**Name :- Pal Gajera**

**Module 2 – Introduction to Programming**

**THEORY**

**Overview of C Programming**

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

ANS:-

Dennis Ritchie created C at Bell Laboratories in the early 1970s. It developed from an older language named B that Ken Thompson created. The main purpose of C's creation was to construct the Unix operating system, which was crucial in the advancement of contemporary computers.

**Setting Up Environment**

**Q2:- 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:-

**1. Installing GCC Compiler**

* **For Windows:**
  + MinGW: Download and install MinGW from [MinGW website](https://osdn.net/projects/mingw/releases/). Add its bin folder (e.g., C:\MinGW\bin) to the system PATH.
  + Check Installation: Open Command Prompt and type gcc --version to confirm GCC is installed.
* **For macOS:**
  + Install Xcode command line tools by running xcode-select --install in Terminal.
* **For Linux:**
  + Run sudo apt install build-essential to install GCC on Ubuntu/Debian.

**2. Setting Up an IDE**

* **DevC++ (Windows):**
  + Download from [DevC++ website](https://sourceforge.net/projects/orwelldevcpp/), and it comes with MinGW preinstalled.
  + Create a new C project, write your code, and press Compile & Run to execute.
* **VS Code (Cross-platform):**
  + Download from [VS Code website](https://code.visualstudio.com/), and install the C/C++ extension.
  + Install GCC and configure tasks.json to build C code. Use Ctrl+Shift+B to compile and run your program.
* **CodeBlocks (Cross-platform):**
  + Download from [CodeBlocks website](http://www.codeblocks.org/downloads/26) (choose version with MinGW for Windows).
  + Create a new C project, write your code, and click Build and Run.

**3. Test Your Setup**

* Write a simple C program:
* #include <stdio.h>
* int main() {
* printf("Hello, World!\n");
* return 0;
* }
* Run it in your chosen IDE. If it prints "Hello, World!", your setup is successful.

**Basic Structure of a C Program**

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

**ANS:-**

**1. Headers**

* Purpose: Headers include libraries that provide useful functions. Common headers are included at the top of the program using the #include directive.
* Example: #include <stdio.h> is used to include the Standard Input/Output library for functions like printf.

**2. The main Function**

* Purpose: The main function is the starting point of every C program. Execution starts here.
* Syntax:
* int main() {
* // Code
* return 0; // End of the program
* }

**3. Comments**

* Purpose: Comments are used to explain the code. They are ignored by the compiler.
  + Single-line comments: // This is a comment
  + Multi-line comments: /\* This is a multi-line comment \*/

**4. Data Types**

* Purpose: Data types define the kind of data variables can store.
  + int: for integers (whole numbers).
  + float: for decimal numbers.
  + char: for single characters.
* Example:
* int age = 25; // Integer
* float salary = 5000.50; // Decimal number
* char grade = 'A'; // Character

**5. Variables**

* Purpose: Variables store data. They need to be declared with a data type before use.
* Example:
* int x = 10; // Integer variable
* float y = 20.5; // Float variable
* char letter = 'A'; // Character variable

**Example C Program:**

#include <stdio.h> // Include standard I/O library

int main() {

int age = 20; // Declare an integer variable

float salary = 3000.75; // Declare a float variable

char grade = 'A'; // Declare a char variable

// Print the values

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

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

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

return 0; // End of the program

}

**Explanation:**

* #include <stdio.h>: Includes the library needed for input and output functions.
* int main(): The main function where the program starts.
* int, float, char: Data types used to declare variables.
* printf: Used to display output to the console.

This is the basic structure of a C program, which includes headers, the main function, comments, data types, and variables.

**Operators in C**

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

**ANS:-**

**1. Arithmetic Operators**

These operators perform basic arithmetic operations.

* +: Addition
* -: Subtraction
* \*: Multiplication
* /: Division
* %: Modulo (remainder of division)

**2. Relational Operators**

These operators are used to compare two values or expressions.

* ==: Equal to
* !=: Not equal to
* >: Greater than
* <: Less than
* >=: Greater than or equal to
* <=: Less than or equal to

**3. Logical Operators**

These operators are used to combine or invert conditions.

