INSTITUTE :- TOPS TECHNOLOGIES (BARODA)

NAME :- Arjun Prajapati

COURSE :- PYTHON FULL STACK DEVELOPER

Module 2 – Introduction to Programming Overview of C Programming

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

**The History and Importance of C Programming**

C programming is one of the most important and widely used programming languages in the world. It was created in the early 1970s by Dennis Ritchie at Bell Labs. The main purpose of C was to develop the UNIX operating system, which was also created at Bell Labs. Before C, programmers used assembly language, which was difficult to write and understand. C made programming easier while still being powerful and fast.

**Evolution of C**

After its creation, C became very popular because it was simple, flexible, and efficient. In 1978, Brian Kernighan and Dennis Ritchie wrote a book called *The C Programming Language*, which helped programmers learn and use C easily. This book became a standard guide for C programming.

In 1989, the American National Standards Institute (ANSI) created a standard version of C, called ANSI C. This standard made sure that C programs could run on different computers without changes. Later, the International Organization for Standardization (ISO) also approved it, and updates like C99 and C11 were introduced to improve the language.

**Why is C Still Used Today?**

Even though many new programming languages like Python, Java, and C++ have been developed, C is still widely used. Here are some reasons why:

1. **Efficiency** – C is very fast because it works closely with the computer’s hardware.
2. **Portability** – Programs written in C can run on different types of computers with little or no change.
3. **Foundation for Other Languages** – Many modern languages, like C++, Java, and Python, are based on C.
4. **Used in System Programming** – Operating systems like Windows, Linux, and macOS are written in C.
5. **Embedded Systems** – C is used in small devices like microwaves, cars, and medical equipment because it is lightweight and efficient.

• LAB EXERCISE: o 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** – C is widely used in microcontrollers, automotive systems, medical devices, and IoT devices due to its efficiency and hardware-level control.

2 **Operating Systems** – Major OSs like Windows, Linux, and macOS have core components written in C because of its speed and direct memory management.

3 **Game Development** – C (along with C++) is used in game engines like Unity and Unreal Engine for performance-intensive tasks like physics calculations and graphics rendering.

2. Setting Up Environment

• THEORY EXERCISE: o Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like DevC++, VS Code, or CodeBlocks.

**Step 1: Install GCC Compiler (MinGW-w64)**

1. **Download** MinGW-w64 from [MinGW-w64 official site](https://www.mingw-w64.org/" \t "_new).
2. **Install** it by following the setup instructions (choose default settings).
3. **Add MinGW to System Path**:
   * Search **"Environment Variables"** in Windows.
   * Edit the **PATH** variable and add the MinGW bin folder (e.g., C:\MinGW\bin).
4. **Verify installation**: Open **Command Prompt** and type:

gcc --version

If it shows a version number, GCC is installed successfully!

**Step 2: Install and Set Up an IDE**

**Option 1: Dev-C++**

1. Download **Dev-C++** from SourceForge.
2. Install and open it.
3. Create a **New Project**, select **C**, and start coding!

**Option 2: VS Code (Recommended)**

1. Download and install **VS Code** from [here](https://code.visualstudio.com/" \t "_new).
2. Open **VS Code** and go to **Extensions** (Ctrl+Shift+X).
3. Search for **"C/C++"** and install the extension.
4. Create a new **C file (.c)**, write your code, and press **Ctrl+Shift+B** to compile and run.

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

#include<stdio.h>

Int main()

{

Printf(“hello World”);

Return 0;

}

3. Basic Structure of a C Program

• THEORY EXERCISE: o Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.

