**Module 2 – Introduction to Programming**

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

Ans. A successor to the programming language B, C was originally developed at Bell Labs by Ritchie between 1972 and 1973 to construct utilities running on Unix. It was applied to re-implementing the kernel of the Unix operating. During the 1980s, C gradually gained popularity.

Develops valuable skills: CP hones your problem-solving, algorithmic thinking, coding efficiency, and time management skills. These are highly valued by MAANG and other tech giants in their technical interviews.

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. C programming is extensively used in three major areas: embedded systems, operating systems, and game development. Embedded systems rely on C for low-level control and efficiency, operating systems use C for core functionality and interaction with hardware, and game development utilizes C for performance-critical tasks and graphics.

1. 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.

Ans. To install a C compiler like GCC and an IDE like DevC++, VS Code, or Code::Blocks, you'll first need to download and install the appropriate compiler (e.g., MinGW for Windows, or your distribution's package manager for Linux). Then, you'll download and install your chosen IDE. Finally, you'll configure the IDE to recognize the compiler and link it to your projects.

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

Ans. To run your first C program, install a C compiler like GCC (GNU Compiler Collection) and configure an IDE like Visual Studio Code. Then, write the "Hello, World!" program, compile it, and run it.

1. Install a C Compiler:

* **GCC (GNU Compiler Collection):**

GCC is a widely used and free C compiler. You can download and install it on Windows, macOS, and Linux.

* + **Windows:** Download MinGW (Minimalist GNU for Windows) or TDM-GCC. [upGrad](https://www.upgrad.com/tutorials/software-engineering/c-tutorial/guide-to-build-c-hello-world-program/) recommends these.
  + **macOS:** Install the Xcode command-line tools.
  + **Linux (Ubuntu/Debian):** Use sudo apt update and sudo apt install gcc.
* **Verification:**

After installation, open a terminal or command prompt and type gcc --version to verify the compiler is installed correctly

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

Ans. #include <stdio.h>  
  
int main() {  
 int number, month\_number;  
  
 *// Check if a number is even or odd*  
 printf("Enter an integer: ");  
 scanf("%d", &number);  
  
 if (number % 2 == 0) {  
 printf("%d is an even number.\n", number);  
 } else {  
 printf("%d is an odd number.\n", number);  
 }  
  
 *// Display the month name using a switch statement*  
 printf("\nEnter a number between 1 and 12 to display the month name: ");  
 scanf("%d", &month\_number);  
  
 switch (month\_number) {  
 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");

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

Ans. While, for, and do-while loops are different ways to execute a block of code repeatedly. For loops are used when the number of iterations is known in advance, for loops are used when you need to iterate over a collection, and do-while loops ensure the code inside the loop is executed at least once before the condition is checked.

8. 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() {  
 *// Using while loop*  
 int i = 1;  
 printf("Numbers from 1 to 10 using while loop:\n");  
 while (i <= 10) {  
 printf("%d ", i);  
 i++;  
 }  
 printf("\n");  
  
 *// Using for loop*  
 printf("Numbers from 1 to 10 using for loop:\n");  
 for (i = 1; i <= 10; i++) {  
 printf("%d ", i);  
 }  
 printf("\n");  
  
 *// Using do-while loop*  
 i = 1; *// Reset i to 1*  
 printf("Numbers from 1 to 10 using do-while loop:\n");  
 do {  
 printf("%d ", i);  
 i++;  
 } while (i <= 10);  
 printf("\n");

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

Ans. In C, break, continue, and goto are jump statements that control the flow of execution within loops and switch statements. break terminates the current loop or switch statement, continue skips the remaining statements in the current iteration of a loop, and goto jumps to a labeled location within the same function.

10. 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("Using break statement:\n");  
 for (int i = 1; i <= 10; i++) {  
 if (i == 5) {  
 break; *// Exit the loop when i is 5*  
 }  
 printf("%d ", i);  
 }  
  
 printf("\nUsing continue statement:\n");  
 for (int i = 1; i <= 10; i++) {  
 if (i == 3) {  
 continue; *// Skip the current iteration when i is 3*  
 }  
 printf("%d ", i);  
 }  
 printf("\n");

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

Ans. A function in C is a block of code designed to perform a specific task. It promotes code reusability and modularity.

Function Declaration

A function declaration specifies the function's name, return type, and parameters without defining its implementation.

Example:

C

int add(int a, int b);

Function Definition

12. 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 num;  
  
 printf("Enter a non-negative integer: ");  
 scanf("%d", &num);  
  
 *// Function call and output*  
 if (num < 0) {  
 printf("Factorial is not defined for negative numbers.\n");  
 } else {  
 printf("Factorial of %d is %lld\n", num, factorial(num));  
 }  
  
 return 0;  
}  
  
*// Function definition*  
long long factorial(int n) {  
 if (n == 0 || n == 1) {  
 return 1;  
 } else {  
 return n \* factorial(n - 1);  
 }  
}

The program calculates the factorial of a non-negative integer entered by the user. It includes a function declaration for factorial, the function definition, and a call to the function within the main function. The factorial function uses recursion to compute the factorial. It handles the base cases for 0 and 1, and recursively calls itself for other positive integers. A check for negative input is included to handle invalid input.

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

Ans. n C, an array is a collection of data elements of the same type, stored in contiguous memory locations. It provides a way to manage and access multiple values through a single variable name, using indices to identify each element's position.

One-Dimensional Arrays

A one-dimensional array is a linear sequence of elements. It's the simplest form of an array, where elements are arranged in a single row or column.

C

#include <stdio.h>  
  
int main() {  
 int numbers[5] = {10, 20, 30, 40, 50}; *// Declaration and initialization*  
   
 *// Accessing elements*  
 printf("First element: %d\n", numbers[0]); *// Output: 10*  
 printf("Third element: %d\n", numbers[2]); *// Output: 30*  
   
 *// Modifying an element*  
 numbers[1] = 25;  
 printf("Second element after modification: %d\n", numbers[1]); *// Output:*   
   
 return 0;  
}

14. 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.