* &&: Logical AND (both conditions must be true)
* ||: Logical OR (at least one condition must be true)
* !: Logical NOT (inverts the condition)

**4. Assignment Operators**

These operators are used to assign values to variables.

* =: Simple assignment
* +=: Add and assign
* -=: Subtract and assign
* \*=: Multiply and assign
* /=: Divide and assign
* %=: Modulo and assign

**5. Increment/Decrement Operators**

These operators increase or decrease the value of a variable by 1.

* ++: Increment (increases by 1)
* --: Decrement (decreases by 1)

**6. Bitwise Operators**

These operators perform operations on bits (binary data).

* &: Bitwise AND
* |: Bitwise OR
* ^: Bitwise XOR (exclusive OR)
* ~: Bitwise NOT (inverts all bits)
* <<: Left shift
* >>: Right shift

**7. Conditional (Ternary) Operator**

This operator is used as a shorthand for if-else conditions.

* ?:: Conditional (ternary) operator
  + Syntax: condition ? expr1 : expr2
  + If condition is true, expr1 is executed; otherwise, expr2 is executed.

**Control Flow Statements in C**

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

**ANS**

**1. if Statement**

The if statement checks if a condition is true. If the condition evaluates to true, the code inside the if block is executed. If it is false, the program continues after the if block.

**2. else Statement**

The else statement follows an if and provides an alternative block of code to execute when the if condition is false. It ensures that one of the blocks of code is always executed—either the if block or the else block.

**3. Nested if-else Statement**

A nested if-else statement occurs when one if or else is inside another if or else. This allows you to check multiple conditions in a hierarchical manner. Each condition is checked based on whether the previous ones were true or false.

**4. switch Statement**

The switch statement allows you to test a variable against a series of values (called cases). It is used when there are multiple potential values to compare. Each case corresponds to a possible value of the variable, and the code for the matching case is executed. If no match is found, the default block is executed (if provided).

**Looping in C**

**Q6:- Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.**

**ANS**

**1. while Loop**

* Condition Checked: The condition is evaluated before entering the loop.
* Execution: The loop executes only if the condition is true at the start. If the condition is false initially, the loop does not execute at all.
* Use Case: Ideal when you don’t know how many times the loop will run in advance, and you want to repeat the block of code as long as a condition is true.

**2. for Loop**

* Condition Checked: The condition is checked before each iteration, just like the while loop.
* Execution: The for loop is typically used when the number of iterations is known ahead of time. It usually includes initialization, condition checking, and update in a single line.
* Use Case: Best when you know exactly how many times you need to loop, such as iterating over a range or array.

**3. do-while Loop**

* Condition Checked: The condition is evaluated after the loop executes, meaning the loop will run at least once.
* Execution: The code inside the loop executes at least once, and then the condition is checked for subsequent iterations.
* Use Case: Useful when you need the loop to execute at least once, such as when prompting the user for input or performing a task at least once before checking the condition.

**Loop Control Statements**

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

**ANS**

**1. break Statement**

* **Purpose**: The break statement is used to **exit** a loop or a switch statement prematurely, before the loop or switch condition is false.
* **Use**: It is typically used when you want to stop a loop based on some condition.

**Example**:

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

if (i == 3) {

break; // Exit the loop when i equals 3

}

printf("%d ", i);

}

// Output: 0 1 2

**2. continue Statement**

* **Purpose**: The continue statement is used to **skip the current iteration** of a loop and move to the next iteration, bypassing the remaining code in the loop body for that iteration.
* **Use**: It's useful when you want to skip certain conditions without exiting the loop entirely.

**Example**:

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

if (i == 3) {

continue; // Skip the current iteration when i equals 3

}

printf("%d ", i);

}

// Output: 0 1 2 4

**3. goto Statement**

* **Purpose**: The goto statement is used to **jump to a specific point** in the code (a label), bypassing the normal flow of execution.
* **Use**: It's generally avoided in most cases because it makes code harder to follow, but it can be used for breaking out of deeply nested loops or for error handling.