**Headers** – Include standard libraries using #include.

**Main Function** – main() is the entry point of the program.

**Comments** – Used for explanations (// for single-line, /\* \*/ for multi-line).

**Data Types & Variables** – Define variables with appropriate types (int, float, char, etc.).

**Statements & Logic** – The program logic is written inside {}.

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

#include <stdio.h>

#define PI 3.14159

int main() {

int age = 25;

float height = 5.9;

char grade = 'A';

const int YEAR = 2024;

// Displaying values

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

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

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

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

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

return 0;

}

4. Operators in C

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

**Operators in C (Simple Explanation)**

**1. Arithmetic Operators (Perform mathematical operations)**

| **Operator** | **Description** | **Example (a = 10, b = 5)** | **Result** |
| --- | --- | --- | --- |
| + | Addition | a + b | 15 |
| - | Subtraction | a - b | 5 |
| \* | Multiplication | a \* b | 50 |
| / | Division | a / b | 2 |
| % | Modulus (Remainder) | a % b | 0 |

**2. Relational Operators (Compare values, return true (1) or false (0))**

| **Operator** | **Description** | **Example (a = 10, b = 5)** | **Result** |
| --- | --- | --- | --- |
| == | Equal to | a == b | 0 (false) |
| != | Not equal to | a != b | 1 (true) |
| > | Greater than | a > b | 1 (true) |
| < | Less than | a < b | 0 (false) |
| >= | Greater than or equal to | a >= b | 1 (true) |
| <= | Less than or equal to | a <= b | 0 (false) |

**3. Logical Operators (Used in conditions, return true (1) or false (0))**

| **Operator** | **Description** | **Example (a = 10, b = 5)** | **Result** |
| --- | --- | --- | --- |
| && | Logical AND | (a > 5) && (b > 2) | 1 (true) |
| ` |  | ` | Logical OR |
| ! | Logical NOT | !(a > b) | 0 (false) |

**4. Assignment Operators (Assign values to variables)**

| **Operator** | **Description** | **Example (a = 10)** | **Equivalent** |
| --- | --- | --- | --- |
| = | Assign value | a = 5 | a = 5 |
| += | Add and assign | a += 5 | a = a + 5 |
| -= | Subtract and assign | a -= 3 | a = a - 3 |
| \*= | Multiply and assign | a \*= 2 | a = a \* 2 |
| /= | Divide and assign | a /= 2 | a = a / 2 |
| %= | Modulus and assign | a %= 3 | a = a % 3 |

**5. Increment & Decrement Operators (Increase or decrease value by 1)**

| **Operator** | **Description** | **Example (a = 10)** | **Result** |
| --- | --- | --- | --- |
| ++a | Pre-increment | ++a | a = 11 before using it |
| a++ | Post-increment | a++ | Uses a = 10, then increases to 11 |
| --a | Pre-decrement | --a | a = 9 before using it |
| a-- | Post-decrement | a-- | Uses a = 10, then decreases to 9 |

**6. Bitwise Operators (Operate at the binary level)**

| **Operator** | **Description** | **Example (a = 5 (0101), b = 3 (0011))** | **Result** |
| --- | --- | --- | --- |
| & | AND | a & b | 1 (0001) |
| ` | ` | OR | `a |
| ^ | XOR | a ^ b | 6 (0110) |
| ~ | NOT | ~a | -6 (inverts bits) |
| << | Left Shift | a << 1 | 10 (shifts left) |
| >> | Right Shift | a >> 1 | 2 (shifts right) |

**7. Conditional (Ternary) Operator (Shortens if-else conditions)**

Format:

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

Example:

int a = 10, b = 5;

int min = (a < b) ? a : b;

• LAB EXERCISE: o Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results. 5. Control Flow Statements in C

#include <stdio.h>

int main() {

int a, b;

printf("Enter two integers: ");

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

// Arithmetic operations

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

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

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

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

if (b != 0) { // Prevent division by zero

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

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

} else {

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

}

// Relational operations

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

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

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

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

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

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

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

// Logical operations

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

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

return 0;

}

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

**1. if Statement**

#include <stdio.h>

int main() {

int num = 10;

if (num > 0) {

printf("Positive number\n");

}

return 0;

}

**2. if-else Statement**

#include <stdio.h>

int main() {

int num = -5;

if (num > 0) {

printf("Positive number\n");

} else {

printf("Negative number\n");

}

return 0;

}

**3. Nested if-else**

if-else inside another if-else.