1. Ans. Store the given integers in the one-dimensional array and print them.
2. Store the given integers in the two-dimensional array.
3. Calculate the sum of all elements in the two-dimensional array.
4. Print the sum.

Step by step solution

**Step 1** . Store and print elements of the one-dimensional array

* + Declare an integer array of size



55

5

: int arr[5];

* + Initialize the array with values: arr[0] = 10; arr[1] = 20; arr[2] = 30; arr[3] = 40; arr[4] = 50;
  + Print the elements using a loop:
    - for (int i = 0; i < 5; i++)
    - printf("%d ", arr[i]);

**Step 2** . Store elements in the two-dimensional array

* + Declare a



3×33 cross 3

3×3

integer array: int matrix[3][3];

* + Initialize the matrix with values:
    - matrix[0][0] = 1; matrix[0][1] = 2; matrix[0][2] = 3;
    - matrix[1][0] = 4; matrix[1][1] = 5; matrix[1][2] = 6;
    - matrix[2][0] = 7; matrix[2][1] = 8; matrix[2][2] = 9;

**Step 3** . Calculate the sum of elements in the two-dimensional array

* + Initialize a variable sum to



00

0

: int sum = 0;

* + Iterate through the matrix using nested loops:
    - for (int i = 0; i < 3; i++)
    - for (int j = 0; j < 3; j++)
    - Add each element to sum: sum += matrix[i][j];

**Step 4** . Print the sum

* + Print the value of sum: printf("Sum of all elements: %d\n", sum);

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

Ans. Pointers in C are variables that store the memory address of another variable. They "point" to the location where data is stored, enabling direct memory manipulation.

Declaration and Initialization

Pointers are declared using an asterisk \* before the variable name, along with the data type they will point to:

C

int \*ptr; // Declares an integer pointer

Initialization involves assigning the memory address of a variable to the pointer, using the address-of operator &:

C

int num = 10;  
int \*ptr = &num; // ptr now holds the address of num

It's also possible to initialize a pointer to NULL, indicating that it doesn't point to any valid memory location:

C

int \*ptr = NULL;

Importance of Pointers

Pointers are crucial in C for several reasons:

* **Dynamic Memory Allocation:**

Pointers enable allocating memory during runtime using functions like malloc() and calloc(), which is essential for creating dynamic data structures.

* **Function Arguments Modification:**

By passing pointers as arguments to functions, modifications made to the pointed-to variables within the function are reflected outside the function scope.

* **Efficient Data Handling:**

Pointers facilitate efficient manipulation of arrays and data structures, avoiding the need to copy large amounts of data.

* **Implementing Data Structures:**

Pointers are fundamental for implementing complex data structures like linked lists, trees, and graphs.

* **Direct Memory Access:**

Pointers provide direct access to memory locations, allowing for low-level operations and system programming.

16. 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;  
 int \*ptr;  
 ptr = &num;  
 printf("Value of num before modification: %d\n", num);  
 \*ptr = 20;  
 printf("Value of num after modification: %d\n", num);  
 return 0;  
}

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

Ans. String handling functions in C, like strlen(), strcpy(), strcat(), strcmp(), and strchr(), are essential for working with text in C programs. strlen() returns the length of a string, strcpy() copies a string, strcat() concatenates strings, strcmp() compares strings, and strchr() searches for a character within a string.

18. 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];  
 char str2[100];  
  
 printf("Enter the first string: ");  
 fgets(str1, sizeof(str1), stdin);  
 str1[strcspn(str1, "\n")] = 0;  
  
 printf("Enter the second string: ");  
 fgets(str2, sizeof(str2), stdin);  
 str2[strcspn(str2, "\n")] = 0;  
   
 strcat(str1, str2);  
  
 printf("Concatenated string: %s\n", str1);  
 printf("Length of concatenated string: %zu\n", strlen(str1));  
  
 return 0;  
}

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

Ans. In C, a structure is a composite data type that groups together variables of potentially different types under a single name. It is defined using the struct keyword. Structures are useful for representing real-world entities or organizing related data.

Declaration

A structure is declared using the following syntax:

C

struct struct\_name {  
 data\_type member1;  
 data\_type member2;  
 *// ...*  
};

20. 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> *// Required for strcpy*  
  
*// Define the structure to store student details*  
struct Student {  
 char name[50]; *// Store the student's name*  
 int rollNo; *// Store the student's roll number*  
 float marks; *// Store the student's marks*  
};  
  
int main() {  
 *// Declare an array of structures to store details of 3 students*  
 struct Student students[3];  
  
 *// Input details for each student*  
 for (int i = 0; i < 3; i++) {  
 printf("Enter details for student %d:\n", i + 1);  
  
 *// Input name (using fgets to handle spaces)*  
 printf("Name: ");  
 fgets(students[i].name, sizeof(students[i].name), stdin);  
 *// Remove the trailing newline character added by fgets*  
  
 *// Input roll number*  
 printf("Roll Number: ");  
 scanf("%d", &students[i].rollNo);  
 *// Consume the newline character after scanf*  
 while (getchar() != '\n');  
  
 *// Input marks*  
 printf("Marks: ");  
 scanf("%f", &students[i].marks);  
 *// Consume the newline character after scanf*  
 while (getchar() != '\n');  
 }  
  
 *// Print the details of each student*  
 printf("\n--- 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].rollNo);  
 printf("Marks: %.2f\n", students[i].marks);  
 printf("----------------------\n");  
 }  
  
 return 0;  
}

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

Ans. File handling in C is crucial for enabling persistent data storage and retrieval, allowing programs to work with data beyond their runtime. It's essential for creating, reading, writing, and modifying data stored in files, making it a fundamental aspect of many C programs. Key file operations include opening files, closing files, reading data from files, and writing data to files.

Importance of File Handling in C:

* **Data Persistence:**

File handling enables programs to store data permanently in files on the computer's storage, ensuring that data is retained even after the program terminates.