**Example**:

int i = 0;

start:

if (i == 3) {

goto end; // Jump to the 'end' label

}

printf("%d ", i);

i++;

goto start; // Jump back to the 'start' label

end:

printf("\nDone\n");

// Output: 0 1 2 Done

**Functions in C**

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

**ANS**

**1. Function Declaration**

* **Purpose**: The function declaration (or **prototype**) tells the compiler about the function's name, return type, and parameters. It is usually written before the main function.
* **Syntax**:
* return\_type function\_name(parameter1\_type, parameter2\_type, ...);

**2. Function Definition**

* **Purpose**: The function definition provides the actual implementation (or body) of the function, where the specific task is carried out.
* **Syntax**:
* return\_type function\_name(parameter1\_type, parameter2\_type, ...) {
* // Code to execute
* return value; // Optional, based on return type
* }

**3. Calling a Function**

* **Purpose**: A function is called in the main function (or anywhere in the program) by using its name and passing the required arguments.
* **Syntax**:
* function\_name(argument1, argument2, ...);

**Arrays in C**

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

**ANS**

**1. One-Dimensional Array**

A **one-dimensional array** is a list of elements, all of the same data type, accessed using a single index.

* **Syntax**:
* data\_type array\_name[size];
* **Example**:
* int numbers[5] = {1, 2, 3, 4, 5};
* **Explanation**:
  + numbers is a one-dimensional array of integers, with 5 elements.
  + You can access an element using its index, for example, numbers[0] will give 1.

**2. Multi-Dimensional Array**

A **multi-dimensional array** is an array of arrays. The most common type is a two-dimensional array, which can be thought of as a table with rows and columns.

* **Syntax**:
* data\_type array\_name[rows][columns];
* **Example** (2D array):
* int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
* **Explanation**:
  + matrix is a 2D array with 2 rows and 3 columns.
  + You can access an element using two indices, for example, matrix[1][2] will give 6.

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

|  |  |  |
| --- | --- | --- |
| **Feature** | **One-Dimensional Array** | **Multi-Dimensional Array** |
| **Structure** | A simple list of elements | A list of lists (rows and columns) |
| **Indexing** | Uses one index | Uses multiple indices (rows, columns) |
| **Example** | int numbers[5] = {1, 2, 3, 4, 5}; | int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}}; |
| **Use Case** | Storing a single series of data | Storing tabular or grid-like data |

**Pointers in C**

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

**Declaring and Initializing Pointers**

* **Declaration**: To declare a pointer, you specify the type of variable it will point to, followed by an asterisk (\*).
* data\_type \*pointer\_name;
* **Initialization**: A pointer is initialized by assigning it the address of another variable using the **address-of operator** (&).
* pointer\_name = &variable\_name;

**Why Are Pointers Important in C?**

* **Memory Efficiency**: Pointers allow direct access to memory, making programs more efficient, especially when dealing with large data structures or arrays.
* **Dynamic Memory Allocation**: Pointers are essential for dynamic memory allocation (using malloc, calloc, etc.).
* **Function Arguments**: They allow passing large structures or arrays to functions efficiently by passing the memory address instead of copying the entire data.
* **Manipulating Data**: Pointers enable more complex data manipulation, such as modifying variables in different functions or working with arrays.

**Strings in C**

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

**ANS**

**1. strlen()**

* **Purpose**: Returns the length of a string (number of characters excluding the null-terminating character \0).
* **Syntax**: size\_t strlen(const char \*str);
* **Use Case**: Useful when you need to know the length of a string.
* **Example**:
* char str[] = "Hello";
* int length = strlen(str); // length will be 5

**2. strcpy()**

* **Purpose**: Copies one string into another.
* **Syntax**: char \*strcpy(char \*dest, const char \*src);
* **Use Case**: Useful when you want to copy the content of one string to another.
* **Example**:
* char src[] = "World";
* char dest[10];
* strcpy(dest, src); // dest will now be "World"

**3. strcat()**

* **Purpose**: Concatenates (joins) two strings. The second string is added at the end of the first string.
* **Syntax**: char \*strcat(char \*dest, const char \*src);
* **Use Case**: Useful when you want to combine two strings into one.
* **Example**:
* char str1[20] = "Hello";
* char str2[] = " World";
* strcat(str1, str2); // str1 will be "Hello World"

**4. strcmp()**

* **Purpose**: Compares two strings lexicographically (character by character). Returns 0 if they are equal, a negative value if the first string is smaller, or a positive value if the first string is larger.
* **Syntax**: int strcmp(const char \*str1, const char \*str2);
* **Use Case**: Useful when you want to compare two strings (e.g., checking for equality).
* **Example**:
* char str1[] = "Apple";
* char str2[] = "Banana";
* int result = strcmp(str1, str2); // result will be negative (Apple < Banana)