#include <stdio.h>

int main() {

int num = 0;

if (num > 0) {

printf("Positive\n");

} else {

if (num == 0) {

printf("Zero\n");

} else {

printf("Negative\n");

}

}

return 0;

}

**4. switch Statement**

#include <stdio.h>

int main() {

int day = 2;

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");

}

return 0;

}

• 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>

int main() {

int num, month;

printf("Enter a number: ");

scanf("%d", &num);

if (num % 2 == 0) {

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

} else {

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

}

printf("Enter a number (1-12) for the month: ");

scanf("%d", &month);

switch (month) {

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

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

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

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

case 5: printf("May\n"); break;

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

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

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

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

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

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

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

default: printf("Invalid month number\n");

}

return 0;

}

6. Looping in C

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

**Comparison of Loops in C**

**1. while Loop**

* **Condition** is checked **before** running the loop.
* Runs **zero or more times** depending on the condition.
* **Best for**: Unknown number of iterations, loop runs as long as the condition is true.

**2. for Loop**

* **Condition** is checked **before** running the loop.
* Runs a **set number of times** (defined by initialization, condition, and increment/decrement).
* **Best for**: Known number of iterations, like counting or looping through an array.

**3. do-while Loop**

* **Condition** is checked **after** the loop runs.
* Always runs **at least once**, even if the condition is false.
* **Best for**: When you want the loop to run at least once before checking the condition.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Loop Type** | **Condition Check** | **Executes At Least Once?** | **Best For** | | --- | --- | --- | --- | | **while** | Before loop | No | Unknown iterations based on condition. | | **for** | Before loop | No | Known number of iterations. | | **do-while** | After loop | Yes | At least one iteration before condition. | |

**Scenarios for Each Loop in C**

**1. while Loop**

* **Scenario**: When you don’t know how many times the loop will run, but you want the loop to continue as long as a condition is true.
* **Example**: Reading user input until they enter a valid value, or waiting for a sensor to trigger an event.
* **Use case**:

while (user\_input != 0) {

// process input

scanf("%d", &user\_input);

}

**2. for Loop**

* **Scenario**: When you know exactly how many times the loop should run, such as when iterating over a fixed range or array.
* **Example**: Looping through the elements of an array, or counting from 1 to 10.
* **Use case**:

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

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

}

**3. do-while Loop**

* **Scenario**: When you need the loop to execute at least once, regardless of the condition, and then repeat based on the condition.
* **Example**: Asking the user if they want to continue after performing an action.
* **Use case**:

do {

printf("Perform an action\n");

printf("Do you want to continue? (1/0): ");

scanf("%d", &choice);

} while (choice == 1);

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

**While loop :**

#include <stdio.h>

int main() {

int i = 1;

while (i <= 10) {

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

i++;

}

return 0;

}

**for Loop:**

#include <stdio.h>

int main() {

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

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

}

return 0;

}

**do-while Loop:**

#include <stdio.h>

int main() {

int i = 1;

do {

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

i++;

} while (i <= 10);

return 0;

}

7. Loop Control Statements

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

**1. break Statement**  
The break statement is used to exit a loop prematurely, regardless of the loop’s condition. It is typically used when a certain condition is met, and the loop should stop immediately.

**Example**:

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

if (i == 5) {

break;

}

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

}

**2. continue Statement**  
The continue statement is used to skip the current iteration of the loop and move to the next iteration. It doesn’t terminate the loop; it just skips the remaining code in that iteration.

**Example**:

c

CopyEdit

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

if (i == 3) {

continue;

}

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

}

**3. goto Statement**  
The goto statement is used to transfer control to a specific labeled part of the program. While it allows for jumping to different sections of code, it is generally avoided in structured programming due to making code harder to read and maintain.