22. 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.

Ans. include <stdio.h>  
#include <stdlib.h>  
  
#define MAX\_LINE\_LENGTH 256  
  
int main() {  
 FILE \*file;  
 char filename[] = "my\_file.txt";  
 char stringToWrite[] = "Hello, Maninagar, Ahmedabad!\n";  
 char line[MAX\_LINE\_LENGTH];  
  
 *// 1. Open the file in write mode ("w")*  
 file = fopen(filename, "w");  
  
 if (file == NULL) {  
 printf("Error opening file!\n");  
 return 1;  
 }  
  
 *// 2. Write a string to the file*  
 fprintf(file, "%s", stringToWrite); *// Use fprintf for formatted output*  
   
 *// 3. Close the file*  
 fclose(file);  
  
 *// 4. Open the file in read mode ("r")*  
 file = fopen(filename, "r");  
  
 if (file == NULL) {  
 printf("Error opening file for reading!\n");  
 return 1;  
 }  
  
 *// 5. Read the file and display its contents*  
 printf("Contents of %s:\n", filename);  
  
 while (fgets(line, MAX\_LINE\_LENGTH, file) != NULL) { *// Use fgets for line-by-line reading*  
 printf("%s", line);  
 }  
  
 *// 6. Close the file*  
 fclose(file);

1. Operators LAB EXERCISE 1: Simple Calculator

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

Ans. A C program that performs basic arithmetic operations based on user input.

What's given in the problem

* The program should take two numbers as input.
* The program should take an operator as input.
* The program should perform the operation.
* The program should handle invalid operator input.

Helpful information

* The scanf function can be used to read input from the user.
* The printf function can be used to print output to the user.
* The switch statement can be used to perform different actions based on the value of a variable.
* The modulo operator % returns the remainder of a division.

How to solve

1. Get input from the user.
2. Perform the calculation.
3. Print the result.
4. Handle invalid input.

**Step 1**.Get input from the user

* + Prompt the user to enter the first number.
    - printf("Enter the first number: ");
  + Read the first number using scanf.
    - scanf("%lf", &num1);
  + Prompt the user to enter the second number.
    - printf("Enter the second number: ");
  + Read the second number using scanf.
    - scanf("%lf", &num2);
  + Prompt the user to enter the operator.
    - printf("Enter the operator (+, -, \*, /, %): ");
  + Read the operator using scanf.
    - scanf(" %c", &operator);

**Step 2**.Perform the calculation using switch statement

**Step 3**.Print the result

24. 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.

Ans. include <stdio.h>  
  
int main() {  
 int number;  
  
 printf("Enter an integer: ");  
 scanf("%d", &number);  
  
 *// Check if the number is even or odd*  
 if (number % 2 == 0) {  
 printf("%d is an even number.\n", number);  
 } else {  
 printf("%d is an odd number.\n", number);  
 }  
  
 *// Check if the number is positive, negative, or zero*  
 if (number > 0) {  
 printf("%d is a positive number.\n", number);  
 } else if (number < 0) {  
 printf("%d is a negative number.\n", number);  
 } else {  
 printf("%d is zero.\n", number);  
 }  
  
 *// Check if the number is a multiple of both 3 and 5*  
 if (number % 3 == 0 && number % 5 == 0) {  
 printf("%d is a multiple of both 3 and 5.\n", number);  
 } else {  
 printf("%d is not a multiple of both 3 and 5.\n", number);  
 }  
  
 return 0;  
}

Explanation:

1. **1. Input:**

The program takes an integer input from the user using scanf().

1. **2. Even/Odd Check:**

The % (modulo) operator is used to find the remainder when number is divided by 2. If the remainder is 0, the number is even. Otherwise, it's odd.

1. **3. Positive/Negative/Zero Check:**

The program uses if-else if-else statements to check the sign of the number:

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

* Ans. Marks greater than



9090

90

: Grade A

* Marks greater than



7575

75

and less than or equal to



9090

90

: Grade B

* Marks greater than



5050

50

and less than or equal to



7575

75

: Grade C

* Marks less than or equal to



5050

50

: Grade D

How to solve

Use if-else statements to check the conditions and print the corresponding grade.

**Step 1** . Get input from the user.

* + Declare an integer variable marks.
  + Prompt the user to enter the marks.
  + Read the marks using scanf.

**Step 2** . Check the conditions and print the grade.

* + Use if-else if-else statements to check the conditions.
  + If marks > 90, print "Grade A".
  + Else if marks > 75 and marks <= 90, print "Grade B".
  + Else if marks > 50 and marks <= 75, print "Grade C".
  + Else print "Grade D".

Solution

C

#include <stdio.h>  
int main() {  
 int marks;  
 printf("Enter the marks: ");  
 scanf("%d", &marks);  
 if (marks > 90) {  
 printf("Grade A\n");  
 } else if (marks > 75 && marks <= 90) {  
 printf("Grade B\n");  
 } else if (marks > 50 && marks <= 75) {  
 printf("Grade C\n");  
 } else {  
 printf("Grade D\n");  
 }  
 return 0;  
}

25. Write a C program that takes three numbers from the user and determines: o The largest number. o The smallest number. • Challenge: Solve the problem using both if-else and switch-case statements.