**5. strchr()**

* **Purpose**: Finds the first occurrence of a character in a string.
* **Syntax**: char \*strchr(const char \*str, int ch);
* **Use Case**: Useful when you need to find a character in a string (e.g., checking for a specific character).
* **Example**:
* char str[] = "Hello";
* char \*ptr = strchr(str, 'e'); // ptr will point to the character 'e' in the string

**Structures in C**

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

**ANS**

**1. Declaring a Structure**

To declare a structure, use the struct keyword followed by the structure name and its members enclosed in curly braces.

* **Syntax**:
* struct structure\_name {
* data\_type member1;
* data\_type member2;
* // More members
* };
* **Example**:
* struct Student {
* int rollNo;
* char name[50];
* float marks;
* };

**2. Initializing a Structure**

You can initialize a structure in two ways:

1. **At the time of declaration**.
2. **Using individual assignments**.

* **Syntax for direct initialization**:
* struct structure\_name variable\_name = {value1, value2, ...};
* **Example**:
* struct Student student1 = {1, "John", 85.5};
* **Using assignments**:
* struct Student student1;
* student1.rollNo = 1;
* strcpy(student1.name, "John");
* student1.marks = 85.5;

**3. Accessing Structure Members**

You can access structure members using the **dot operator (.)** for regular variables or **arrow operator (->)** for pointers to structures.

* **Example (dot operator)**:
* printf("Roll Number: %d\n", student1.rollNo);
* printf("Name: %s\n", student1.name);
* printf("Marks: %.2f\n", student1.marks);
* **Example (arrow operator)** (for pointers):
* struct Student \*ptr = &student1;
* printf("Roll Number: %d\n", ptr->rollNo);

**File Handling in C**

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

**ANS**

**Importance of File Handling in C**

File handling in C allows programs to **read from** and **write to** files, making it possible to store data permanently (instead of just in memory). This is useful for tasks such as saving user input, reading data, or logging information. Without file handling, data would be lost once the program ends.

**File Operations in C**

There are several essential file operations, such as opening, closing, reading, and writing files. These operations are performed using functions from the C standard library.

**1. Opening a File**

To open a file, use the fopen() function, which requires the file name and the mode (read, write, etc.).

* **Syntax**:
* FILE \*fopen(const char \*filename, const char \*mode);
* **Modes**:
  + "r": Read mode (opens file for reading).
  + "w": Write mode (creates an empty file for writing).
  + "a": Append mode (opens file for writing, adding content at the end).
  + "r+": Read and write mode.
* **Example**:
* FILE \*file = fopen("data.txt", "w");

**2. Writing to a File**

You can write data to a file using functions like fprintf() (for formatted text) or fputs() (for plain text).

* **Syntax**:
* int fprintf(FILE \*stream, const char \*format, ...);
  + Writes formatted output to a file.
* **Example**:
* fprintf(file, "Hello, World!\n");

**3. Reading from a File**

To read data from a file, use functions like fscanf() (for formatted input) or fgets() (to read a line).

* **Syntax**:
* int fscanf(FILE \*stream, const char \*format, ...);
  + Reads formatted input from a file.
* **Example**:
* char line[100];
* fgets(line, sizeof(line), file); // Reads a line of text

**4. Closing a File**

Once you're done with a file, use the fclose() function to close it and free up system resources.

* **Syntax**:
* int fclose(FILE \*stream);
* **Example**:
* fclose(file);

**LAB**

**Q1:- 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 fundamental in several areas due to its efficiency and close-to-hardware capabilities. Here are three key domains where C is extensively used:

1. **Embedded Systems**: C is the go-to language for programming embedded systems, which are specialized, resource-constrained devices. These systems control everything from household appliances to automotive electronics. C’s ability to directly manipulate hardware and manage memory efficiently makes it ideal for environments with limited computing resources, such as microcontrollers and IoT devices.
2. **Operating Systems**: C plays a central role in the development of operating systems. Many well-known operating systems, such as Linux and Unix, are built primarily in C. It provides the low-level access needed to interact with hardware components, manage memory, and create efficient system processes. C's ability to optimize performance and its portability across different architectures make it indispensable for OS development.
3. **Game Development**: In high-performance game engines and real-time simulations, C is used for low-level processing and optimizing performance. While modern game engines often use C++ or other languages for game logic, C is still used in the core engine for handling tasks like graphics rendering, physics calculations, and real-time interactions. Classic games like *Doom* and *Quake* were built using C due to its ability to provide maximum control over system resources.