**Example**:

c

CopyEdit

#include <stdio.h>

int main() {

int i = 0;

start:

if (i >= 5) return 0;

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

i++;

goto start;

}

• LAB EXERCISE: o 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() {

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

if (i == 5) {

break;

}

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

}

return 0;

}

**2. Using the continue Statement to Skip Number 3**

#include <stdio.h>

int main() {

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

if (i == 3) {

continue;

}

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

}

return 0;

}

8. Functions in C

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

**What are Functions in C?**  
A function is a block of code that performs a specific task. It helps in breaking down complex problems into smaller, manageable parts. Functions allow code reusability and organization.

* **Function Declaration**: A prototype of the function that tells the compiler about the function’s name, return type, and parameters.
* **Function Definition**: The actual implementation of the function, containing the code that defines what the function does.
* **Function Call**: The act of invoking the function to execute its code.

**Example**:

c

CopyEdit

#include <stdio.h>

int add(int a, int b) { // Function definition

return a + b;

}

int main() {

int result = add(3, 4); // Function call

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

return 0;

}

• LAB EXERCISE: o 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) {

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

return 1;

}

return n \* factorial(n - 1);

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

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

return 0;

}

9. Arrays in C

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

**Concept of Arrays in C**

An **array** in C is a collection of elements of the same data type stored in contiguous memory locations. The array allows you to store multiple values in a single variable, rather than having separate variables for each value.

**Types of Arrays:**

1. **One-Dimensional Array**  
   A one-dimensional array is like a list of values. It stores elements in a single row.

**Example**:

#include <stdio.h>

int main() {

int numbers[5] = {1, 2, 3, 4, 5};

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

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

}

return 0;

}

1. **Multi-Dimensional Array**  
   A multi-dimensional array is an array of arrays. It stores data in a table-like format, such as rows and columns (2D array), or even more dimensions (3D, etc.).

**Example of a 2D Array (Matrix)**:

#include <stdio.h>

int main() {

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

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

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

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

}

printf("\n");

}

return 0;

}

**Differences Between One-Dimensional and Multi-Dimensional Arrays**

| **Aspect** | **One-Dimensional Array** | **Multi-Dimensional Array** |
| --- | --- | --- |
| **Structure** | A single list of elements | An array of arrays (e.g., rows and columns) |
| **Example** | int arr[5] = {1, 2, 3, 4, 5}; | int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}}; |
| **Access** | arr[0], arr[1], etc. | matrix[0][0], matrix[1][2], etc. |
| **Dimensions** | 1 dimension (single row) | 2 or more dimensions (rows, columns, etc.) |

• LAB EXERCISE: o 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.

**One-Dimensional Array Example:**

#include <stdio.h>

int main() {

int arr[5] = {1, 2, 3, 4, 5};

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

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

}

printf("\n");

return 0;

}

**Two-Dimensional Array Example (3x3 Matrix) with Sum:**

#include <stdio.h>

int main() {

int matrix[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

int sum = 0;

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

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

sum += matrix[i][j];

}

}

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

return 0;

}

10. Pointers in C

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

**Pointers in C (Short and Simple)**

* **What are Pointers?**  
  Pointers are variables that store the **memory address** of another variable.
* **Declaration of Pointers**:  
  Syntax:
* type \*pointer\_name;

Example:

int \*ptr;

* **Initialization of Pointers**:  
  To initialize, assign the address of a variable using &:
* int num = 10;
* int \*ptr = &num;
* **Dereferencing Pointers**:  
  Access the value stored at the address using \*:
* printf("%d", \*ptr);
* **Importance of Pointers**:
  1. **Efficient Memory Usage**: Allow dynamic memory allocation.
  2. **Pass-by-Reference**: Modify original values in functions.
  3. **Direct Memory Access**: Access and manipulate memory directly.
* **Example**:
* int num = 20;
* int \*ptr = &num;
* printf("%d", \*ptr); // Prints 20

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

11. Strings in C

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

**1. strlen() Example:**

#include <stdio.h>

#include <string.h>

int main() {

char str[] = "Hello";

printf("Length of string: %d\n", strlen(str));

return 0;

}

**2. strcpy() Example:**

#include <stdio.h>

#include <string.h>

int main() {

char src[] = "Hello";

char dest[10];

strcpy(dest, src);

printf("Copied string: %s\n", dest);

return 0;

