based on the following conditions:

-> Marks > 90: Grade A o Marks > 75 and <= 90: Grade B

-> Marks > 50 and <= 75: Grade C

->Marks <= 50: Grade D

• Use if-else or switch statements for the decision-making process.

#include <stdio.h>

int main() {

int marks;

printf("Enter student marks (0-100): ");

scanf("%d", &marks);

if (marks > 90)

{

printf("Grade: A\n");

}

else if (marks > 75 && marks <= 90)

{

printf("Grade: B\n");

}

else if (marks > 50 && marks <= 75)

{

printf("Grade: C\n");

}

else if (marks <= 50)

{

printf("Grade: D\n");

}

else

{

printf("Invalid input. Please enter marks between 0 and 100.\n");

}

return 0;

}

LAB EXERCISE 2: Number Comparison

• Write a C program that takes three numbers from the user and determines:

o The largest number.

o The smallest number.

• Challenge: Solve the problem using both if-else and switch-case statements

#include <stdio.h>

int main()

{

int a, b, c;

printf("Enter three numbers: ");

scanf("%d %d %d", &a, &b, &c);

// Finding largest

if (a >= b && a >= c)

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

else if (b >= a && b >= c)

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

else

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

// Finding smallest

if (a <= b && a <= c)

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

else if (b <= a && b <= c)

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

else

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

return 0;

}

4.Loops:

LAB EXERCISE: Prime Number Check:

• Write a C program that checks whether a given number is a prime number or not using a for loop.

• Challenge: Modify the program to print all prime numbers between 1 and a given number.

#include <stdio.h>

int main()

{

int num, i, isPrime;

printf("Enter a number: ");

scanf("%d", &num);

// Check if num is prime

if (num <= 1)

{

printf("%d is not a prime number.\n", num);

}

else

{

isPrime = 1;

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

{

if (num % i == 0)

{

isPrime = 0;

break;

}

}

if (isPrime)

printf("%d is a prime number.\n", num);

else

printf("%d is not a prime number.\n", num);

}

// Challenge: Print all prime numbers from 1 to num

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

for (int j = 2; j <= num; j++)

{

isPrime = 1;

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

{

if (j % i == 0)

{

isPrime = 0;

break;

}

}

if (isPrime)

printf("%d ", j);

}

return 0;

}

LAB EXERCISE 2: Multiplication Table:

• Write a C program that takes an integer input from the user and prints its multiplication table using a for loop.

• Challenge: Allow the user to input the range of the multiplication table (e.g., from1 to N).

#include <stdio.h>

int main()

{

int num, range;

// Take number input

printf("Enter a number: ");

scanf("%d", &num);

// Take range input

printf("Enter range : ");

scanf("%d", &range);

// Print multiplication table

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

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

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

}

return 0;

}

LAB EXERCISE 3: Sum of Digits:

• Write a C program that takes an integer from the user and calculates the sum of its digits using a while loop.

• Challenge: Extend the program to reverse the digits of the number.

#include <stdio.h>

int main()

{

int num, digit, sum = 0, reverse = 0;

printf("Enter a number: ");

scanf("%d", &num);

int original = num; // Store original number for display

while (num != 0)

{

digit = num % 10; // Get last digit

sum += digit; // Add to sum

reverse = reverse \* 10 + digit; // Build reverse

num /= 10; // Remove last digit

}

printf("Sum of digits of %d = %d\n", original, sum);

printf("Reversed number = %d\n", reverse);

return 0;

}

1. Arrays:

LAB EXERCISE 1: Maximum and Minimum in Array:

• Write a C program that accepts 10 integers from the user and stores them in an array. The program should then find and print the maximum and minimum values in the array.

• Challenge: Extend the program to sort the array in ascending order.

#include <stdio.h>

int main()

{

int arr[10], i, j, temp;

int max, min;

// Input 10 numbers

printf("Enter 10 integers:\n");

for (i = 0; i < 10; i++)

{

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

}

// Find max and min

max = min = arr[0];

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

{

if (arr[i] > max)

max = arr[i];

if (arr[i] < min)

min = arr[i];

}

// Sort the array (ascending order)

for (i = 0; i < 9; i++)

{

for (j = i + 1; j < 10; j++)

{

if (arr[i] > arr[j])

{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

// Output

printf("Maximum value: %d\n", max);

printf("Minimum value: %d\n", min);

printf("Sorted array: ");

for (i = 0; i < 10; i++)

{

printf("%d ", arr[i]);

}

return 0;

}

LAB EXERCISE 2: Matrix Addition:

• Write a C program that accepts two 2x2 matrices from the user and adds them. Display the resultant matrix.

• Challenge: Extend the program to work with 3x3 matrices and matrix multiplication.