Ans. #include <iostream>  
#include <algorithm>  
  
using namespace std;  
  
int main() {  
 int num1, num2, num3;  
  
 *// Input three numbers from the user*  
 cout << "Enter three numbers: ";  
 cin >> num1 >> num2 >> num3;  
  
 *// Method 1: Using if-else statements*  
 if (num1 >= num2 && num1 >= num3) {  
 cout << "Largest (if-else): " << num1 << endl;  
 if (num2 <= num3) {  
 cout << "Smallest (if-else): " << num2 << endl;  
 } else {  
 cout << "Smallest (if-else): " << num3 << endl;  
 }  
 } else if (num2 >= num1 && num2 >= num3) {  
 cout << "Largest (if-else): " << num2 << endl;  
 if (num1 <= num3) {  
 cout << "Smallest (if-else): " << num1 << endl;  
 } else {  
 cout << "Smallest (if-else): " << num3 << endl;  
 }  
 } else {  
 cout << "Largest (if-else): " << num3 << endl;  
 if (num1 <= num2) {  
 cout << "Smallest (if-else): " << num1 << endl;  
 } else {  
 cout << "Smallest (if-else): " << num2 << endl;  
 }  
 }  
  
 *// Method 2: Using switch-case statements*  
 int temp = num1;  
 switch (true) {  
 case (num1 > num2):  
 if (num1 > num3) {  
 temp = num1;  
 } else {  
 temp = num3;  
 }  
 break;  
 case (num2 > num1):  
 if (num2 > num3) {  
 temp = num2;  
 } else {  
 temp = num3;  
 }  
 break;  
 default:  
 temp = num3; *// In case all numbers are equal*  
 }  
 cout << "Largest (switch-case): " << temp << endl;  
   
 int min\_val = num1;  
 switch(true) {  
 case (num1 < num2 && num1 < num3):  
 min\_val = num1;  
 break;  
 case (num2 < num1 && num2 < num3):  
 min\_val = num2;  
 break;  
 default:  
 min\_val = num3;  
 }  
 cout << "Smallest (switch-case): " << min\_val << endl;  
}

26. • Write a C program that checks whether a given number is a prime number or not using a for loop. • Challenge: Modify the program to print all prime numbers between 1 and a given number.

Ans. #include <stdio.h>  
#include <stdbool.h>  
  
*// Function to check if a number is prime*  
bool isPrime(int num) {  
 if (num <= 1) return false;  
 for (int i = 2; i \* i <= num; i++) {  
 if (num % i == 0) return false;  
 }  
 return true;  
}  
  
int main() {  
 int number, limit;  
  
 *// Check a single number for primality*  
 printf("Enter a number to check if it's prime: ");  
 scanf("%d", &number);  
  
 if (isPrime(number)) {  
 printf("%d is a prime number.\n", number);  
 } else {  
 printf("%d is not a prime number.\n", number);  
 }  
  
 *// Print all prime numbers up to a limit*  
 printf("Enter a limit to print prime numbers up to: ");  
 scanf("%d", &limit);  
  
 printf("Prime numbers between 1 and %d are:\n", limit);  
 for (int i = 2; i <= limit; i++) {  
 if (isPrime(i)) {  
 printf("%d ", i);  
 }  
 }  
 printf("\n");  
  
 return 0;  
}

27. Write a C program that takes an integer input from the user and prints its multiplication table using a for loop. • Challenge: Allow the user to input the range of the multiplication table (e.g., from 1 to N)

Ans. #include <stdio.h>  
  
int main() {  
 int number, range;  
  
 printf("Enter the number to display its multiplication table: ");  
 scanf("%d", &number);  
  
 printf("Enter the range for the multiplication table: ");  
 scanf("%d", &range);  
  
 printf("Multiplication table of %d:\n", number);  
  
 for (int i = 1; i <= range; i++) {  
 printf("%d x %d = %d\n", number, i, number \* i);  
 }  
  
 return 0;

28. • Write a C program that takes an integer from the user and calculates the sum of its digits using a while loop. • Challenge: Extend the program to reverse the digits of the number.

Ans. #include <stdio.h>  
  
int main() {  
 int num, originalNum, remainder, sum = 0, reversedNum = 0;  
  
 *// Input from the user*  
 printf("Enter an integer: ");  
 scanf("%d", &num);  
  
 *// Calculate the sum of digits*  
 originalNum = num;  
 while (num > 0) {  
 remainder = num % 10;  
 sum += remainder;  
 num /= 10;  
 }  
 printf("Sum of digits: %d\n", sum);  
  
 *// Reverse the digits*  
 num = originalNum;  
 while (num > 0) {  
 remainder = num % 10;  
 reversedNum = (reversedNum \* 10) + remainder;  
 num /= 10;  
 }  
 printf("Reversed number: %d\n", reversedNum);

}

29. • Write a C program that accepts 10 integers from the user and stores them in an array. The program should then find and print the maximum and minimum values in the array. • Challenge: Extend the program to sort the array in ascending order.

Ans. #include <stdio.h>  
  
int main() {  
 int numbers[10];  
 int i, j, temp, max, min;  
  
 *// Input 10 integers from the user*  
 printf("Enter 10 integers: ");  
 for (i = 0; i < 10; i++) {  
 scanf("%d", &numbers[i]);  
 }  
  
 *// Find maximum and minimum values*  
 max = numbers[0];  
 min = numbers[0];  
 for (i = 1; i < 10; i++) {  
 if (numbers[i] > max) {  
 max = numbers[i];  
 }  
 if (numbers[i] < min) {  
 min = numbers[i];  
 }  
 }  
  
 *// Print the maximum and minimum values*  
 printf("Maximum value: %d\n", max);  
 printf("Minimum value: %d\n", min);  
  
 *// Sort the array in ascending order (Bubble Sort)*  
 for (i = 0; i < 10 - 1; i++) {  
 for (j = 0; j < 10 - i - 1; j++) {  
 if (numbers[j] > numbers[j + 1]) {  
 temp = numbers[j];  
 numbers[j] = numbers[j + 1];  
 numbers[j + 1] = temp;  
 }  
 }  
 }  
  
 *// Print the sorted array*  
 printf("Sorted array: ");  
 for (i = 0; i < 10; i++) {  
 printf("%d ", numbers[i]);  
 }  
 printf("\n");  
  
 return 0;  
}

30. Write a C program that accepts two 2x2 matrices from the user and adds them. Display the resultant matrix. • Challenge: Extend the program to work with 3x3 matrices and matrix multiplication.