In all these fields, C's low-level capabilities, memory management, and performance optimization make it essential for tasks that require efficiency and direct hardware interaction.

**Q2:- Write your first program to print "Hello, World!" and run it.**

**ANS**

**CODE:-**

#include<stdio.h>

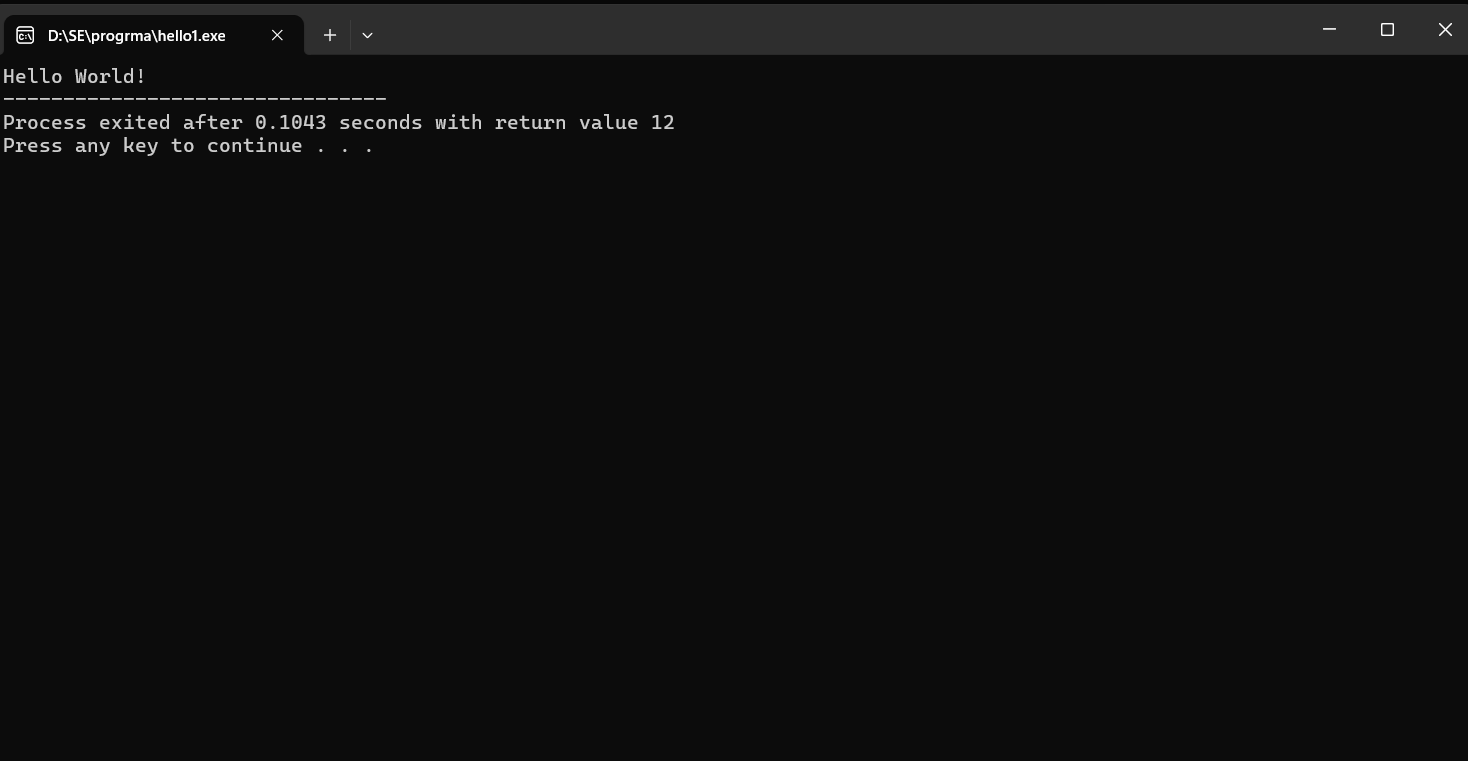
main()