}

**3. strcat() Example:**

#include <stdio.h>

#include <string.h>

int main() {

char str1[20] = "Hello ";

char str2[] = "World";

strcat(str1, str2);

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

return 0;

}

**4. strcmp() Example:**

#include <stdio.h>

#include <string.h>

int main() {

char str1[] = "Apple";

char str2[] = "Banana";

int result = strcmp(str1, str2);

if (result < 0) {

printf("str1 is less than str2\n");

} else if (result > 0) {

printf("str1 is greater than str2\n");

} else {

printf("str1 is equal to str2\n");

}

return 0;

}

**5. strchr() Example:**

#include <stdio.h>

#include <string.h>

int main() {

char str[] = "Hello";

char \*result = strchr(str, 'e');

if (result != NULL) {

printf("Character found: %s\n", result);

} else {

printf("Character not found.\n");

}

return 0;

}

* **strlen()**: Find the length of a string (e.g., for loop iterations).
* **strcpy()**: Copy one string into another (e.g., copying user input).
* **strcat()**: Combine strings (e.g., forming a full name).
* **strcmp()**: Compare strings (e.g., password verification).
* **strchr()**: Search for a character (e.g., check if a character exists in a string).

• LAB EXERCISE: o 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().

#include <stdio.h>

#include <string.h>

int main() {

char str1[100], str2[100];

printf("Enter the first string: ");

gets(str1, sizeof(str1), stdin);

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

printf("Enter the second string: ");

gets(str2, sizeof(str2), stdin);

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

strcat(str1, str2);

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

printf("Length of the concatenated string: %lu\n", strlen(str1));

return 0;

}

12. Structures in C

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

**Structures in C (Short and Simple)**

A **structure** in C is a user-defined data type that allows grouping different types of variables (members) under one name. It is used to represent a collection of related data.

**Declaring a Structure:**

A structure is declared using the struct keyword followed by its name and members enclosed in curly braces.

**Syntax**:

struct structure\_name {

type member1;

type member2;

};

**Example**:

struct Person {

char name[50];

int age;

float height;

};

**Initializing a Structure:**

You can initialize a structure when declaring it or later.

**Example**:

struct Person p1 = {"John", 25, 5.9};

**Accessing Structure Members:**

To access structure members, use the dot operator (.) with the structure variable.

**Example**:

#include <stdio.h>

struct Person {

char name[50];

int age;

float height;

};

int main() {

struct Person p1 = {"John", 25, 5.9};

printf("Name: %s\n", p1.name);

printf("Age: %d\n", p1.age);

printf("Height: %.2f\n", p1.height);

return 0;

}

**Key Points:**

* A structure allows grouping different data types.
* Use the . operator to access structure members.

• LAB EXERCISE: o 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.

#include <stdio.h>

struct Student {

char name[50];

int roll\_number;

float marks;

};

int main() {

struct Student students[3];

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

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

printf("Name: ");

fgets(students[i].name, sizeof(students[i].name), stdin);

printf("Roll Number: ");

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

printf("Marks: ");

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

getchar();

}

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

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

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

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

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

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

}

return 0;

}

13. File Handling in C

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

• LAB EXERCISE: o 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. 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() {

char operator;

double num1, num2, result;

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

scanf(" %c", &operator);

printf("Enter two numbers: ");

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

switch (operator) {

case '+':

result = num1 + num2;

printf("Result: %f\n", result);

break;

case '-':

result = num1 - num2;

printf("Result: %f\n", result);

break;

case '\*':

result = num1 \* num2;

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

break;

case '/':

if (num2 != 0)

printf("Result: %2lf\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("Error! Invalid operator.\n");

}

return 0;

}

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

int valid = 0;

while (!valid) {

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

scanf(" %c", &operator);

if (operator == '+' || operator == '-' || operator == '\*' || operator == '/' || operator == '%') {

valid = 1;