Ans. #include <stdio.h>  
  
*// Function to input matrix elements from the user*  
void inputMatrix(int matrix[][3], int rows, int cols) {  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 printf("Enter element at position [%d][%d]: ", i + 1, j + 1);  
 scanf("%d", &matrix[i][j]);  
 }  
 }  
}  
  
*// Function to add two matrices*  
void addMatrices(int matrix1[][3], int matrix2[][3], int result[][3], int rows, int cols) {  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 result[i][j] = matrix1[i][j] + matrix2[i][j];  
 }  
 }  
}  
  
*// Function to multiply two matrices*  
void multiplyMatrices(int matrix1[][3], int matrix2[][3], int result[][3], int rows1, int cols1, int rows2, int cols2) {  
 for (int i = 0; i < rows1; i++) {  
 for (int j = 0; j < cols2; j++) {  
 result[i][j] = 0; *// Initialize the element to 0*  
 for (int k = 0; k < cols1; k++) {  
 result[i][j] += matrix1[i][k] \* matrix2[k][j];  
 }  
 }  
 }  
}  
  
*// Function to display the matrix*  
void displayMatrix(int matrix[][3], int rows, int cols) {  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 printf("%d ", matrix[i][j]);  
 }  
 printf("\n");  
 }  
}  
  
int main() {  
 int matrixA[3][3], matrixB[3][3], matrixSum[3][3], matrixProduct[3][3];  
 int rows = 3, cols = 3; *// Assuming 3x3 matrices*  
  
 printf("Enter elements for matrix A:\n");  
 inputMatrix(matrixA, rows, cols);  
  
 printf("\nEnter elements for matrix B:\n");  
 inputMatrix(matrixB, rows, cols);  
  
 printf("\nMatrix A:\n");  
 displayMatrix(matrixA, rows, cols);  
  
 printf("\nMatrix B:\n");  
 displayMatrix(matrixB, rows, cols);  
  
 *// Matrix Addition*  
 addMatrices(matrixA, matrixB, matrixSum, rows, cols);  
 printf("\nMatrix Sum (A + B):\n");  
 displayMatrix(matrixSum, rows, cols) *// Matrix Multiplication*  
 multiplyMatrices(matrixA, matrixB, matrixProduct, rows, cols, rows, cols);  
 printf("\nMatrix Product (A \* B):\n");  
 displayMatrix(matrixProduct, rows, cols);  
  
 return 0;  
}

31. Write a C program that takes N numbers from the user and stores them in an array. The program should then calculate and display the sum of all array elements. • Challenge: Modify the program to also find the average of the numbers.

Ans. #include <stdio.h>  
  
int main() {  
 int n, i, sum = 0;  
 float average;  
  
 *// Get the number of elements from the user*  
 printf("Enter the number of elements: ");  
 scanf("%d", &n);  
  
 *// Declare an array to store the elements*  
 int arr[n];  
  
 *// Get the elements from the user and store them in the array*  
 printf("Enter the elements: ");  
 for (i = 0; i < n; i++) {  
 scanf("%d", &arr[i]);  
 }  
  
 *// Calculate the sum of the elements*  
 for (i = 0; i < n; i++) {  
 sum += arr[i];  
 }  
  
 *// Calculate the average of the elements*  
 average = (float)sum / n;  
  
 *// Display the sum and average*  
 printf("Sum of elements: %d\n", sum);  
 printf("Average of elements: %.2f\n", average);  
  
 return 0;  
}

32. Write a C program that generates the Fibonacci sequence up to N terms using a recursive function. • Challenge: Modify the program to calculate the Nth Fibonacci number using both iterative and recursive methods. Compare their efficiency.

Ans. #include <stdio.h>  
#include <time.h> *// For measuring execution time*  
  
*// Recursive Fibonacci function to calculate the nth Fibonacci number*  
int fibonacciRecursive(int n) {  
 if (n <= 1) {  
 return n;  
 }  
 return fibonacciRecursive(n - 1) + fibonacciRecursive(n - 2);  
}  
  
*// Iterative Fibonacci function to calculate the nth Fibonacci number*  
int fibonacciIterative(int n) {  
 if (n <= 1) {  
 return n;  
 }  
 int a = 0, b = 1, temp;  
 for (int i = 2; i <= n; i++) {  
 temp = a + b;  
 a = b;  
 b = temp;  
 }  
 return b;  
}  
  
*// Function to generate the Fibonacci sequence up to n terms (recursive)*  
void generateFibonacciRecursive(int n) {  
 if (n <= 0) {  
 return;  
 }  
 for (int i = 0; i < n; i++) {  
 printf("%d ", fibonacciRecursive(i));  
 }  
 printf("\n");  
}  
  
*// Function to generate the Fibonacci sequence up to n terms (iterative)*  
void generateFibonacciIterative(int n) {  
 if (n <= 0) {  
 return;  
 }  
 int a = 0, b = 1;  
 printf("%d %d ", a, b); *// Print the first two terms*  
 for (int i = 2; i < n; i++) {  
 int temp = a + b;  
 printf("%d ", temp);  
 a = b;  
 b = temp;  
 }  
 printf("\n");  
}  
  
int main() {  
 int n;  
 printf("Enter the number of terms: ");  
 scanf("%d", &n);  
  
 *// Generate Fibonacci sequence using recursion*  
 printf("Fibonacci sequence (recursive):\n");  
 generateFibonacciRecursive(n);  
  
 *// Generate Fibonacci sequence using iteration*  
 printf("Fibonacci sequence (iterative):\n");  
 generateFibonacciIterative(n);  
  
 *// Calculate the nth Fibonacci number using both methods*  
 printf("The %dth Fibonacci number (recursive): %d\n", n, fibonacciRecursive(n));  
 printf("The %dth Fibonacci number (iterative): %d\n", n, fibonacciIterative(n));  
  
 *// Efficiency comparison (measure execution time)*  
 clock\_t start\_recursive, end\_recursive, start\_iterative, end\_iterative;  
  
 *// Recursive*  
 start\_recursive = clock();  
 fibonacciRecursive(n); *// Calculate (or just call to execute)*  
 end\_recursive = clock();  
  
 *// Iterative*  
 start\_iterative = clock();  
 fibonacciIterative(n); *// Calculate (or just call to execute)*  
 end\_iterative = clock();  
  
 double time\_recursive = ((double)(end\_recursive - start\_recursive)) / CLOCKS\_PER\_SEC;  
 double time\_iterative = ((double)(end\_iterative - start\_iterative)) / CLOCKS\_PER\_SEC;  
  
 printf("\nTime taken by recursive method: %f seconds\n", time\_recursive);  
 printf("Time taken by iterative method: %f seconds\n", time\_iterative);  
  
 return 0;  
}

33. Write a C program that calculates the factorial of a given number using a function. • Challenge: Implement both an iterative and a recursive version of the factorial function and compare their performance for large numbers.