{

printf("Hello World!");

}

**OUTPUT:-**



**Q3:- Write a C program that includes variables, constants, and comments.**

**ANS**

**CODE:-**

#include <stdio.h>

main() {

int age = 25;

float height = 5.9;

const float PI = 3.14159;

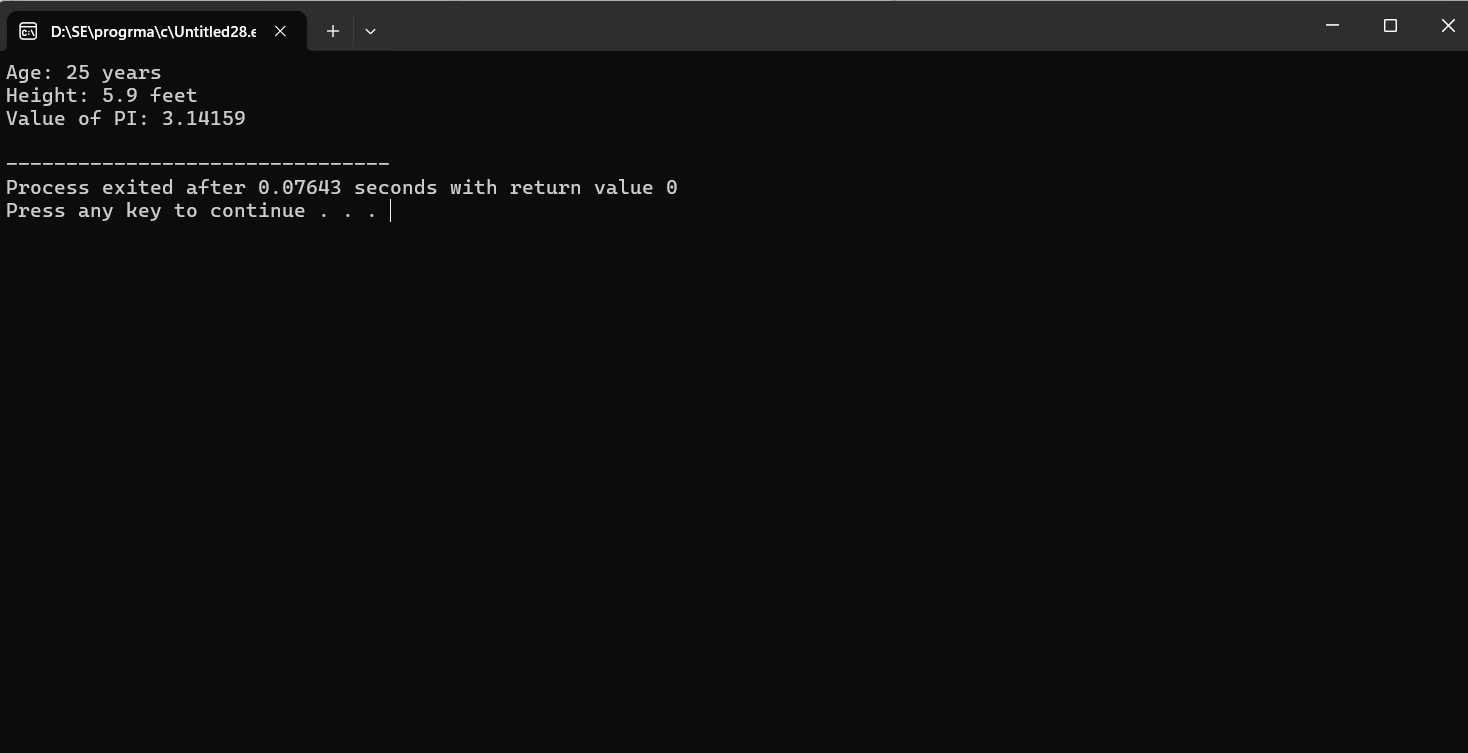
printf("Age: %d years\n", age);