} else {

printf("Invalid operator! Please enter a valid one.\n");

}

}

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

**1. Check Even or Odd**

#include <stdio.h>

int main() {

int num;

printf("Enter an integer: ");

scanf("%d", &num);

if (num % 2 == 0)

printf("Even\n");

else

printf("Odd\n");

return 0;

}

**2. Check Positive, Negative, or Zero**

#include <stdio.h>

int main() {

int num;

printf("Enter an integer: ");

scanf("%d", &num);

if (num > 0)

printf("Positive\n");

else if (num < 0)

printf("Negative\n");

else

printf("Zero\n");

return 0;

}

**3. Check if a Number is a Multiple of Both 3 and 5**

#include <stdio.h>

int main() {

int num;

printf("Enter an integer: ");

scanf("%d", &num);

if (num % 3 == 0 && num % 5 == 0)

printf("Multiple of both 3 and 5\n");

else

printf("Not a multiple of both 3 and 5\n");

return 0;

}

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

#include <stdio.h>

int main() {

int marks;

printf("Enter marks: ");

scanf("%d", &marks);

if (marks > 90)

printf("Grade A\n");

else if (marks > 75)

printf("Grade B\n");

else if (marks > 50)

printf("Grade C\n");

else

printf("Grade D\n");

return 0;

}

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

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.

#include <stdio.h>

int main() {

int a, b, c;

printf("Enter three numbers: ");

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

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);

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;

}

• 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);

switch (a > b && a > c) {

case 1:

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

break;

default:

switch (b > a && b > c) {

case 1:

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

break;

default:

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

}

}

switch (a < b && a < c) {

case 1:

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

break;

default:

switch (b < a && b < c) {

case 1:

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

break;

default:

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

}

}

return 0;

}

3. 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.

#include <stdio.h>

int main() {

int num, i, isPrime = 1;

printf("Enter a number: ");

scanf("%d", &num);

if (num <= 1) {

isPrime = 0;

} else {

for (i = 2; i \* i <= num; 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);

return 0;

}

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

#include <stdio.h>

int main() {

int num, i, j, isPrime;

printf("Enter a number: ");

scanf("%d", &num);

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

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

isPrime = 1;

for (j = 2; j \* j <= i; j++) {

if (i % j == 0) {

isPrime = 0;

break;

}

}

if (isPrime)

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.

#include <stdio.h>

int main() {

int num, i;

printf("Enter an integer: ");

scanf("%d", &num);

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

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

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

}

return 0;

}

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

#include <stdio.h>

int main() {

int num, range, i;

printf("Enter an integer: ");

scanf("%d", &num);

printf("Enter the range for the multiplication table: ");

scanf("%d", &range);

printf("Multiplication table of %d from 1 to %d is:\n", num, range);

for (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.

#include <stdio.h>

int main() {

int num, sum = 0;

printf("Enter an integer: ");

scanf("%d", &num);

while (num != 0) {

sum += num % 10;

num /= 10;

}

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

return 0;

}

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

#include <stdio.h>

int main() {

int num, sum = 0, reversed = 0, remainder;

printf("Enter an integer: ");

scanf("%d", &num);

int originalNum = num;

while (num != 0) {

remainder = num % 10;

sum += remainder;

reversed = reversed \* 10 + remainder;

num /= 10;

}

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

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

return 0;

}

4. 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.

#include <stdio.h>

int main() {

int arr[10], i, max, min;

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

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

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

}

max = arr[0];

min = arr[0];

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

if (arr[i] > max) max = arr[i];

if (arr[i] < min) min = arr[i];

}

printf("Max: %d\nMin: %d\n", max, min);

return 0;

}

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

#include <stdio.h>

int main() {

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

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

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

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

}

max = arr[0];

min = arr[0];

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

if (arr[i] > max) max = arr[i];

if (arr[i] < min) min = arr[i];

}

printf("Max: %d\nMin: %d\n", max, min);

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;

}

}

}

printf("Sorted array: ");

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

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

}

printf("\n");

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.

