**Module 2 – Introduction to Programming**

**1.Overview of C Programming**

**THEORY EXERCISE 1:**

Q1.Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.

Ans: **Essay: The History and Evolution of C Programming Language**

The C programming language stands as one of the most influential and enduring languages in the history of computer science. Developed in the early 1970s, C not only laid the foundation for many modern programming languages but also played a critical role in the development of operating systems, embedded systems, and application software. Its simplicity, efficiency, and flexibility continue to make it relevant even today.

**History and Development**

The history of C can be traced back to the late 1960s at Bell Labs, where Dennis Ritchie and Ken Thompson were working on developing the Unix operating system. At that time, most system programming was done in assembly language, which was hardware-specific and difficult to maintain. Thompson initially created the B programming language, which was derived from BCPL (Basic Combined Programming Language), but B lacked certain features needed for system-level programming.

Recognizing these limitations, Dennis Ritchie developed the C programming language in 1972. C was designed to provide low-level access to memory, a simple set of keywords, and clean style while being more structured and readable than assembly language. With the rewriting of the Unix operating system in C, the language quickly proved its efficiency and portability, helping it gain widespread popularity.

**Evolution and Standardization**

Over time, C underwent several revisions and improvements. In 1983, the American National Standards Institute (ANSI) began the process of standardizing C to ensure consistency across different compilers and systems. The result was **ANSI C**, or **C89**, which became the foundation of modern C programming.

Later updates followed:

* **C99 (1999):** Introduced new data types like long long int, inline functions, and new library functions.
* **C11 (2011):** Added features like multithreading support, better Unicode handling, and improved safety.
* **C17 (2017):** A bug-fix version without major changes.
* **C23 (expected in 2023):** Aims to introduce improved language features and better interoperability with C++.

Each update has sought to make the language more modern while preserving its core philosophy of simplicity and performance.

**LAB EXERCISE 1:**

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

Ans: **1. Embedded Systems**

**Example:** **Automotive Control Systems (e.g., Engine Control Units - ECUs)**

* **Usage:** C is widely used in embedded systems such as those found in automobiles, medical devices, and consumer electronics. In automotive systems, C is used to program microcontrollers that manage fuel injection, braking, climate control, and other vital operations.
* **Why C?**
  + It offers **direct hardware access** and **low-level memory management**, which is essential for controlling physical components.
  + C code can be **highly optimized**, ensuring **fast execution** and **low power consumption**, both critical for embedded applications.
  + C programs have a **small footprint**, which fits the limited memory and processing power of embedded devices.

**2. Operating Systems**

**Example:** **Unix/Linux Operating Systems**

* **Usage:** The majority of modern operating systems, including Unix, Linux, and even parts of Windows, are written in C. The Linux kernel, which powers a wide variety of devices from servers to smartphones (via Android), is almost entirely written in C.
* **Why C?**
  + C allows **fine-grained control over system resources**, memory, and processor instructions.
  + It provides the **performance** and **efficiency** needed for managing hardware-level tasks such as memory management, file systems, and device drivers.
  + C's **portability** allows the operating system to run on different hardware platforms with minimal changes.

**3. Game Development**

**Example:** **Doom (1993) and Modern Game Engines (e.g., Unreal Engine - core components)**

* **Usage:** Many early and performance-critical parts of modern games are written in C. For example, classic games like *Doom* were originally written in C. Even today, performance-critical components like physics engines, rendering engines, and input handling in game engines often use C or C++.
* **Why C?**
  + C offers **high performance**, which is crucial for real-time game applications that demand fast response times and smooth graphics rendering.
  + Game engines need **efficient memory usage and direct access to hardware** like graphics cards and controllers.
  + C provides a good balance between **low-level control** and **development efficiency**, making it ideal for core engine development.

**2. Setting Up Environment**

**THEORY EXERCISE 2:**

Q2. Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like Dev C++, VS Code, or Code Blocks.

Ans: **Option 1: Install GCC Compiler (Windows using MinGW)**

**Step 1: Download MinGW**

* Visit: https://osdn.net/projects/mingw/releases/
* Download the mingw-get-setup.exe installer.

**Step 2: Install MinGW**

* Run the installer.
* In the installer:
  + Select mingw32-gcc-g++, mingw32-gcc-objc, and mingw32-base.
  + Click **Installation → Apply Changes**.

**Step 3: Set Environment Variable**

* Go to Control Panel → System → Advanced system settings → Environment Variables.
* Under **System variables**, find and edit Path.
* Add this path: C:\MinGW\bin

**Step 4: Verify GCC Installation**

* Open **Command Prompt** and type:

bash

CopyEdit

gcc --version

* You should see the installed version of GCC.

**Option 2: Set Up an IDE**

**1. Dev-C++ (Beginner-Friendly)**

**Steps:**

1. Download from: https://sourceforge.net/projects/orwelldevcpp/
2. Install and open Dev-C++.
3. Go to **File → New → Source File**, and write your C code.
4. Click **Execute → Compile & Run**.

Dev-C++ comes bundled with GCC, so no separate compiler installation is needed.

**2. Code::Blocks (With Built-in Compiler Option)**

**Steps:**

1. Download from: https://www.codeblocks.org/downloads/
2. Choose the version with **MinGW compiler** included (codeblocks-20.03mingw-setup.exe).
3. Install Code::Blocks and open it.
4. Go to **File → New → Project → Console Application**.
5. Select **C language**, name your project, and start coding.
6. Click **Build and Run**.

**3. Visual Studio Code (VS Code - Professional Setup)**

**Steps:**

1. Download and install:
   * VS Code: <https://code.visualstudio.com/>
   * GCC via **MinGW** (see Option 1 above)
2. **Install the C/C++ Extension** in VS Code:
   * Open VS Code.
   * Go to **Extensions** (Ctrl+Shift+X) and install **C/C++ by Microsoft**.
3. **Set Up Tasks for Build and Run:**
   * Create a .c file.
   * Add a tasks.json to automate compilation.
   * Press Ctrl+Shift+B to build.
4. **Run Your Program:**
   * Use **Terminal → Run Build Task**.
   * Use ./a.exe in the terminal to run the program.

**Summary Table**

| **IDE** | **Compiler Needed?** | **Suitable For** | **Notes** |
| --- | --- | --- | --- |
| Dev-C++ | No (included) | Beginners | Lightweight, easy to use |
| Code::Blocks | No (if you choose MinGW version) | Students & Intermediate users | IDE + Compiler in one download |
| VS Code | Yes (Install GCC separately) | Advanced users, professionals | Highly customizable, modern interface |

**Final Tip**

After setup, create a simple C program to test:

c

CopyEdit

#include <stdio.h>

int main() {

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

return 0;

}

**LAB EXERCISE 2:**

2 .Install a C compiler on your system and configure the IDE. Write your first program to print "Hello, World!" and run it.

Ans: **Step-by-Step Instructions**

**1. Install GCC Compiler (via MinGW)**

**➤ Download & Install:**

1. Go to: https://osdn.net/projects/mingw/releases/
2. Download: mingw-get-setup.exe
3. Run the installer.

**➤ During Installation:**

* Select the following packages:
  + mingw32-gcc-g++
  + mingw32-base
* Click **Installation → Apply Changes**

**➤ Add MinGW to System PATH:**

1. Open **System Properties → Environment Variables**
2. Find Path under "System Variables", click **Edit**
3. Add:

makefile

C:\MinGW\bin

**➤ Verify Installation:**

Open **Command Prompt** and type:

bash

gcc --version

If installed correctly, you’ll see the GCC version.

**2. Install and Set Up IDE**

**Option A: Use Code::Blocks (With Compiler)**

1. Download from: https://www.codeblocks.org/downloads/
2. Choose: codeblocks-20.03mingw-setup.exe
3. Install it and open Code::Blocks
4. Go to **File → New → Project → Console Application**
5. Select **C Language**, click **Next**
6. Give your project a name and location, click **Finish**

**3. Write Your First Program**

Inside the IDE (e.g., Code::Blocks or Dev-C++), write the following code:

#include <stdio.h>

int main() {

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

return 0;

}

**4. Compile and Run the Program**

* In Code::Blocks or Dev-C++, press **F9** or click **Build & Run**
* You will see the output:

Hello, World!

**Output Example**

If everything is set up correctly, your terminal or output window will show:

CopyEdit

Hello, World!