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

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

}

**OUTPUT:-**



**Q4:- Declare and use different data types (int, char, float) and display their values.**

**ANS**

**CODE:-**

#include<stdio.h>

main(){

char name[10];

int dob;

float age;

printf("Enter your name ");

scanf("%s",&name);

printf("Enter your DOB ");

scanf("%d",&dob);

printf("Enter your age ");

scanf("%f",&age);

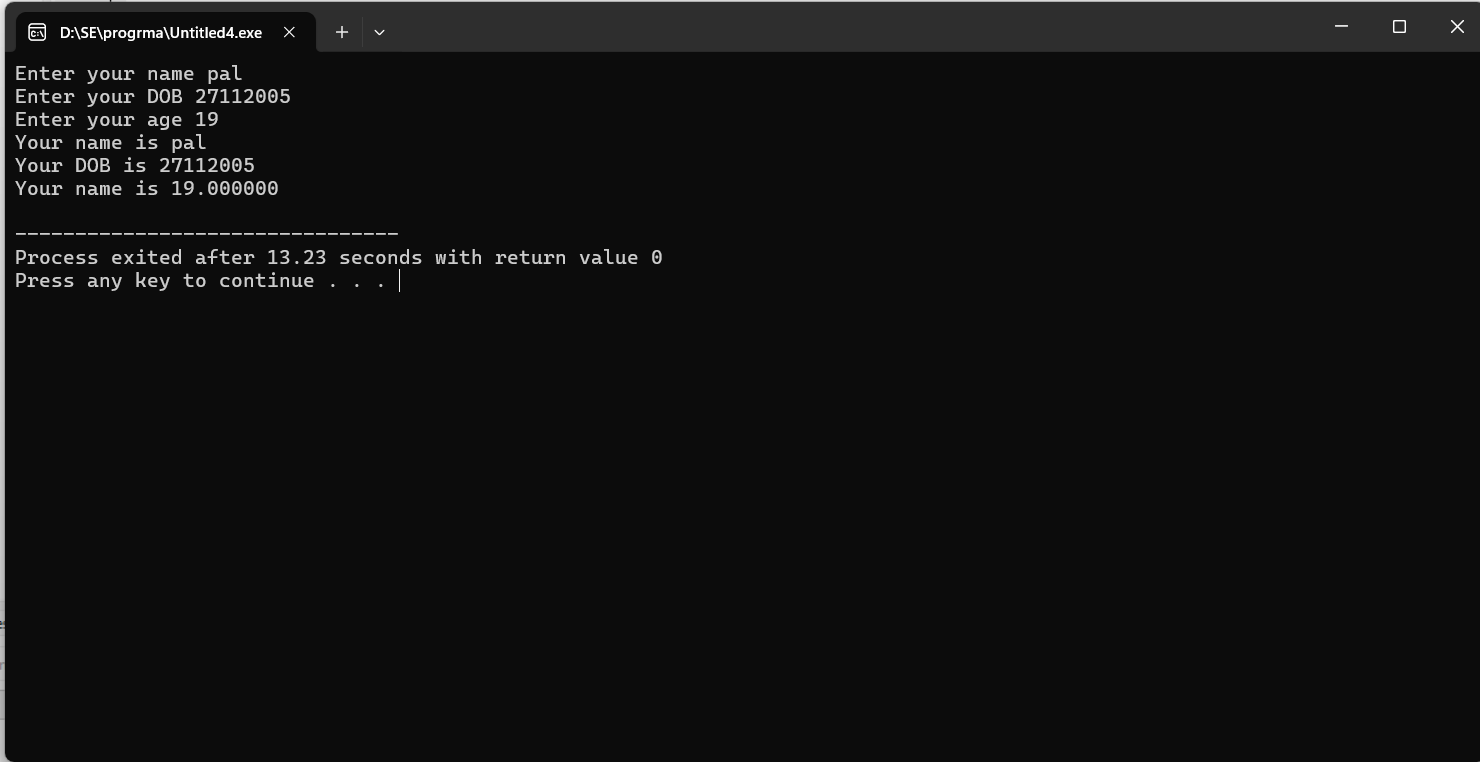
printf("Your name is %s\n",name);

printf("Your DOB is %d\n",dob);

printf("Your name is %f\n",age);

}

**OUTPUT:-**



**Q5:- Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results.**

**ANS**

**CODE:-**

#include <stdio.h>

int main() {

int num1, num2;