#include <stdio.h>

int main()

{

int i, j, k;

// 2x2 Matrices

int mat1[2][2], mat2[2][2], sum[2][2];

printf("Enter 1st 2x2 matrix:\n");

for (i = 0; i < 2; i++)

{

for (j = 0; j < 2; j++)

{

printf("mat1[%d][%d]: ", i, j);

scanf("%d", &mat1[i][j]);

}

}

printf("Enter 2nd 2x2 matrix:\n");

for (i = 0; i < 2; i++)

{

for (j = 0; j < 2; j++)

{

printf("mat2[%d][%d]: ", i, j);

scanf("%d", &mat2[i][j]);

}

}

// Adding 2x2 Matrices

for (i = 0; i < 2; i++)

for (j = 0; j < 2; j++)

sum[i][j] = mat1[i][j] + mat2[i][j];

printf("Sum of 2x2 matrices:\n");

for (i = 0; i < 2; i++)

{

for (j = 0; j < 2; j++)

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

printf("\n");

}

// 3x3 Matrix Multiplication

int a[3][3], b[3][3], result[3][3] = {0};

printf("\nEnter 1st 3x3 matrix:\n");

for (i = 0; i < 3; i++)

for (j = 0; j < 3; j++)

{

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

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

}

printf("Enter 2nd 3x3 matrix:\n");

for (i = 0; i < 3; i++)

for (j = 0; j < 3; j++)

{

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

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

}

// Multiplying 3x3 Matrices

for (i = 0; i < 3; i++)

for (j = 0; j < 3; j++)

for (k = 0; k < 3; k++)

result[i][j] += a[i][k] \* b[k][j];

printf("Product of 3x3 matrices:\n");

for (i = 0; i < 3; i++)

{

for (j = 0; j < 3; j++)

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

printf("\n");

}

return 0;

}

LAB EXERCISE 3: Sum of Array Elements

• Write a C program that takes N numbers from the user and stores them in an array. The program should then calculate and display the sum of all array elements.

• Challenge: Modify the program to also find the average of the numbers

#include <stdio.h>

int main()

{

int n, i;

float sum = 0, avg;

printf("Enter how many numbers you want to input: ");

scanf("%d", &n);

float arr[n];

printf("Enter %d numbers:\n", n);

for(i = 0; i < n; i++)

{

scanf("%f", &arr[i]);

sum += arr[i];

}

avg = sum / n;

printf("Sum = %f\n", sum);

printf("Average = %f\n", avg);

return 0;

}

1. Functions:

LAB EXERCISE 1: Fibonacci Sequence

• Write a C program that generates the Fibonacci sequence up to N terms using a recursive function.

• Challenge: Modify the program to calculate the Nth Fibonacci number using both iterative and recursive methods. Compare their efficiency.

#include <stdio.h>

// Recursive function to get Nth Fibonacci number

int fibonacciRecursive(int n)

{

if (n <= 1)

return n;

return fibonacciRecursive(n - 1) + fibonacciRecursive(n - 2);

}

// Iterative function to get Nth Fibonacci number

int fibonacciIterative(int n)

{

int a = 0, b = 1, c, i;

if (n == 0)

return a;

for (i = 2; i <= n; i++)

{

c = a + b;

a = b;

b = c;

}

return b;

}

int main()

{

int n, i;

printf("Enter number of terms: ");

scanf("%d", &n);

printf("\nFibonacci sequence using recursion:\n");

for (i = 0; i < n; i++)

{

printf("%d ", fibonacciRecursive(i));

}

printf("\n\n%dth Fibonacci (Recursive): %d", n, fibonacciRecursive(n - 1));

printf("\n%dth Fibonacci (Iterative): %d", n, fibonacciIterative(n - 1));

return 0;

}

LAB EXERCISE 2: Factorial Calculation:

• Write a C program that calculates the factorial of a given number using a function.

• Challenge: Implement both an iterative and a recursive version of the factorial function and compare their performance for large numbers

#include <stdio.h>

// Recursive factorial function

long long factorialRecursive(int n)

{

if (n == 0 || n == 1)

return 1;

else

return n \* factorialRecursive(n - 1);

}

// Iterative factorial function

long long factorialIterative(int n)

{

long long result = 1;

for (int i = 2; i <= n; i++)

{

result \*= i;

}

return result;

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

if (num < 0)

{

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

return 1;

}

printf("Factorial (Recursive) of %d = %d\n", num, factorialRecursive(num));

printf("Factorial (Iterative) of %d = %d\n", num, factorialIterative(num));

return 0;

}

LAB EXERCISE 3: Palindrome Check

• Write a C program that takes a number as input and checks whether it is a palindrome using a function.

• Challenge: Modify the program to check if a given string is a palindrome.

#include <stdio.h>

#include <string.h>

// Function to check if a number is a palindrome

int isNumberPalindrome(int num) {

int rev = 0, temp = num;

while (num != 0) {

rev = rev \* 10 + num % 10;

num /= 10;

}

return (rev == temp);

}

// Function to check if a string is a palindrome

int isStringPalindrome(char str[]) {

int i, len = strlen(str);

for (i = 0; i < len / 2; i++) {

if (str[i] != str[len - 1 - i])

return 0;

}

return 1;

}

int main() {

int number;

char text[100];

// Input for number palindrome

printf("Enter a number: ");

scanf("%d", &number);

if (isNumberPalindrome(number))

printf("Number is Palindrome\n");

else

printf("Number is Not Palindrome\n");

// Input for string palindrome

printf("Enter a word: ");

scanf("%s", text);

if (isStringPalindrome(text))

printf("Word is Palindrome\n");

else

printf("Word is Not Palindrome\n");

return 0;

}

1. Strings:

LAB EXERCISE 1: String Reversal

• Write a C program that takes a string as input and reverses it using a function.