#include <stdio.h>

int main() {

int a[2][2], b[2][2], sum[2][2], i, j;

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

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

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

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

}

}

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

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

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

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

}

}

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

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

sum[i][j] = a[i][j] + b[i][j];

}

}

printf("Resultant matrix:\n");

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

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

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

}

printf("\n");

}

return 0;

}

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

#include <stdio.h>

int main() {

int a[3][3], b[3][3], product[3][3], i, j, k;

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

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

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

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

}

}

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

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

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

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

}

}

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

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

product[i][j] = 0;

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

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

}

}

}

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

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

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

printf("%d ", product[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.

#include <stdio.h>

int main() {

int N, sum = 0;

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

scanf("%d", &N);

int arr[N];

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

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

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

sum += arr[i];

}

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

return 0;

}

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

#include <stdio.h>

int main() {

int N, sum = 0;

float avg;

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

scanf("%d", &N);

int arr[N];

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

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

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

sum += arr[i];

}

avg = (float)sum / N;

printf("Sum: %d\nAverage: %.2f\n", sum, avg);

return 0;

}

5. Functions

LAB EXERCISE 1: Fibonacci Sequence

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

#include <stdio.h>

int fibonacci(int n) {

if (n <= 1)

return n;

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

}

int main() {

int N;

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

scanf("%d", &N);

printf("Fibonacci sequence: ");

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

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

}

printf("\n");

return 0;

}

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

#include <stdio.h>

int fibonacci\_recursive(int n) {

if (n <= 1)

return n;

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

}

int fibonacci\_iterative(int n) {

int a = 0, b = 1, temp;

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

temp = a + b;

a = b;

b = temp;

}

return n == 0 ? a : b;

}

int main() {

int N;

printf("Enter the Nth term: ");

scanf("%d", &N);

printf("Nth Fibonacci (Recursive): %d\n", fibonacci\_recursive(N));

printf("Nth Fibonacci (Iterative): %

LAB EXERCISE 2: Factorial Calculation

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

#include <stdio.h>

int factorial(int n) {

if (n <= 1)

return 1;

return n \* factorial(n - 1);

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

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

return 0;

}

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

#include <stdio.h>

long long factorial\_recursive(int n) {

if (n <= 1)

return 1;

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

}

long long factorial\_iterative(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);

printf("Factorial (Recursive) of %d: %lld\n", num, factorial\_recursive(num));

printf("Factorial (Iterative) of %d: %lld\n", num, factorial\_iterative(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.

#include <stdio.h>

int is\_palindrome(int n) {

int reversed = 0, original = n, remainder;

while (n != 0) {

remainder = n % 10;

reversed = reversed \* 10 + remainder;

n /= 10;

}

return original == reversed;

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

if (is\_palindrome(num))

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

else

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

return 0;

}

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

#include <stdio.h>

#include <string.h>

int is\_palindrome(char str[]) {

int start = 0, end = strlen(str) - 1;

while (start < end) {

if (str[start] != str[end])

return 0;

start++;

end--;

}

return 1;

}

int main() {

char str[100];

printf("Enter a string: ");

scanf("%s", str);

if (is\_palindrome(str))

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

else

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

return 0;

}

6. Strings LAB EXERCISE 1: String Reversal

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

#include <stdio.h>

#include <string.h>

void reverse\_string(char str[]) {

int start = 0, end = strlen(str) - 1;

while (start < end) {

char temp = str[start];

str[start] = str[end];

str[end] = temp;

start++;

end--;

}

}

int main() {

char str[100];

printf("Enter a string: ");

scanf("%s", str);

reverse\_string(str);

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

return 0;

}

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

#include <stdio.h>

void reverse\_string(char str[]) {

int start = 0, end = 0;

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

end++;

}

end--;

while (start < end) {

char temp = str[start];

str[start] = str[end];

str[end] = temp;

start++;

end--;

}

}

int main() {

char str[100];

printf("Enter a string: ");

scanf("%s", str);

reverse\_string(str);

printf("Reversed string: %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.