Ans. #include <stdio.h>  
#include <time.h> *// For performance testing*  
  
*// Iterative factorial function*  
unsigned long long factorial\_iterative(int n) {  
 if (n < 0) {  
 return -1; *// Handle negative input*  
 }  
 unsigned long long result = 1;  
 for (int i = 1; i <= n; ++i) {  
 result \*= i;  
 }  
 return result;  
}  
  
*// Recursive factorial function*  
unsigned long long factorial\_recursive(int n) {  
 if (n < 0) {  
 return -1; *// Handle negative input*  
 }  
 if (n == 0 || n == 1) {  
 return 1;  
 }  
 return n \* factorial\_recursive(n - 1);  
}  
  
int main() {  
 int num;  
  
 *// Get input from the user*  
 printf("Enter a non-negative integer: ");  
 scanf("%d", &num);  
  
 *// Perform iterative factorial calculation*  
 clock\_t start\_time\_iter = clock();  
 unsigned long long factorial\_iter = factorial\_iterative(num);  
 clock\_t end\_time\_iter = clock();  
  
 *// Perform recursive factorial calculation*  
 clock\_t start\_time\_rec = clock();  
 unsigned long long factorial\_rec = factorial\_recursive(num);  
 clock\_t end\_time\_rec = clock();  
  
  
 *// Print the results and timings*  
 if (factorial\_iter != -1 && factorial\_rec != -1) {  
 printf("Factorial (Iterative): %llu\n", factorial\_iter);  
 printf("Time (Iterative): %f seconds\n", (double)(end\_time\_iter - start\_time\_iter) / CLOCKS\_PER\_SEC);  
 printf("Factorial (Recursive): %llu\n", factorial\_rec);  
 printf("Time (Recursive): %f seconds\n", (double)(end\_time\_rec - start\_time\_rec) / CLOCKS\_PER\_SEC);  
 } else {  
 printf("Invalid input. Please enter a non-negative integer.\n");  
 }  
  
 return 0;  
}

Key improvements and explanations:

* **Clearer function names:**

factorial\_iterative and factorial\_recursive make the code easier to understand.

* **Error Handling:**

The code now handles negative input and returns -1, which is checked in the main function to prevent crashes or undefined behavior.

* unsigned long long:

Uses unsigned long long data type to handle larger factorial values and avoid integer overflow for larger inputs, especially for recursive calls.

* **Performance Timing:**

The code uses clock() to measure the execution time of both the iterative and recursive functions. This allows for a comparison of their performance on a given input. The CLOCKS\_PER\_SEC constant provides the necessary scaling factor for the time.

* **Detailed Output:**

The output now includes the calculated factorial and the execution time for both methods.

* **Concise and Readable Code:**

The code is formatted for readability and uses comments to explain important parts.

How to compile and run the code:

1. **Save:** Save the code as a .c file (e.g., factorial.c).
2. **Compile:** Open a terminal or command prompt and compile the code using a C compiler (like GCC):

Code

gcc factorial.c -o factorial

1. **Run:** Execute the compiled program:

Code

./factorial

The program will prompt you to enter a number, and it will then display the calculated factorial and the execution times for the iterative and recursive methods.

Performance Comparison:

For smaller numbers, the difference in performance between iterative and recursive solutions is often negligible. However, as the input number increases, the recursive approach's overhead (function call stack, memory allocation) becomes more noticeable. The iterative approach will generally be faster, especially for larger values. You can experiment with different input values to observe the performance differences.

34. Write a C program that takes a number as input and checks whether it is a palindrome using a function. • Challenge: Modify the program to check if a given string is a palindrome.

Ans. #include <stdio.h>  
#include <string.h>  
#include <stdbool.h>  
  
*// Function to check if a number is a palindrome*  
bool isPalindromeNumber(int num) {  
 int reversedNum = 0, remainder, originalNum;  
 originalNum = num;  
  
 while (num != 0) {  
 remainder = num % 10;  
 reversedNum = reversedNum \* 10 + remainder;  
 num /= 10;  
 }  
  
 return originalNum == reversedNum;  
}  
  
*// Function to check if a string is a palindrome*  
bool isPalindromeString(char str[]) {  
 int left = 0;  
 int right = strlen(str) - 1;  
  
 while (left < right) {  
 if (str[left] != str[right]) {  
 return false;  
 }  
 left++;  
 right--;  
 }  
 return true;  
}  
  
int main() {  
 int numberInput;  
 char stringInput[100];  
  
 *// Check for number palindrome*  
 printf("Enter a number: ");  
 scanf("%d", &numberInput);  
 if (isPalindromeNumber(numberInput)) {  
 printf("%d is a palindrome.\n", numberInput);  
 } else {  
 printf("%d is not a palindrome.\n", numberInput);  
 }  
  
 *// Check for string palindrome*  
 printf("Enter a string: ");  
 scanf(" %s", stringInput); *// Add a space before %s to consume the newline character*  
 if (isPalindromeString(stringInput)) {  
 printf("%s is a palindrome.\n", stringInput);  
 } else {  
 printf("%s is not a palindrome.\n", stringInput);  
 }  
  
 return 0;  
}

35. Write a C program that takes a string as input and reverses it using a function. • Challenge: Write the program without using built-in string handling functions.