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

**THEORY EXERCISE 3 :**

Q3. Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.

Ans: **Basic Structure of a C Program**

#include <stdio.h> // Header file

// This is a single-line comment

/\*

This is a multi-line comment

\*/

int main() { // Main function - entry point of the program

// Variable declaration

int age = 20;

float height = 5.9;

char grade = 'A';

// Output statement

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

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

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

return 0; // Exit status

}

**1. Header Files**

**➤ Example:**

c

CopyEdit

#include <stdio.h>

* #include is a **preprocessor directive**.
* <stdio.h> is a **standard library** that provides functions like printf() and scanf().
* Other common headers:
  + <math.h> – Math functions
  + <stdlib.h> – Memory allocation and conversion

**2. Comments**

**➤ Single-line:**

// This is a comment

**➤ Multi-line:**

/\* This is

a multi-line comment \*/

* Comments are **ignored by the compiler**.
* They are used to **explain code** and **improve readability**.

**3. Main Function**

**➤ Example:**

c

CopyEdit

int main() {

// code here

return 0;

}

* The program **starts execution from main()**.
* int means the function returns an integer value (usually 0 for success).
* return 0; tells the OS that the program executed successfully.

**4. Data Types**

| **Data Type** | **Description** | **Example** |
| --- | --- | --- |
| int | Integer numbers | int age = 18; |
| float | Decimal numbers | float pi = 3.14; |
| char | Single characters | char grade = 'A'; |
| double | Double-precision float | double value = 10.4567; |

**5. Variables**

* **Variables** are named memory locations used to store data.
* Syntax:

data\_type variable\_name = value;

**➤ Example:**

int score = 90;

float average = 72.5;

char initial = 'Y';

**Example Program with All Elements**

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

// This program demonstrates basic elements of C

int main() {

int age = 25; // Integer variable

float salary = 5000; // Float variable

char grade = 'B'; // Character variable

// Printing variables

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

printf("Salary: %.2f\n", salary);

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

return 0;

}

**Output:**

Age: 25

Salary: 5000.00

Grade: B

**LAB EXERCISE 3 :**

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

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

// This program demonstrates variables, constants, and different data types

int main() {

// Constant declaration

const float PI = 3.14159;

// Variable declarations

int age = 21; // Integer variable

float height = 5.8; // Float variable

char grade = 'A'; // Character variable

// Displaying values

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

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

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

// Displaying constant value

printf("Value of constant PI: %.5f\n", PI);

return 0; // Exit the program

}

**Output:**

Age: 21 years

Height: 5.8 feet

Grade: A

Value of constant PI: 3.14159

**4.Operators in C**

**THEORY EXERCISE 4 :**

Q4. Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.

Ans:  **1. Arithmetic Operators**

**Used for basic mathematical operations.**

| **Operator** | **Description** | **Example (a = 10, b = 3)** | **Result** |
| --- | --- | --- | --- |
| + | Addition | a + b | 13 |
| - | Subtraction | a - b | 7 |
| \* | Multiplication | a \* b | 30 |
| / | Division | a / b | 3 |
| % | Modulus (remainder) | a % b | 1 |

**2. Relational (Comparison) Operators**

**Used to compare two values; result is either 1 (true) or 0 (false).**

| **Operator** | **Description** | **Example (a = 10, b = 3)** | **Result** |
| --- | --- | --- | --- |
| == | Equal to | a == b | 0 |
| != | Not equal to | a != b | 1 |
| > | Greater than | a > b | 1 |
| < | Less than | a < b | 0 |
| >= | Greater or equal | a >= b | 1 |
| <= | Less or equal | a <= b | 0 |

**3. Logical Operators**

**Used to combine multiple conditions.**

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| && | Logical AND | (a > 5 && b < 5) | 1 |
| ` |  | ` | Logical OR |
| ! | Logical NOT | !(a > b) | 0 |

**4. Assignment Operators**

**Used to assign or update values in variables.**

| **Operator** | **Description** | **Example (a = 10)** | **Meaning** |
| --- | --- | --- | --- |
| = | Assign | a = 5 | Set a to 5 |
| += | Add and assign | a += 2 | a = a + 2 |
| -= | Subtract and assign | a -= 3 | a = a - 3 |
| \*= | Multiply and assign | a \*= 4 | a = a \* 4 |
| /= | Divide and assign | a /= 2 | a = a / 2 |
| %= | Modulo and assign | a %= 3 | a = a % 3 |

**5. Increment/Decrement Operators**

**Used to increase or decrease a value by 1.**

| Operator

Top of Form

Tools

Bottom of Form

**LAB EXERCISE 4 :**

1. Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results.

Ans: #include <stdio.h>

int main() {

int a, b;

// Input from the user

printf("Enter first integer: ");

scanf("%d", &a);

printf("Enter second integer: ");

scanf("%d", &b);

// Arithmetic operations

printf("\n--- Arithmetic 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 is not allowed.\n");

}

// Relational operations

printf("\n--- Relational 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("\n--- Logical Operations ---\n");

printf("(%d != 0) && (%d != 0) : %d\n", a, b, (a != 0) && (b != 0));

printf("(%d != 0) || (%d != 0) : %d\n", a, b, (a != 0) || (b != 0));

printf("!(%d != 0) : %d\n", a, !(a != 0));

return 0;

}

**Output:**

Enter first integer: 10

Enter second integer: 5

--- Arithmetic Operations ---

10 + 5 = 15

10 - 5 = 5

10 \* 5 = 50

10 / 5 = 2

10 % 5 = 0

--- Relational Operations ---

10 == 5 : 0

10 != 5 : 1

10 > 5 : 1

10 < 5 : 0

10 >= 5 : 1

10 <= 5 : 0

--- Logical Operations ---

(10 != 0) && (5 != 0) : 1

(10 != 0) || (5 != 0) : 1

!(10 != 0) : 0

1. **Control Flow Statements in C**

**THEORY EXERCISE 5 :**

Q5. Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.

Ans: **1. if Statement**

**➤ Used to execute a block of code if a condition is true.**

**Syntax:**

if (condition) {

// Code to execute if condition is true

}

**Example:**

int num = 10;

if (num > 0) {

printf("Number is positive.\n");

}

**2. if-else Statement**

**➤ Provides an alternative block if the condition is false.**

**Syntax:**

if (condition) {

// Code if condition is true

} else {

// Code if condition is false

}

**Example:**

int age = 17;

if (age >= 18) {

printf("You are eligible to vote.\n");

} else {

printf("You are not eligible to vote.\n");

}

**3. Nested if-else Statement**

**➤ if-else inside another if-else. Useful for multiple conditions.**

**Syntax:**

if (condition1) {

// Code if condition1 is true

} else {

if (condition2) {

// Code if condition2 is true

} else {

// Code if all conditions are false

}

}

**Example:**

int marks = 75;

if (marks >= 90) {

printf("Grade A\n");

} else if (marks >= 75) {

printf("Grade B\n");

} else if (marks >= 60) {

printf("Grade C\n");

} else {

printf("Fail\n");

}

**4. switch Statement**

**➤ Used to select one block of code to run from many options. Works with integer and char values.**

**Syntax:**

switch (expression) {

case value1:

// Code for value1

break;

case value2:

// Code for value2

break;

...

default:

// Code if no match

}

**Example:**

int day = 3;

switch (day) {

case 1:

printf("Monday\n");

break;

case 2:

printf("Tuesday\n");

break;

case 3:

printf("Wednesday\n");

break;

default:

printf("Invalid day\n");

}

**LAB EXERCISE 5 :**

5. 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.).

Ans: #include <stdio.h>

int main() {

int number, month;

// Step 1: Check Even or Odd using if-else

printf("Enter a number to check even or odd: ");

scanf("%d", &number);

if (number % 2 == 0) {

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

} else {

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

}

// Step 2: Display Month Name using switch

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

scanf("%d", &month);

printf("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");

}

return 0;

}

**Output :**

Enter a number to check even or odd: 9

9 is Odd.

Enter a number (1 to 12) for a month: 4

Month: April

1. **Looping in C**

**THEORY EXERCISE 6 :**

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

Ans: **1. while Loop**

**➤ Definition:**

Executes a block of code **as long as the condition is true**.  
The condition is checked **before the loop body runs**.