#include <stdio.h>

int main() {

char str[100];

int vowels = 0, consonants = 0, i;

printf("Enter a string: ");

scanf("%s", str);

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

if (str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u' ||

str[i] == 'A' || str[i] == 'E' || str[i] == 'I' || str[i] == 'O' || str[i] == 'U') {

vowels++;

} else if ((str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z')) {

consonants++;

}

}

printf("Vowels: %d\nConsonants: %d\n", vowels, consonants);

return 0;

}

• 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, i;

printf("Enter a string: ");

scanf("%s", str);

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

if (str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u' ||

str[i] == 'A' || str[i] == 'E' || str[i] == 'I' || str[i] == 'O' || str[i] == 'U') {

vowels++;

} else if ((str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z')) {

consonants++;

} else if (str[i] >= '0' && str[i] <= '9') {

digits++;

} else {

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.

#include <stdio.h>

#include <ctype.h>

int main() {

char str[100];

int words = 0, i = 0;

printf("Enter a sentence: ");

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

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

if (isspace(str[i]) && !isspace(str[i+1]))

words++;

i++;

}

if (str[0] != '\0')

words++;

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

return 0;

}

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

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int main() {

char str[100], longest[100] = "", word[100];

int words = 0, i = 0, j = 0, max\_len = 0;

printf("Enter a sentence: ");

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

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

if (isspace(str[i]) || str[i+1] == '\0') {

word[j] = '\0';

if (j > max\_len) {

max\_len = j;

strcpy(longest, word);

}

j = 0;

if (str[i] != '\0') words++;

} else {

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

}

i++;

}

if (str[0] != '\0') words++;

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

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).

#include<stdio.h>

int main ()

{

    int n,c,r,arm=0;

    printf("enter the number :");

    scanf("%d",&n);

    c=n;

    while (n>0)

    {

           r=n%10;

           arm=(r\*r\*r)+arm;

           n=n/10;

    }

    (c==arm) ? printf("number is armstrong ") : printf("number is not armstrong");

    return 0;

}

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

#include <stdio.h>

int factorial(int n) {

int fact = 1;

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

fact \*= i;

}

return fact;

}

int main() {

int rows;

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

scanf("%d", &rows);

printf("Pascal's Triangle:\n");

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

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

printf("%d ", factorial(i) / (factorial(j) \* factorial(i - j)));

}

printf("\n");

}

return 0;

}

Lab Challenge 2: Pascal’s Triangle

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

#include <stdio.h>

int main() {

int n, i, j, num = 1;

printf("Enter number of rows: ");

scanf("%d", &n);

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

num = 1;

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

printf("%d ", num);

num = num \* (i - j) / (j + 1);

}

printf("\n");

}

return 0;

}

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

#include <stdio.h>

int pascal(int n, int k) {

if (k == 0 || k == n)

return 1;

return pascal(n - 1, k - 1) + pascal(n - 1, k);

}

int main() {

int n, i, j;

printf("Enter number of rows: ");

scanf("%d", &n);

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

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

printf("%d ", pascal(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, attempts = 0;

srand(time(0));

number = rand() % 100 + 1;

printf("Guess the number between 1 and 100: ");

while (attempts < 10) {

scanf("%d", &guess);

attempts++;

if (guess > number)

printf("Too high! Try again: ");

else if (guess < number)

printf("Too low! Try again: ");

else {

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

break;

}

}

if (guess != number)

printf("Sorry, you couldn't guess the number. It was %d.\n", number);

return 0;

}

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

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main() {

int number, guess, attempts = 0;

srand(time(0));

number = rand() % 100 + 1;

printf("Guess the number between 1 and 100: ");

while (attempts < 10) {

scanf("%d", &guess);

attempts++;

if (guess > number)

printf("Too high! Try a smaller number: ");

else if (guess < number)

printf("Too low! Try a larger number: ");

else {

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

break;

}

}

if (guess != number)

printf("Sorry, you couldn't guess the number. It was %d.\n", number);

return 0;

}