• Challenge: Write the program without using built-in string handling functions.

#include <stdio.h>

void reverse(char s[])

{

int i = 0, j;

while (s[i]) i++; // Length count

for (j = 0, i--; j < i; j++, i--)

{

char t = s[j];

s[j] = s[i];

s[i] = t;

}

}

int main()

{

char str[100];

printf("Enter a string: ");

scanf("%[^\n]", str); // Take full line input

reverse(str);

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

return 0;

}

LAB EXERCISE 2: Count Vowels and Consonants

• Write a C program that takes a string from the user and counts the number of vowels and consonants in the string.

• Challenge: Extend the program to also count digits and special characters.

#include <stdio.h>

int main()

{

char str[100];

int vowels = 0, consonants = 0, digits = 0, special = 0;

printf("Enter a string: ");

scanf(" %[^\n]", str); // Input with spaces

for (int i = 0; str[i] != '\0'; i++)

{

char ch = str[i];

if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))

{

ch = (ch >= 'A' && ch <= 'Z') ? ch + 32 : ch; // Convert to lowercase

if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')

vowels++;

else

consonants++;

}

else if (ch >= '0' && ch <= '9')

digits++;

else if (ch != ' ')

special++;

}

printf("Vowels: %d\nConsonants: %d\nDigits: %d\nSpecial Characters: %d\n", vowels, consonants, digits, special);

return 0;

}

LAB EXERCISE 3: Word Count

• Write a C program that counts the number of words in a sentence entered by the user.

• Challenge: Modify the program to find the longest word in the sentence.

#include <stdio.h>

#include <string.h>

int main()

{

char str[200], word[50], longest[50] = "";

int i = 0, j = 0, maxLen = 0, count = 0;

printf("Enter a sentence: ");

fgets(str, sizeof(str), stdin);

while (str[i] != '\0')

{

if (str[i] != ' ' && str[i] != '\n')

{

word[j++] = str[i];

} else {

if (j > 0) {

word[j] = '\0';

count++;

if (j > maxLen)

{

maxLen = j;

strcpy(longest, word);

}

j = 0;

}

}

i++;

}

// For last word if no space after it

if (j > 0)

{

word[j] = '\0';

count++;

if (j > maxLen) strcpy(longest, word);

}

printf("Total words: %d\n", count);

printf("Longest word: %s\n", longest);

return 0;

}

Extra Logic Building Challenges

Lab Challenge 1: Armstrong Number

• Write a C program that checks whether a given number is an Armstrong number or not (e.g., 153 = 1^3 + 5^3 + 3^3).

• Challenge: Write a program to find all Armstrong numbers between 1 and 1000.

#include <stdio.h>

int isArmstrong(int num)

{

int original = num, sum = 0;

while (num > 0)

{

int digit = num % 10;

sum += digit \* digit \* digit;

num /= 10;

}

return (sum == original);

}

int main() {

int num;

// Check ek number

printf("Enter a number: ");

scanf("%d", &num);

if (isArmstrong(num))

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

else

printf("%d is not an Armstrong number.\n", num);

// 1 se 1000 tak ke Armstrong numbers

printf("Armstrong numbers from 1 to 1000:\n");

for (int i = 1; i <= 1000; i++)

{

if (isArmstrong(i))

printf("%d ", i);

}

return 0;

}

Lab Challenge 2: Pascal’s Triangle

• Write a C program that generates Pascal’s Triangle up to N rows using loops.

• Challenge: Implement the same program using a recursive function.

#include <stdio.h>

int combination(int n, int r)

{

if (r == 0 || r == n)

return 1;

return combination(n - 1, r - 1) + combination(n - 1, r);

}

int main() {

int n;

printf("Enter number of rows: ");

scanf("%d", &n);

for (int i = 0; i < n; i++)

{

for (int j = 0; j <= i; j++)

{

printf("%d ", combination(i, j));

}

printf("\n");

}

return 0;

}

Lab Challenge 3: Number Guessing Game

• Write a C program that implements a simple number guessing game. The program should generate a random number between 1 and 100, and the user should guess the number within a limited number of attempts.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main() {

int number, guess, tries = 0;

srand(time(0)); // Random number seed

number = rand() % 100 + 1; // 1 se 100 ke beech number

printf("1 se 100 ke beech number guess karo.\n");

while (tries < 7) {

printf("Apni guess do: ");

scanf("%d", &guess);

tries++;

if (guess == number) {

printf("Sahi guess! Tumne %d tries me laga liya.\n", tries);

break;

}

else if (guess < number) {

printf("Thoda bada socho.\n");

}

else {

printf("Thoda chhota socho.\n");

}

}

if (tries == 7 && guess != number)

printf("Koshish khatam! Number tha %d.\n", number);

return 0;

}