**Syntax:**

while (condition) {

// code block

}

**Example:**

int i = 1;

while (i <= 5) {

printf("%d ", i);

i++;

}

**Best Use Case:**

* When the **number of iterations is unknown** in advance.
* Example: reading input until the user enters 0.

**2. for Loop**

**➤ Definition:**

A loop with a **counter**, typically used when the number of iterations is **known or defined**.

**Syntax:**

for (initialization; condition; update) {

// code block

}

**Example:**

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

printf("%d ", i);

}

**Best Use Case:**

* When you **know exactly how many times** to repeat.
* Example: printing numbers 1 to 100, iterating over arrays.

**3. do-while Loop**

**➤ Definition:**

**Always executes the loop body at least once**, then continues **as long as the condition is true**.

**Syntax:**

do {

// code block

} while (condition);

**Example:**

int i = 1;

do {

printf("%d ", i);

i++;

} while (i <= 5);

**LAB EXERCISE 6 :**

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

Ans: #include <stdio.h>

int main() {

int i;

// Using while loop

printf("Using while loop:\n");

i = 1;

while (i <= 10) {

printf("%d ", i);

i++;

}

// New line for clarity

printf("\n\n");

// Using for loop

printf("Using for loop:\n");

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

printf("%d ", i);

}

// New line for clarity

printf("\n\n");

// Using do-while loop

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

i = 1;

do {

printf("%d ", i);

i++;

} while (i <= 10);

// Final newline

printf("\n");

return 0;

}

**Output:**

Using while loop:

1 2 3 4 5 6 7 8 9 10

Using for 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 7:**

Q7.Explain the use of break, continue, and goto statements in C. Provide examples of each.

Ans: **1. break Statement**

**➤ Use:**

Exits **immediately** from a loop (for, while, do-while) or a switch statement.

**Example (in a loop):**

#include <stdio.h>

int main() {

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

if (i == 5)

break; // Exit loop when i is 5

printf("%d ", i);

}

return 0;

}

**Output:**

1 2 3 4

break is useful when you want to **stop the loop early**.

**2. continue Statement**

**➤ Use:**

**Skips** the current iteration and continues with the **next one**.

**Example (in a loop):**

#include <stdio.h>

int main() {

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

if (i == 3)

continue; // Skip when i is 3

printf("%d ", i);

}

return 0;

}

**Output:**

1 2 4 5

continue is useful when you want to **skip specific conditions** but keep the loop running.

**3. goto Statement**

**➤ Use:**

**Jumps** to a labeled statement in the program.  
**Not recommended** for regular use—it can make code **hard to read and debug**.

**Syntax:**

goto label\_name;

// ...

label\_name:

// statements

**Example:**

#include <stdio.h>

int main() {

int i = 1;

start:

if (i <= 5) {

printf("%d ", i);

i++;

goto start; // Jumps back to the label

}

return 0;

}

**Output:**

1 2 3 4 5

**LAB EXERCISE 7:**

7.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.

Ans: #include <stdio.h>

int main() {

printf("Printing numbers until 5 (using break):\n");

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

if (i == 5)

break; // Exit loop when i is 5

printf("%d ", i);

}

return 0;

}

**Output:**

Printing numbers until 5 (using break):

1 2 3 4

**Continue**

#include <stdio.h>

int main() {

printf("Printing numbers 1 to 10 (skip 3 using continue):\n");

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

if (i == 3)

continue; // Skip the iteration when i is 3

printf("%d ", i);

}

return 0;

}

**Output:**

Printing numbers 1 to 10 (skip 3 using continue):

12 4 5 6 7 8 9 10

1. **Functions in C**

**THEORY EXERCISE 8 :**

Q8. What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.

Ans: **Types of Functions in C**

1. **Library Functions** – Built-in (e.g., printf(), scanf(), sqrt())
2. **User-defined Functions** – Created by the programmer

**Components of a Function**

**1. Function Declaration (Prototype)**

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

return\_type function\_name(parameter\_list);

**Example:**

int add(int a, int b); // Declaration

**2. Function Definition**

The actual body of the function where the task is written.

return\_type function\_name(parameter\_list) {

// function body

}

**Example:**

int add(int a, int b) {

return a + b;

}

**3. Function Call**

Executes the function from main() or another function.

int result = add(5, 3); // Calling the function

**Complete Example Program**

#include <stdio.h>

// Function Declaration

int add(int, int);

int main() {

int x = 10, y = 5;

// Function Call

int result = add(x, y);

printf("Sum: %d\n", result);

return 0;

}

// Function Definition

int add(int a, int b) {

return a + b;

}

**Output:**

Sum: 15

**LAB EXERCISE 8 :**

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

Ans: #include <stdio.h>

// Function declaration

long long factorial(int n);

int main() {

int number;

// User input

printf("Enter a positive integer: ");

scanf("%d", &number);

// Validate input

if (number < 0) {

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

} else {

// Function call

long long result = factorial(number);

printf("Factorial of %d = %lld\n", number, result);

}

return 0;

}

// Function definition

long long factorial(int n) {

long long fact = 1;

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

fact \*= i;

}

return fact;

}

**Sample Output :**

Enter a positive integer: 5

Factorial of 5 = 120

1. **Arrays in C**

**THEORY EXERCISE 9 :**

Q9. Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.

Ans: **What is an Array in C?**

An **array** is a **collection of elements of the same data type** stored in **contiguous memory locations**.  
Each element in an array is accessed using an **index**, starting from 0.

**Why Use Arrays?**

* To store **multiple values** of the same type (e.g., 10 integers)
* Easy to **process using loops**
* Efficient use of memory and cleaner code

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

A simple linear collection of elements.

**➤ Syntax:**

data\_type array\_name[size];

**➤ Example:**

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

**➤ Access Elements:**

printf("%d", numbers[2]); // prints 30

**➤ Loop Through 1D Array:**

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

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

}

**2. Multi-Dimensional Array**

Used to represent **matrices** or **tables**. The most common is a **2D array**.

**➤ Syntax:**

data\_type array\_name[rows][columns];

**➤ Example:**

int matrix[2][3] = {

{1, 2, 3},

{4, 5, 6}

};

**➤ Access Elements:**

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

**➤ Loop Through 2D Array:**

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

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

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

}

printf("\n");

}

**LAB EXERCISE 9 :**

9.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.

Ans: #include <stdio.h>

int main() {

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

printf("Elements in 1D array:\n");

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

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

}

printf("\n\n");

int matrix[3][3];

int sum = 0;

// Taking input for the 2D array

printf("Enter elements for a 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]; // Add to total sum

}

}

// Displaying the 2D array

printf("\nMatrix:\n");

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

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

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

}

printf("\n");

}

// Displaying the sum

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

return 0;

}

**Output :**

Elements in 1D array:

10 20 30 40 50

Enter elements for a 3x3 matrix:

Element [0][0]: 1

Element [0][1]: 2

Element [0][2]: 3

Element [1][0]: 4

Element [1][1]: 5

Element [1][2]: 6

Element [2][0]: 7

Element [2][1]: 8

Element [2][2]: 9

Matrix:

1 2 3

4 5 6

7 8 9

Sum of all elements = 45

**10. Pointers in C**

**THEORY EXERCISE 10:**

Q10. Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

Ans: **What Are Pointers in C?**

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

Instead of holding a direct value (like 10), a pointer holds the **location** (address) in memory where that value is stored.

**Why Are Pointers Important in C?**

Pointers are powerful and essential in C because they:

* Allow **dynamic memory allocation**
* Enable **pass-by-reference** to functions (efficient data manipulation)
* Support efficient **array and string processing**
* Are used in building **complex data structures** (e.g., linked lists, trees)
* Enable **system-level programming** (e.g., accessing hardware)

**How to Declare and Initialize Pointers**

**Declaration:**

data\_type \*pointer\_name;

Example:

int \*ptr; // Pointer to an int

**Initialization:**

int x = 10;

ptr = &x; // ptr holds the address of x

**Example: Pointer Basics**

#include <stdio.h>

int main() {

int num = 25;

int \*p; // Declare pointer

p = &num; // Initialize pointer with address of num

printf("Value of num: %d\n", num); // 25

printf("Address of num: %p\n", &num); // e.g. 0x7ffee5e4c8cc

printf("Value of p (address): %p\n", p); // same as &num

printf("Value at address p (using \*p): %d\n", \*p); // 25

return 0;

}

**LAB EXERCISE 10 :**

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

Ans: #include <stdio.h>

int main() {

int num = 10; // Declare an integer variable

int \*ptr; // Declare a pointer to int

ptr = &num; // Assign address of num to ptr

// Before modification

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

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

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

printf("Value at ptr (i.e., \*ptr): %d\n", \*ptr);

// Modify the value of num using pointer

\*ptr = 25;

// After modification

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

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

printf("Value at ptr (i.e., \*ptr): %d\n", \*ptr);

return 0;

}

**Output:**

Original value of num: 10

Address of num: 0x7ffee3e92a4c

Pointer ptr holds address: 0x7ffee3e92a4c

Value at ptr (i.e., \*ptr): 10

After modifying value through pointer:

New value of num: 25

Value at ptr (i.e., \*ptr): 25

**11. Strings in C**

**THEORY EXERCISE 11 :**

Q11.Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.