Ans. #include <stdio.h>  
  
*// Function to reverse a string*  
void reverseString(char \*str) {  
 int length = 0;  
 int i, j;  
 char temp;  
  
 *// Calculate the length of the string*  
 while (str[length] != '\0') {  
 length++;  
 }  
  
 *// Reverse the string by swapping characters from the beginning and end*  
 for (i = 0, j = length - 1; i < j; i++, j--) {  
 temp = str[i];  
 str[i] = str[j];  
 str[j] = temp;  
 }  
}  
  
int main() {  
 char inputString[100];  
  
 *// Get input string from the user*  
 printf("Enter a string: ");  
 scanf("%[^\n]s", inputString);  
  
 *// Reverse the string using the function*  
 reverseString(inputString);  
  
 *// Print the reversed string*  
 printf("Reversed string: %s\n", inputString);  
  
 return 0;  
}

36. Write a C program that takes a string from the user and counts the number of vowels and consonants in the string. • Challenge: Extend the program to also count digits and special characters.

Ans. #include <stdio.h>  
#include <string.h>  
#include <ctype.h>  
  
int main() {  
 char str[100];  
 int vowels = 0, consonants = 0, digits = 0, specialChars = 0;  
 int i;  
  
 printf("Enter a string: ");  
 fgets(str, sizeof(str), stdin);  
  
 *// Remove the newline character from fgets input*  
 str[strcspn(str, "\n")] = 0;  
  
 for (i = 0; str[i] != '\0'; i++) {  
 char ch = str[i];  
  
 if (isalpha(ch)) { *// Check if it's an alphabet*  
 if (tolower(ch) == 'a' || tolower(ch) == 'e' || tolower(ch) == 'i' || tolower(ch) == 'o' || tolower(ch) == 'u') {  
 vowels++;  
 } else {  
 consonants++;  
 }  
 } else if (isdigit(ch)) { *// Check if it's a digit*  
 digits++;  
 } else { *// Otherwise, it's a special character*  
 specialChars++;  
 }  
 }  
  
 printf("Vowels: %d\n", vowels);  
 printf("Consonants: %d\n", consonants);  
 printf("Digits: %d\n", digits);  
 printf("Special characters: %d\n", specialChars);  
  
 return 0;  
}

Explanation:

1. **1. Include Headers:**
   * stdio.h: For standard input/output functions like printf and scanf.
   * string.h: For string manipulation functions like strlen.
   * ctype.h: For character classification functions like isalpha, isdigit, and tolower.
2. **2. Declare Variables:**
   * str[100]: A character array (string) to store the input string.
   * vowels, consonants, digits, specialChars: Integer variables to store the counts of each character type.
   * i: An integer variable for the loop counter.
3. **3. Get Input:**
   * printf("Enter a string: ");: Prompts the user to enter a string.
   * fgets(str, sizeof(str), stdin);: Reads the input string from the standard input and stores it in the str array. fgets is safer than gets as it prevents buffer overflows.
   * str[strcspn(str, "\n")] = 0;: Removes the newline character (\n) that fgets might add at the end of the input.
4. **4. Iterate through the String:**
   * for (i = 0; str[i] != '\0'; i++): A loop that iterates through each character of the string until the null terminator (\0) is encountered.
5. **5. Character Classification:**
   * char ch = str[i];: Gets the current character.
   * if (isalpha(ch)): Checks if the character is an alphabet (a-z or A-Z).
     + If it's an alphabet:
       - if (tolower(ch) == 'a' || tolower(ch) == 'e' || tolower(ch) == 'i' || tolower(ch) == 'o' || tolower(ch) == 'u'): Checks if the lowercase version of the character is a vowel.
         * If it's a vowel, increment vowels.
         * Otherwise (it's a consonant), increment consonants.
   * else if (isdigit(ch)): Checks if the character is a digit (0-9).
     + If it's a digit, increment digits.
   * else: If it's not an alphabet or digit, it's a special character.
     + Increment specialChars.
6. **6. Print Results:**
   * printf("Vowels: %d\n", vowels);, etc.: Prints the counts of each character type.
7. **7. Return 0:**
   * return 0;: Indicates that the program executed successfully

37. • Write a C program that counts the number of words in a sentence entered by the user. • Challenge: Modify the program to find the longest word in the sentence.

Ans. **#**include <stdio.h>  
#include <string.h>  
#include <ctype.h>  
  
int main() {  
 char sentence[1000];  
 char \*token;  
 int word\_count = 0;  
 int max\_length = 0;  
 char longest\_word[100];  
  
 printf("Enter a sentence: ");  
 fgets(sentence, sizeof(sentence), stdin);  
  
 *// Remove trailing newline character if present*  
 sentence[strcspn(sentence, "\n")] = 0;  
  
 token = strtok(sentence, " ");  
 while (token != NULL) {  
 word\_count++;  
 if (strlen(token) > max\_length) {  
 max\_length = strlen(token);  
 strcpy(longest\_word, token);  
 }  
 token = strtok(NULL, " ");  
 }  
  
 printf("Number of words: %d\n", word\_count);  
 printf("Longest word: %s\n", longest\_word);  
  
 return 0;  
}

This program uses strtok to split the input sentence into individual words, counts the words, and tracks the longest word encountered. It handles sentences with multiple spaces and punctuation correctly. The fgets function is used for safer input reading, and the trailing newline character is removed to avoid issues with strtok

38. Write a C program that checks whether a given number is an Armstrong number or not (e.g., 153 = 1^3 + 5^3 + 3^3). • Challenge: Write a program to find all Armstrong numbers between 1 and 1000.