Ans: **1. strlen(): Get the Length of a String**

**Syntax:**

size\_t strlen(const char \*str);

**Description:**

Returns the **number of characters** in the string (excluding the null '\0').

**Example:**

#include <stdio.h>

#include <string.h>

int main() {

char name[] = "Alice";

printf("Length: %lu\n", strlen(name)); // Output: 5

return 0;

}

**Use Case: Input validation, string size calculations.**

**2. strcpy(): Copy One String into Another**

**Syntax:**

char \*strcpy(char \*dest, const char \*src);

**Description:**

Copies the contents of src into dest. Assumes dest has enough memory.

**Example:**

char src[] = "Hello";

char dest[20];

strcpy(dest, src);

**Use Case: Assign or duplicate strings.**

**3. strcat(): Concatenate Two Strings**

**Syntax:**

char \*strcat(char \*dest, const char \*src);

**Description:**

Appends the src string to the end of dest. Assumes dest is large enough.

**Example:**

char greeting[50] = "Hello, ";

char name[] = "John";

strcat(greeting, name); // greeting becomes "Hello, John"

**Use Case: Building full messages, file paths, or commands.**

**4. strcmp(): Compare Two Strings**

**Syntax:**

int strcmp(const char \*str1, const char \*str2);

**Description:**

Returns:

* 0 if strings are equal
* <0 if str1 < str2
* >0 if str1 > str2 (lexicographically)

**Example:**

char a[] = "apple";

char b[] = "banana";

int result = strcmp(a, b); // returns negative value

**Use Case: Sorting, searching, checking for string equality.**

**5. strchr(): Find First Occurrence of a Character**

**Syntax:**

char \*strchr(const char \*str, int ch);

**Description:**

Returns a pointer to the first occurrence of ch in str, or NULL if not found.

**Example:**

char str[] = "programming";

char \*ptr = strchr(str, 'g');

**LAB EXERCISE 11 :**

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

Ans: #include <stdio.h>

#include <string.h>

int main() {

char str1[100], str2[100];

// Input two strings

printf("Enter the first string: ");

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

// Remove newline character if present

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

printf("Enter the second string: ");

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

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

// Concatenate str2 to str1

strcat(str1, str2);

// Display result

printf("\nConcatenated 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

**12. Structures in C**

**THEORY EXERCISE 12 :**

Q12. Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.

Ans: **What Are Structures in C?**

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

It is useful for **modeling real-world entities** like a student, book, or employee that have multiple properties (e.g., name, ID, marks).

**Why Use Structures?**

* To group **related but different types of data**
* To organize and manage **complex records**
* To pass multiple values easily to functions

**Declaring a Structure**

**Syntax:**

struct StructureName {

data\_type member1;

data\_type member2;

// ...

};

**Example:**

struct Student {

int id;

char name[50];

float marks;

};

**Initializing a Structure**

**Method 1: Separate Declaration and Initialization**

struct Student s1;

s1.id = 101;

strcpy(s1.name, "Alice");

s1.marks = 89.5;

**Method 2: Direct Initialization**

struct Student s2 = {102, "Bob", 91.2};

**Accessing Structure Members**

Use the **dot (.) operator** with the structure variable.

printf("ID: %d\n", s1.id);

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

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

**Full Example: C Program Using Structure**

#include <stdio.h>

#include <string.h>

// Declare structure

struct Student {

int id;

char name[50];

float marks;

};

int main() {

struct Student s1;

// Initialize structure members

s1.id = 101;

strcpy(s1.name, "Alice");

s1.marks = 87.5;

// Access and display members

printf("Student Details:\n");

printf("ID: %d\n", s1.id);

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

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

return 0;

}

**LAB EXERCISE 12 :**

12.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 3 students and print them.

Ans: #include <stdio.h>

#include <string.h>

// Define structure

struct Student {

char name[50];

int roll;

float marks;

};

int main() {

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

// Input student details

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

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

printf("Name: ");

scanf(" %[^\n]", students[i].name); // Read string with spaces

printf("Roll Number: ");

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

printf("Marks: ");

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

printf("\n");

}

// Display student details

printf("---- Student Details ----\n");

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

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

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

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

printf("Marks : %.2f\n", students[i].marks);

printf("\n");

}

return 0;

}

**Output:**

Enter details for Student 1:

Name: Alice

Roll Number: 101

Marks: 92.5

Enter details for Student 2:

Name: Bob

Roll Number: 102

Marks: 85.0

Enter details for Student 3:

Name: Charlie

Roll Number: 103

Marks: 78.5

---- Student Details ----

Student 1:

Name : Alice

Roll Number: 101

Marks : 92.50

Student 2:

Name : Bob

Roll Number: 102

Marks : 85.00

Student 3:

Name : Charlie

Roll Number: 103

Marks : 78.50

**13. File Handling in C**

**THEORY EXERCISE 13 :**

Q13. Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.

Ans: **Importance of File Handling in C**

In C, **file handling** is essential for:

* Storing data **permanently** (unlike variables which are temporary)
* **Reading and writing** large amounts of data efficiently
* Creating or processing files such as logs, configurations, reports, etc.
* Allowing **data sharing** between programs and users

C provides built-in support for file operations using the **Standard I/O Library (<stdio.h>)**.

**Basic File Operations in C**

| **Operation** | **Function Used** | **Description** |
| --- | --- | --- |
| Open a file | fopen() | Open file for reading/writing |
| Close a file | fclose() | Close an open file |
| Read from file | fscanf(), fgets() | Read formatted or line-by-line |
| Write to file | fprintf(), fputs() | Write formatted or line-by-line |

**File Opening Modes**

| **Mode** | **Meaning** |
| --- | --- |
| "r" | Open for reading |
| "w" | Open for writing (overwrite) |
| "a" | Open for appending |
| "r+" | Read & write, file must exist |
| "w+" | Read & write, create/overwrite |
| "a+" | Read & append |

**Example: Writing and Reading a File in C**

#include <stdio.h>

int main() {

FILE \*fp;

// --- Writing to a file ---

fp = fopen("data.txt", "w"); // open for writing

if (fp == NULL) {

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

return 1;

}

fprintf(fp, "Hello, File Handling in C!\n");

fprintf(fp, "This is a new line.\n");

fclose(fp); // Close after writing

// --- Reading from a file ---

char line[100];

fp = fopen("data.txt", "r"); // open for reading

if (fp == NULL) {

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

return 1;

}

printf("File contents:\n");

while (fgets(line, sizeof(line), fp)) {

printf("%s", line); // Print each line

}

fclose(fp); // Close after reading

return 0;

}

**Output**

File contents:

Hello, File Handling in C!

This is a new line.

**LAB EXERCISE 13 :**

13. 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.

Ans: #include <stdio.h>

int main() {

FILE \*fp;

char str[] = "Welcome to File Handling in C!";

char buffer[100];

// Step 1: Create and open file for writing

fp = fopen("example.txt", "w"); // "w" creates or overwrites the file

if (fp == NULL) {

printf("Error opening file for writing.\n");

return 1;

}

// Step 2: Write string into file

fprintf(fp, "%s\n", str);

// Step 3: Close the file

fclose(fp);

printf("Data written to file successfully.\n");

// Step 4: Open the same file for reading

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

if (fp == NULL) {

printf("Error opening file for reading.\n");

return 1;

}

// Step 5: Read and display contents

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

while (fgets(buffer, sizeof(buffer), fp)) {

printf("%s", buffer);

}

// Step 6: Close the file again

fclose(fp);

return 0;

}

**Output**

Data written to file successfully.

Reading from file:

Welcome to File Handling in C!

**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.

Ans: #include <stdio.h>

int main() {

float num1, num2;

char op;

// Input two numbers and an operator

printf("Enter first number: ");

scanf("%f", &num1);

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

scanf(" %c", &op); // Note the space before %c to consume any leftover newline

printf("Enter second number: ");

scanf("%f", &num2);

// Perform operation

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: Division by zero!\n");

break;

case '%':

if ((int)num2 != 0)

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

else

printf("Error: Modulus by zero!\n");

break;

default:

printf("Invalid operator!\n");

}

return 0;

}

**Output :**

Enter first number: 12

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

Enter second number: 4

Result = 48.00

**Challenge:** Extend the program to handle invalid operator inputs.

Ans: #include <stdio.h>

int main() {

float num1, num2;

char op;

// Input two numbers and an operator

printf("Enter first number: ");

scanf("%f", &num1);

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

scanf(" %c", &op); // Space before %c handles any leftover newline

printf("Enter second number: ");

scanf("%f", &num2);

// Perform operation with validation

if (op == '+') {

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

}

else if (op == '-') {

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

}

else if (op == '\*') {

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

}

else if (op == '/') {

if (num2 != 0)

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

else

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

}

else if (op == '%') {

if ((int)num2 != 0)

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

else

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

}

else {

// Handle invalid operator input

printf("Error: Invalid operator '%c'. Please use one of +, -, \*, /, %%.\n", op);

}

return 0;

}

**Sample Outputs**

**Valid input:**

Enter first number: 10

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

Enter second number: 5

Result = 15.00

**Invalid operator:**

Enter first number: 8

Enter an operator (+, -, \*, /, %): $

Enter second number: 2

Error: Invalid operator '$'. Please use one of +, -, \*, /, %.

**Enhancements in This Version**

* Handles invalid characters like @, #, etc.
* Prevents crash or undefined behavior due to unknown input
* Displays a helpful error message

Top of Form

**LAB EXERCISE 2: Check Number Properties**

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

o Whether the number is even or odd.

o Whether the number is positive, negative, or zero.

o Whether the number is a multiple of both 3 and 5.

Ans : #include <stdio.h>

int main() {

int num;

// Input number

printf("Enter an integer: ");

scanf("%d", &num);

// Check even or odd using modulus operator

if (num % 2 == 0) {

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

} else {

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

}

// Check positive, negative, or zero using relational operators

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");

}

// Check if number is divisible by both 3 and 5 using logical AND

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;

}

**Output :**

**Enter an integer: 15**

**The number is odd.**

**The number is positive.**

**The number is a multiple of both 3 and 5.**

**2. 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:

o Marks > 90: Grade A

o Marks > 75 and <= 90: Grade B

o Marks > 50 and <= 75: Grade C

o Marks <= 50: Grade D

Ans: #include <stdio.h>

int main() {

float marks;

// Input student marks

printf("Enter the marks of the student: ");

scanf("%f", &marks);

// Check and display grade

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 >= 0 && marks <= 50) {

printf("Grade: D\n");

} else {

printf("Invalid input! Marks should be between 0 and 100.\n");

}

return 0;

}Bottom of Form

**Output :**

Enter the marks of the student: 82

Grade: B

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

Ans: **If-else**

#include <stdio.h>

int main() {

float marks;

// Input marks

printf("Enter the marks of the student: ");

scanf("%f", &marks);

// Decision making using if-else

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 >= 0 && marks <= 50) {

printf("Grade: D\n");

} else {

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

}

return 0;

}

**Switch statement**

#include <stdio.h>

int main() {

int marks;

// Input marks

printf("Enter the marks of the student (0–100): ");

scanf("%d", &marks);

// Decision making using switch

switch (marks / 10) {

case 10: // fall-through

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;

case 4:

case 3:

case 2:

case 1:

case 0:

printf("Grade: D\n");

break;

default:

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

}

return 0;

}

**Output:**

Enter the marks of the student: 78

Grade: B

**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.

Ans: #include <stdio.h>

int main() {

float num1, num2, num3;

// Input three numbers

printf("Enter three numbers:\n");

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

// Find the largest number

float largest;

if (num1 >= num2 && num1 >= num3)

largest = num1;

else if (num2 >= num1 && num2 >= num3)

largest = num2;

else

largest = num3;

// Find the smallest number

float smallest;

if (num1 <= num2 && num1 <= num3)

smallest = num1;

else if (num2 <= num1 && num2 <= num3)

smallest = num2;

else

smallest = num3;

// Output results

printf("Largest number = %.2f\n", largest);

printf("Smallest number = %.2f\n", smallest);

return 0;

}

**Output:**

Enter three numbers:

8 15 3

Largest number = 15.00

Smallest number = 3.00

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

Ans: **If – else statement**

#include <stdio.h>

int main() {

float a, b, c, largest, smallest;

printf("Enter three numbers: ");

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

// Find the largest

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

largest = a;

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

largest = b;

else

largest = c;

// Find the smallest

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

smallest = a;

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

smallest = b;

else

smallest = c;

printf("Using if-else:\n");

printf("Largest = %.2f\n", largest);

printf("Smallest = %.2f\n", smallest);

return 0;

}

**Switch statement**

#include <stdio.h>

int main() {

float a, b, c;

int largest\_case = 0, smallest\_case = 0;

printf("Enter three numbers: ");

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

// Assign a code for largest

if (a >= b && a >= c) largest\_case = 1;

else if (b >= a && b >= c) largest\_case = 2;

else largest\_case = 3;

// Assign a code for smallest

if (a <= b && a <= c) smallest\_case = 1;

else if (b <= a && b <= c) smallest\_case = 2;

else smallest\_case = 3;

printf("Using switch-case:\n");

// Display largest

switch (largest\_case) {

case 1: printf("Largest = %.2f\n", a); break;

case 2: printf("Largest = %.2f\n", b); break;

case 3: printf("Largest = %.2f\n", c); break;

default: printf("Error in largest case.\n");

}

// Display smallest

switch (smallest\_case) {

case 1: printf("Smallest = %.2f\n", a); break;

case 2: printf("Smallest = %.2f\n", b); break;

case 3: printf("Smallest = %.2f\n", c); break;

default: printf("Error in smallest case.\n");

}

return 0;

}

**Output :**

Enter three numbers: 10 25 3

Using if-else:

Largest = 25.00

Smallest = 3.00

Using switch-case:

Largest = 25.00

Smallest = 3.00

1. **Loops**

**LAB EXERCISE 1: Prime Number Check**

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

Ans:

#include <stdio.h>

int main() {

int num, i, isPrime = 1;

// Input from user

printf("Enter a number: ");

scanf("%d", &num);

// Check for numbers less than 2

if (num <= 1) {

isPrime = 0;

} else {

// Check divisibility from 2 to num-1

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;

}

**Output:**

Enter a number: 7

7 is a prime number.

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

**Ans:**

#include <stdio.h>

int main() {

int limit, num, i, isPrime;

// Input from user

printf("Enter the upper limit: ");

scanf("%d", &limit);

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

// Loop through all numbers from 2 to limit

for (num = 2; num <= limit; num++) {

isPrime = 1; // Assume number is prime

// Check divisibility

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

if (num % i == 0) {

isPrime = 0; // Not prime

break;

}

}

// If number is prime, print it

if (isPrime) {

printf("%d ", num);

}

}

printf("\n");

return 0;

}

**Output :**

Enter the upper limit: 20

Prime numbers between 1 and 20 are:

2 3 5 7 11 13 17 19

**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.

**Ans:**

#include <stdio.h>

int main() {

int num, i;

// Input from user

printf("Enter a number to print its multiplication table: ");

scanf("%d", &num);

printf("Multiplication table for %d:\n", num);

// Print multiplication table from 1 to 10

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

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

}

return 0;

}

**Output :**

Enter a number to print its multiplication table: 5

Multiplication table for 5:

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

5 x 10 = 50

**Challenge:** Allow the user to input the range of the multiplication table (e.g., from 1 to N).

**Ans:**

#include <stdio.h>

int main() {

int num, range, i;

// Input from user

printf("Enter a number to print its multiplication table: ");

scanf("%d", &num);

printf("Enter the range (e.g., up to what number to multiply): ");

scanf("%d", &range);

printf("Multiplication table for %d up to %d:\n", num, range);

// Print multiplication table from 1 to range

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

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

}

return 0;

}

**Output :**

Enter a number to print its multiplication table: 7

Enter the range (e.g., up to what number to multiply): 12

Multiplication table for 7 up to 12:

7 x 1 = 7

7 x 2 = 14

7 x 3 = 21

...