Ans. include <stdio.h>  
#include <math.h>  
  
*// Function to check if a number is an Armstrong number*  
int isArmstrong(int num) {  
 int originalNum, remainder, n = 0, result = 0;  
  
 originalNum = num;  
  
 *// Count the number of digits*  
 while (originalNum != 0) {  
 originalNum /= 10;  
 ++n;  
 }  
  
 originalNum = num;  
  
 *// Calculate the sum of digits raised to the power of n*  
 while (originalNum != 0) {  
 remainder = originalNum % 10;  
 result += pow(remainder, n);  
 originalNum /= 10;  
 }  
  
 *// Check if the number is an Armstrong number*  
 if (result == num)  
 return 1;  
 else  
 return 0;  
}  
  
int main() {  
 int num, i;  
  
 *// Check if a given number is an Armstrong number*  
 printf("Enter a number: ");  
 scanf("%d", &num);  
  
 if (isArmstrong(num))  
 printf("%d is an Armstrong number.\n", num);  
 else  
 printf("%d is not an Armstrong number.\n", num);  
  
 *// Find all Armstrong numbers between 1 and 1000*  
 printf("Armstrong numbers between 1 and 1000 are:\n");  
 for (i = 1; i <= 1000; i++) {  
 if (isArmstrong(i))  
 printf("%d ", i);  
 }  
 printf("\n");  
  
 return 0;  
}

39. Write a C program that generates Pascal’s Triangle up to N rows using loops. • Challenge: Implement the same program using a recursive function.

Ans. #include <stdio.h>  
  
*// Function to generate Pascal's Triangle using loops*  
void pascalTriangleLoops(int n) {  
 for (int i = 0; i < n; i++) {  
 int num = 1;  
 *// Print leading spaces*  
 for (int j = 0; j < n - i - 1; j++) {  
 printf(" ");  
 }  
 *// Print numbers*  
 for (int j = 0; j <= i; j++) {  
 printf("%4d", num);  
 num = num \* (i - j) / (j + 1);  
 }  
 printf("\n");  
 }  
}  
  
*// Function to calculate the value at a specific position using recursion*  
int pascalValueRecursive(int row, int col) {  
 if (col == 0 || col == row) {  
 return 1;  
 } else {  
 return pascalValueRecursive(row - 1, col - 1) + pascalValueRecursive(row - 1, col);  
 }  
}  
  
*// Function to generate Pascal's Triangle using recursion*  
void pascalTriangleRecursive(int n) {  
 for (int i = 0; i < n; i++) {  
 *// Print leading spaces*  
 for (int j = 0; j < n - i - 1; j++) {  
 printf(" ");  
 }  
 *// Print numbers*  
 for (int j = 0; j <= i; j++) {  
 printf("%4d", pascalValueRecursive(i, j));  
 }  
 printf("\n");  
 }  
}  
  
int main() {  
 int numRows;  
 printf("Enter the number of rows for Pascal's Triangle: ");  
 scanf("%d", &numRows);  
  
 printf("\nPascal's Triangle using loops:\n");  
 pascalTriangleLoops(numRows);  
  
 printf("\nPascal's Triangle using recursion:\n");  
 pascalTriangleRecursive(numRows);  
   
 return 0;

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

Ans. #include <iostream>  
#include <cstdlib>  
#include <ctime>  
  
using namespace std;  
  
int main() {  
 *// Seed the random number generator*  
 srand(time(0));  
  
 *// Generate a random number between 1 and 100*  
 int secretNumber = (rand() % 100) + 1;  
  
 *// Set the maximum number of attempts*  
 int maxAttempts = 7;  
  
 *// Initialize variables*  
 int guess;  
 int attempts = 0;  
 bool guessed = false;  
  
 cout << "Welcome to the Number Guessing Game!" << endl;  
 cout << "I'm thinking of a number between 1 and 100." << endl;  
 cout << "Try to guess it within " << maxAttempts << " attempts." << endl;  
  
 while (attempts < maxAttempts) {  
 cout << "Attempt " << attempts + 1 << ": Enter your guess: ";  
 cin >> guess;  
  
 attempts++;  
  
 if (guess < secretNumber) {  
 cout << "Too low! Try again." << endl;  
 } else if (guess > secretNumber) {  
 cout << "Too high! Try again." << endl;  
 } else {  
 cout << "Congratulations! You guessed the number in " << attempts << " attempts!" << endl;  
 guessed = true;  
 break;  
 }  
 }  
  
 if (!guessed) {  
 cout << "You ran out of attempts. The number was " << secretNumber << "." << endl;  
 }  
  
 return 0;  
}

Explanation:

1. **1. Include headers:**
   * iostream for input/output operations.
   * cstdlib for rand() function (random number generation).
   * ctime for time() function (used for seeding the random number generator).
2. **2. Seed the random number generator:**
   * srand(time(0)); initializes the random number generator using the current time. This ensures that the game generates different numbers each time you run it.
3. **3. Generate a random number:**
   * int secretNumber = (rand() % 100) + 1; generates a random integer between 1 and 100 (inclusive).
4. **4. Set maximum attempts:**
   * int maxAttempts = 7; sets the maximum number of guesses the user can make.
5. **5. Initialize variables:**
   * int guess; will store the user's guess.
   * int attempts; keeps track of the number of attempts.
   * bool guessed; is a flag to indicate if the user guessed the number correctly.
6. **6. Game loop:**
   * while (attempts < maxAttempts): This loop continues until the user has made the maximum number of attempts.
   * Inside the loop:
     + cout << "Attempt " << attempts + 1 << ": Enter your guess: "; prompts the user for their guess.
     + cin >> guess; reads the user's input.
     + attempts++; increments the attempt counter.
     + **Hints:**
       - if (guess < secretNumber): If the guess is too low, a "Too low!" message is displayed.
       - else if (guess > secretNumber): If the guess is too high, a "Too high!" message is displayed.
       - else: If the guess is correct, a congratulatory message is displayed, the guessed flag is set to true, and the loop breaks.
7. **7. Game over message:**
   * if (!guessed): If the user runs out of attempts and the guessed flag is still false, a message is displayed telling the user the secret number.
8. **8. Return 0:**
   * return 0; indicates that the program executed successfully.

}