7 x 12 = 84

**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.

**Ans:** #include <stdio.h>

int main() {

int number, digit, sum = 0;

// Take input from the user

printf("Enter an integer: ");

scanf("%d", &number);

// Make number positive if it's negative

if (number < 0) {

number = -number;

}

// Calculate sum of digits using while loop

while (number != 0) {

digit = number % 10; // Extract last digit

sum += digit; // Add to sum

number /= 10; // Remove last digit

}

// Display result

printf("Sum of the digits: %d\n", sum);

return 0;

}

**Output:**

Enter an integer: 1234

Sum of the digits: 10

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

**Ans:**  #include <stdio.h>

int main() {

int number, digit;

int sum = 0, reverse = 0;

// Take input from the user

printf("Enter an integer: ");

scanf("%d", &number);

// Store original number for display

int originalNumber = number;

// Make number positive if it's negative (for digit operations)

int temp = (number < 0) ? -number : number;

// Process digits using while loop

while (temp != 0) {

digit = temp % 10; // Extract last digit

sum += digit; // Add to sum

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

temp /= 10; // Remove last digit

}

// Print the results

printf("Original number: %d\n", originalNumber);

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

// Maintain the sign of reversed number if input was negative

if (originalNumber < 0)

reverse = -reverse;

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

return 0;

}

**Output:**

Enter an integer: -1234

Original number: -1234

Sum of digits: 10

Reversed number: -4321

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.

**Ans:** #include <stdio.h>

int main() {

int numbers[10];

int i, max, min;

// Input 10 integers

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

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

printf("Number %d: ", i + 1);

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

}

// Initialize max and min to the first element

max = min = numbers[0];

// Find max and min

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

if (numbers[i] > max) {

max = numbers[i];

}

if (numbers[i] < min) {

min = numbers[i];

}

}

// Output results

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

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

return 0;

}

**Output:**

Enter 10 integers:

Number 1: 34

Number 2: 12

Number 3: 89

Number 4: 4

Number 5: 56

Number 6: 78

Number 7: 23

Number 8: 90

Number 9: 67

Number 10: 15

Maximum value: 90

Minimum value: 4

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

**Ans:** #include <stdio.h>

int main() {

int numbers[10];

int i, j, temp;

int max, min;

// Input 10 integers

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

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

printf("Number %d: ", i + 1);

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

}

// Initialize max and min

max = min = numbers[0];

// Find max and min

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

if (numbers[i] > max) {

max = numbers[i];

}

if (numbers[i] < min) {

min = numbers[i];

}

}

// Sort array in ascending order using Bubble Sort

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

for (j = 0; j < 9 - i; j++) {

if (numbers[j] > numbers[j + 1]) {

// Swap

temp = numbers[j];

numbers[j] = numbers[j + 1];

numbers[j + 1] = temp;

}

}

}

// Print results

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

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

printf("Sorted array in ascending order:\n");

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

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

}

printf("\n");

return 0;

}

**Output:**

Enter 10 integers:

Number 1: 34

Number 2: 12

Number 3: 89

Number 4: 4

Number 5: 56

Number 6: 78

Number 7: 23

Number 8: 90

Number 9: 67

Number 10: 15

Maximum value: 90

Minimum value: 4

Sorted array in ascending order:

4 12 15 23 34 56 67 78 89 90

**LAB EXERCISE 2: Matrix Addition**

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

**Ans:** #include <stdio.h>

int main() {

int matrix1[2][2], matrix2[2][2], sum[2][2];

int i, j;

// Input first matrix

printf("Enter elements of the first 2x2 matrix:\n");

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

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

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

scanf("%d", &matrix1[i][j]);

}

}

// Input second matrix

printf("Enter elements of the second 2x2 matrix:\n");

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

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

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

scanf("%d", &matrix2[i][j]);

}

}

// Calculate the sum of matrices

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

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

sum[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

// Display the resultant matrix

printf("\nResultant matrix after addition:\n");

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

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

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

}

printf("\n");

}

return 0;

}

**Output:**

Enter elements of the first 2x2 matrix:

Element [0][0]: 1

Element [0][1]: 2

Element [1][0]: 3

Element [1][1]: 4

Enter elements of the second 2x2 matrix:

Element [0][0]: 5

Element [0][1]: 6

Element [1][0]: 7

Element [1][1]: 8

Resultant matrix after addition:

6 8

10 12

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

**Ans:** #include <stdio.h>

int main() {

int matrix1[3][3], matrix2[3][3], sum[3][3], product[3][3];

int i, j, k;

// Input for first 3x3 matrix

printf("Enter elements of the first 3x3 matrix:\n");

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

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

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

scanf("%d", &matrix1[i][j]);

}

}

// Input for second 3x3 matrix

printf("Enter elements of the second 3x3 matrix:\n");

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

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

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

scanf("%d", &matrix2[i][j]);

}

}

// Calculate sum of the matrices

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

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

sum[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

// Calculate product of the matrices

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

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

product[i][j] = 0; // Initialize product cell

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

product[i][j] += matrix1[i][k] \* matrix2[k][j];

}

}

}

// Display sum matrix

printf("\nSum of the two matrices:\n");

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

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

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

}

printf("\n");

}

// Display product matrix

printf("\nProduct of the two matrices:\n");

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

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

printf("%d\t", product[i][j]);

}

printf("\n");

}

return 0;

}

**Output:**

Enter elements of the first 3x3 matrix:

Element [0][0]: 1

Element [0][1]: 2

Element [0][2]: 3

Element [1][0]: 4

Element [1][1]: 5

Element [1][2]: 6

Element [2][0]: 7

Element [2][1]: 8

Element [2][2]: 9

Enter elements of the second 3x3 matrix:

Element [0][0]: 9

Element [0][1]: 8

Element [0][2]: 7

Element [1][0]: 6

Element [1][1]: 5

Element [1][2]: 4

Element [2][0]: 3

Element [2][1]: 2

Element [2][2]: 1

Sum of the two matrices:

10 10 10

10 10 10

10 10 10

Product of the two matrices:

30 24 18

84 69 54

138 114 90

**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.

**Ans:** #include <stdio.h>

int main() {

int N, i, sum = 0;

// Ask user for the number of elements

printf("Enter the number of elements (N): ");

scanf("%d", &N);

// Declare an array of size N

int numbers[N];

// Input elements into the array

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

for (i = 0; i < N; i++) {

printf("Element %d: ", i + 1);

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

sum += numbers[i]; // Add to sum directly while inputting

}

// Display the sum

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

return 0;

}

**Output:**

Enter the number of elements (N): 5

Enter 5 integers:

Element 1: 10

Element 2: 20

Element 3: 30

Element 4: 40

Element 5: 50

Sum of all array elements = 150

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

**Ans**: #include <stdio.h>

int main() {

int N, i, sum = 0;

float average;

// Ask user for the number of elements

printf("Enter the number of elements (N): ");

scanf("%d", &N);

// Declare an array of size N

int numbers[N];

// Input elements and calculate sum

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

for (i = 0; i < N; i++) {

printf("Element %d: ", i + 1);

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

sum += numbers[i];

}

// Calculate average

average = (float)sum / N;

// Display results

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

printf("Average of the numbers = %.2f\n", average);

return 0;

}

**Output:**

Enter the number of elements (N): 4

Enter 4 integers:

Element 1: 10

Element 2: 20

Element 3: 30

Element 4: 40

Sum of all array elements = 100

Average of the numbers = 25.00

1. **Functions**

**LAB EXERCISE 1: Fibonacci Sequence**

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

**Ans: #**include <stdio.h>

// Recursive function to return nth Fibonacci number

int fibonacci(int n) {

if (n == 0)

return 0;

else if (n == 1)

return 1;

else

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

}

int main() {

int N, i;

// Input number of terms

printf("Enter the number of terms (N): ");

scanf("%d", &N);

// Print Fibonacci sequence

printf("Fibonacci sequence up to %d terms:\n", N);

for (i = 0; i < N; i++) {

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

}

printf("\n");

return 0;

}

**Output:**

Enter the number of terms (N): 7

Fibonacci sequence up to 7 terms:

0 1 1 2 3 5 8

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

**Ans:** #include <stdio.h>

#include <time.h>

// Recursive function

int fibonacci\_recursive(int n) {

if (n == 0)

return 0;

else if (n == 1)

return 1;

else

return fibonacci\_recursive(n - 1) + fibonacci\_recursive(n - 2);

}

// Iterative function

int fibonacci\_iterative(int n) {

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

if (n == 0) return 0;

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

next = a + b;

a = b;

b = next;

}

return b;

}

int main() {

int N;

clock\_t start, end;

double time\_recursive, time\_iterative;

// Input

printf("Enter the value of N: ");

scanf("%d", &N);

// Recursive method

start = clock();

int fib\_rec = fibonacci\_recursive(N);

end = clock();

time\_recursive = (double)(end - start) / CLOCKS\_PER\_SEC;

// Iterative method

start = clock();

int fib\_iter = fibonacci\_iterative(N);

end = clock();

time\_iterative = (double)(end - start) / CLOCKS\_PER\_SEC;

// Output results

printf("\nNth Fibonacci number (Recursive): %d\n", fib\_rec);

printf("Time taken (Recursive): %.6f seconds\n", time\_recursive);

printf("\nNth Fibonacci number (Iterative): %d\n", fib\_iter);

printf("Time taken (Iterative): %.6f seconds\n", time\_iterative);

return 0;

}

**Output:**

Enter the value of N: 30

Nth Fibonacci number (Recursive): 832040

Time taken (Recursive): 0.455000 seconds

Nth Fibonacci number (Iterative): 832040

Time taken (Iterative): 0.000002 seconds

**LAB EXERCISE 2: Factorial Calculation**

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

**Ans:** #include <stdio.h>

long long factorial(int n) {

long long fact = 1;

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

fact \*= i;

}

return fact;

}

int main() {

int num;

// Input from user

printf("Enter a non-negative integer: ");

scanf("%d", &num);

// Check for valid input

if (num < 0) {

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

} else {

// Call factorial function

long long result = factorial(num);

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

}

return 0;

}

**Output:**

Enter a non-negative integer: 5

Factorial of 5 = 120

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

**Ans:** #include <stdio.h>

#include <time.h>

// Recursive factorial function

long long factorial\_recursive(int n) {

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

return 1;

else

return n \* factorial\_recursive(n - 1);

}

// Iterative factorial function

long long factorial\_iterative(int n) {

long long result = 1;

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

result \*= i;

}

return result;

}

int main() {

int n;

clock\_t start, end;

double time\_recursive, time\_iterative;

// Input from user

printf("Enter a non-negative integer: ");

scanf("%d", &n);

if (n < 0) {

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

return 1;

}

// Recursive calculation and timing

start = clock();

long long fact\_rec = factorial\_recursive(n);

end = clock();

time\_recursive = (double)(end - start) / CLOCKS\_PER\_SEC;

// Iterative calculation and timing

start = clock();

long long fact\_iter = factorial\_iterative(n);

end = clock();

time\_iterative = (double)(end - start) / CLOCKS\_PER\_SEC;

// Output results

printf("\nFactorial of %d (Recursive): %lld\n", n, fact\_rec);

printf("Time taken (Recursive): %.6f seconds\n", time\_recursive);

printf("\nFactorial of %d (Iterative): %lld\n", n, fact\_iter);

printf("Time taken (Iterative): %.6f seconds\n", time\_iterative);

return 0;

}

**Output:**

Enter a non-negative integer: 20

Factorial of 20 (Recursive): 2432902008176640000

Time taken (Recursive): 0.000002 seconds

Factorial of 20 (Iterative): 2432902008176640000

Time taken (Iterative): 0.000001 seconds

**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.

**Ans:** #include <stdio.h>

// Function to check if a number is palindrome

int isPalindrome(int num) {

int original = num, reversed = 0, digit;

while (num > 0) {

digit = num % 10;

reversed = reversed \* 10 + digit;

num /= 10;

}

return (original == reversed);

}

int main() {

int number;

// Input from user

printf("Enter a number: ");

scanf("%d", &number);

// Negative numbers are not considered palindrome

if (number < 0) {

printf("Negative numbers are not considered palindrome.\n");

} else if (isPalindrome(number)) {

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

} else {

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

}

return 0;

}

**Output:**

Enter a number: 121

121 is a palindrome.

Enter a number: 123

123 is not a palindrome.

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

**Ans:** #include <stdio.h>

#include <string.h>

#include <ctype.h>

// Function to check if a string is a palindrome

int isPalindrome(char str[]) {

int left = 0;

int right = strlen(str) - 1;

while (left < right) {

// Convert to lowercase for case-insensitive comparison

if (tolower(str[left]) != tolower(str[right])) {

return 0; // Not a palindrome

}

left++;

right--;

}

return 1; // Is a palindrome

}

int main() {

char str[100];

// Input string from user

printf("Enter a string: ");

scanf("%s", str); // Note: This reads a single word. Use fgets() if spaces are needed.

// Check and display result

if (isPalindrome(str)) {

printf("\"%s\" is a palindrome.\n", str);

} else {

printf("\"%s\" is not a palindrome.\n", str);

}

return 0;

}

Output:

Enter a string: RaceCar

"RaceCar" is a palindrome.

Enter a string: Hello

"Hello" is not a palindrome.

1. **Strings**

**LAB EXERCISE 1: String Reversal**

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

**Ans:** #include <stdio.h>

#include <string.h>

// Function to reverse a string

void reverseString(char str[]) {

int start = 0;

int end = strlen(str) - 1;

char temp;

while (start < end) {

// Swap characters

temp = str[start];

str[start] = str[end];

str[end] = temp;

start++;

end--;

}

}

int main() {

char str[100];

// Input string

printf("Enter a string: ");

scanf("%s", str); // Reads a single word. Use fgets for full lines.

// Reverse the string

reverseString(str);

// Output the result

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

return 0;

}

**Output:**

Enter a string: Hello

Reversed string: olleH

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

**Ans:** #include <stdio.h>

// Function to calculate length of string manually

int stringLength(char str[]) {

int length = 0;

while (str[length] != '\0') {

length++;

}

return length;

}

// Function to reverse the string

void reverseString(char str[]) {

int start = 0;

int end = stringLength(str) - 1;

char temp;

while (start < end) {

// Swap characters

temp = str[start];

str[start] = str[end];

str[end] = temp;

start++;

end--;

}

}

int main() {

char str[100];

// Input string from user

printf("Enter a string: ");

scanf("%s", str); // Use fgets if you want to include spaces

// Reverse the string

reverseString(str);

// Display result

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

return 0;

}

**Output:**

Enter a string: Code

Reversed string: edoC

**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.

**Ans:** #include <stdio.h>

// Function to check if a character is a vowel

int isVowel(char ch) {

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

return (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u');

}

int main() {

char str[100];

int i = 0, vowels = 0, consonants = 0;

// Input string

printf("Enter a string: ");

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

// Count vowels and consonants

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

char ch = str[i];

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

if (isVowel(ch))

vowels++;

else

consonants++;

}

i++;

}

// Output results

printf("Number of vowels: %d\n", vowels);

printf("Number of consonants: %d\n", consonants);

return 0;

}

**Output:**

Enter a string: Hello World

Number of vowels: 3

Number of consonants: 7

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

**Ans:** #include <stdio.h>

// Function to check if a character is a vowel

int isVowel(char ch) {

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

return (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u');

}

int main() {

char str[100];

int i = 0, vowels = 0, consonants = 0, digits = 0, specialChars = 0;

// Input string

printf("Enter a string: ");

scanf(" %[^\n]", str); // Reads full line including spaces

// Analyze characters

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

char ch = str[i];

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

if (isVowel(ch))

vowels++;

else

consonants++;

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

digits++;

} else {

specialChars++;

}

i++;

}

// Output results

printf("\nNumber of vowels: %d\n", vowels);

printf("Number of consonants: %d\n", consonants);

printf("Number of digits: %d\n", digits);

printf("Number of special characters: %d\n", specialChars);

return 0;

}

**Output:**

Enter a string: Hello World! 123

Number of vowels: 3

Number of consonants: 7

Number of digits: 3

Number of special characters: 4

**LAB EXERCISE 3: Word Count**

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

**Ans:** #include <stdio.h>

int countWords(char str[]) {

int i = 0, words = 0, inWord = 0;

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

if (str[i] != ' ' && str[i] != '\t' && str[i] != '\n') {

if (!inWord) {

words++;

inWord = 1; // We're inside a word

}

} else {

inWord = 0; // We've exited a word

}

i++;

}

return words;

}

int main() {

char sentence[200];

// Input sentence

printf("Enter a sentence: ");

fgets(sentence, sizeof(sentence), stdin); // Allows spaces

// Remove trailing newline if present

int len = 0;

while (sentence[len] != '\0') {

if (sentence[len] == '\n') {

sentence[len] = '\0';

break;

}

len++;

}

// Count and display word count

int wordCount = countWords(sentence);

printf("Number of words: %d\n", wordCount);

return 0;

}

**Output:**

Enter a sentence: Hello, how are you doing today?

Number of words: 6

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

**Ans:** #include <stdio.h>

#include <string.h>

#include <ctype.h>

int main() {

char sentence[200];

char longestWord[100];

char currentWord[100];

int i = 0, j = 0, maxLength = 0, length = 0;

// Input the sentence

printf("Enter a sentence: ");

fgets(sentence, sizeof(sentence), stdin);

// Remove trailing newline if present

int len = 0;

while (sentence[len] != '\0') {

if (sentence[len] == '\n') {

sentence[len] = '\0';

break;

}

len++;

}

int wordCount = 0;

while (1) {

char ch = sentence[i];

if (ch != ' ' && ch != '\t' && ch != '\0') {

currentWord[j++] = ch;

} else {

if (j > 0) {

currentWord[j] = '\0'; // Terminate current word

wordCount++;

length = j;

if (length > maxLength) {

maxLength = length;

strcpy(longestWord, currentWord); // Store the longest word

}

j = 0; // Reset for next word

}

}

if (ch == '\0')

break;

i++;

}

// Output results

printf("\nTotal number of words: %d\n", wordCount);

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

printf("Length of longest word: %d\n", maxLength);

return 0;

}

**Output:**

Enter a sentence: Programming in C is fun and challenging

Total number of words: 6

Longest word: challenging

Length of longest word: 11

**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).

**Ans:** #include <stdio.h>

#include <math.h>

int main() {

int num, original, remainder, digits = 0;

int sum = 0;

// Input from user

printf("Enter a number: ");

scanf("%d", &num);

original = num;

// Count number of digits

int temp = num;

while (temp != 0) {

digits++;

temp /= 10;

}

// Calculate the sum of each digit raised to the power of 'digits'

temp = num;

while (temp != 0) {

remainder = temp % 10;

sum += pow(remainder, digits);

temp /= 10;

}

// Check for Armstrong number

if (sum == original)

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

else

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

return 0;

}

**Output:**

Enter a number: 153

153 is an Armstrong number.

Enter a number: 123

123 is not an Armstrong number.

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

**Ans:** #include <stdio.h>

#include <math.h>

int main() {

int num, original, temp, digits, remainder, sum;

printf("Armstrong numbers between 1 and 1000 are:\n");

for (num = 1; num <= 1000; num++) {

original = num;

temp = num;

digits = 0;

sum = 0;

// Count number of digits

while (temp != 0) {

digits++;

temp /= 10;

}

// Reset temp to original value

temp = num;

// Calculate sum of digits raised to the power of 'digits'

while (temp != 0) {

remainder = temp % 10;

sum += pow(remainder, digits);

temp /= 10;

}

// Check if it's an Armstrong number

if (sum == original) {

printf("%d\n", original);

}

}

return 0;

}

**Output:**

Armstrong numbers between 1 and 1000 are:

1

2

3

4

5

6

7

8

9

153

370

371

407

**Lab Challenge 2: Pascal’s Triangle**

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

**Ans:** #include <stdio.h>

// Function to calculate binomial coefficient (nCr)

int binomialCoefficient(int n, int r) {

int result = 1;

if (r > n - r) // Take advantage of symmetry

r = n - r;

for (int i = 0; i < r; i++) {

result \*= (n - i);

result /= (i + 1);

}

return result;

}

int main() {

int rows;

// Input number of rows

printf("Enter number of rows for Pascal's Triangle: ");

scanf("%d", &rows);

// Generate Pascal's Triangle

for (int i = 0; i < rows; i++) {

// Print spaces for formatting

for (int space = 0; space < rows - i - 1; space++) {

printf(" ");

}

// Print numbers in the row

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

printf("%4d", binomialCoefficient(i, j));

}

printf("\n");

}

return 0;

}

**Output:**

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

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

**Ans:** #include <stdio.h>

// Recursive function to calculate nCr (binomial coefficient)

int binomialCoefficient(int n, int r) {

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

return 1;

else

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

}

int main() {

int rows;

// Input number of rows

printf("Enter number of rows for Pascal's Triangle: ");

scanf("%d", &rows);

// Generate Pascal's Triangle

for (int i = 0; i < rows; i++) {

// Print leading spaces for formatting

for (int space = 0; space < rows - i - 1; space++) {

printf(" ");

}

// Print binomial coefficients for the row

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

printf("%4d", binomialCoefficient(i, j));

}

printf("\n");

}

return 0;

}

**Output:**

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

**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.

**Ans:** #include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main() {

int number, guess, attempts = 0, maxAttempts = 7;

// Seed the random number generator

srand(time(0));

number = rand() % 100 + 1; // Random number between 1 and 100

printf("Welcome to the Number Guessing Game!\n");

printf("I'm thinking of a number between 1 and 100.\n");

printf("You have %d attempts to guess it.\n\n", maxAttempts);

// Game loop

while (attempts < maxAttempts) {

printf("Attempt %d: Enter your guess: ", attempts + 1);

scanf("%d", &guess);

attempts++;

if (guess == number) {

printf(" Congratulations! You guessed the number in %d attempts.\n", attempts);

break;

} else if (guess < number) {

printf("Too low! Try again.\n");

} else {

printf("Too high! Try again.\n");

}

}

if (guess != number) {

printf("\n Sorry! You've used all your attempts.\n");

printf("The correct number was: %d\n", number);

}

return 0;

}

**Output:**

Welcome to the Number Guessing Game!

I'm thinking of a number between 1 and 100.

You have 7 attempts to guess it.

Attempt 1: Enter your guess: 50

Too low! Try again.

Attempt 2: Enter your guess: 75

Too high! Try again.

Attempt 3: Enter your guess: 62

Too high! Try again.

Attempt 4: Enter your guess: 56

Too low! Try again.

Attempt 5: Enter your guess: 59

Congratulations! You guessed the number in 5 attempts.

**Challenge:** Provide hints to the user if the guessed number is too high or too low.

**Ans:** #include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

int main() {

int number, guess, attempts = 0, maxAttempts = 7, difference;

// Seed the random number generator

srand(time(0));

number = rand() % 100 + 1; // Random number between 1 and 100

printf(" Welcome to the Number Guessing Game!\n");

printf("I'm thinking of a number between 1 and 100.\n");

printf("You have %d attempts to guess it.\n\n", maxAttempts);

// Game loop

while (attempts < maxAttempts) {

printf("Attempt %d: Enter your guess: ", attempts + 1);

scanf("%d", &guess);

attempts++;

difference = abs(guess - number);

if (guess == number) {

printf(" Congratulations! You guessed the number in %d attempts.\n", attempts);

break;

} else {

if (guess < number) {

if (difference >= 20)

printf("Way too low!\n");

else if (difference >= 10)

printf("Too low!\n");

else

printf("A little low, you're close!\n");

} else {

if (difference >= 20)

printf("Way too high!\n");

else if (difference >= 10)

printf("Too high!\n");

else

printf("A little high, you're close!\n");

}

}

}

if (guess != number) {

printf("\n Sorry! You've used all your attempts.\n");

printf("The correct number was: %d\n", number);

}

return 0;

}

**Output:**

Enter your guess: 25

Way too low!

Enter your guess: 40

A little low, you're close!

Enter your guess: 45

Congratulations! You guessed the number in 3 attempts.