// Input two integers

printf("Enter first integer: ");

scanf("%d", &num1);

printf("Enter second integer: ");

scanf("%d", &num2);

// Arithmetic operations

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

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

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

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

// Relational operations

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

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

// Logical operations

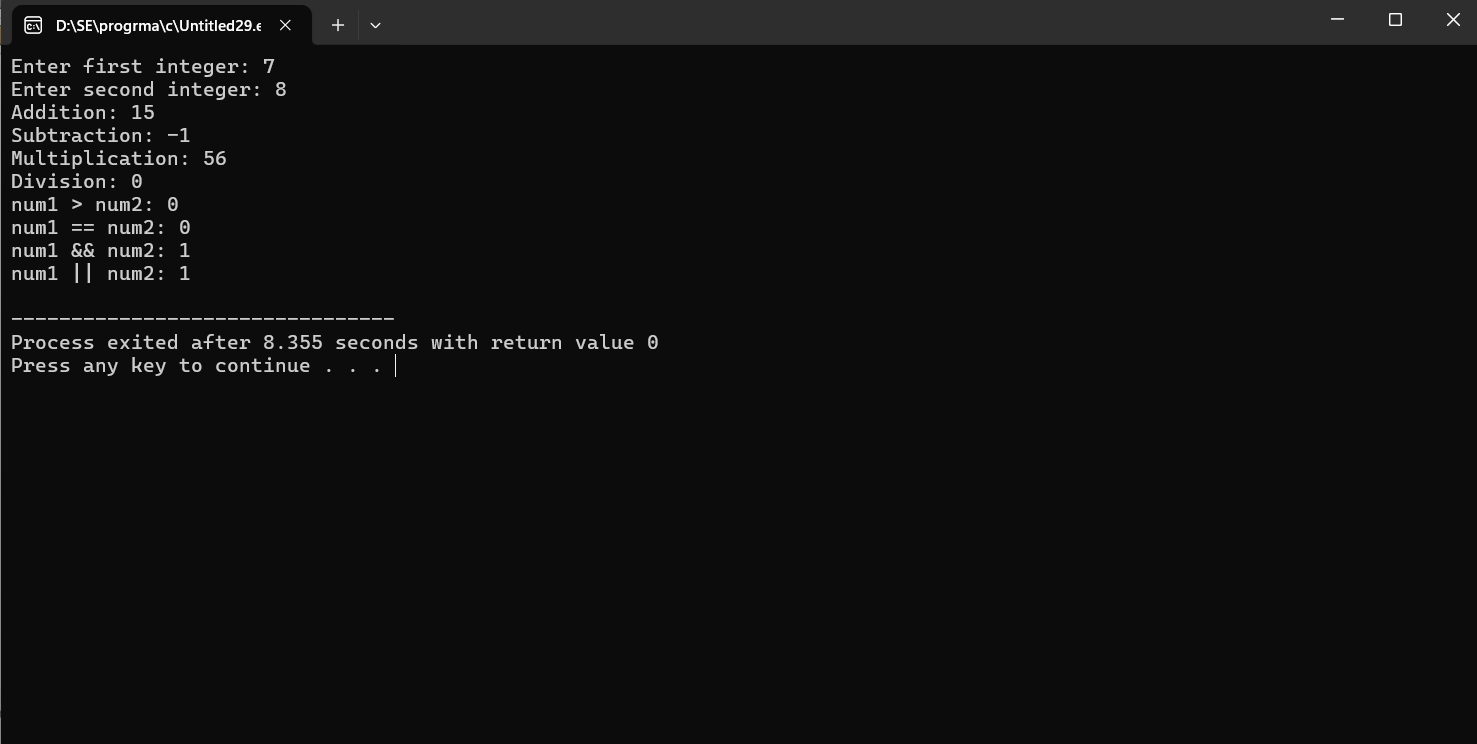
printf("num1 && num2: %d\n", num1 && num2);

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

return 0;

}

**OUTPUT:-**

****

**Q6:- Write a C program to check if a number is even or odd using an if-else statement.**

**ANS**

**CODE:-**

#include<stdio.h>

main(){

int num1;

printf("Enter number 1: ");

scanf("%d",&num1);

if(num1%2==0)

{

printf("This number is even");

}

else

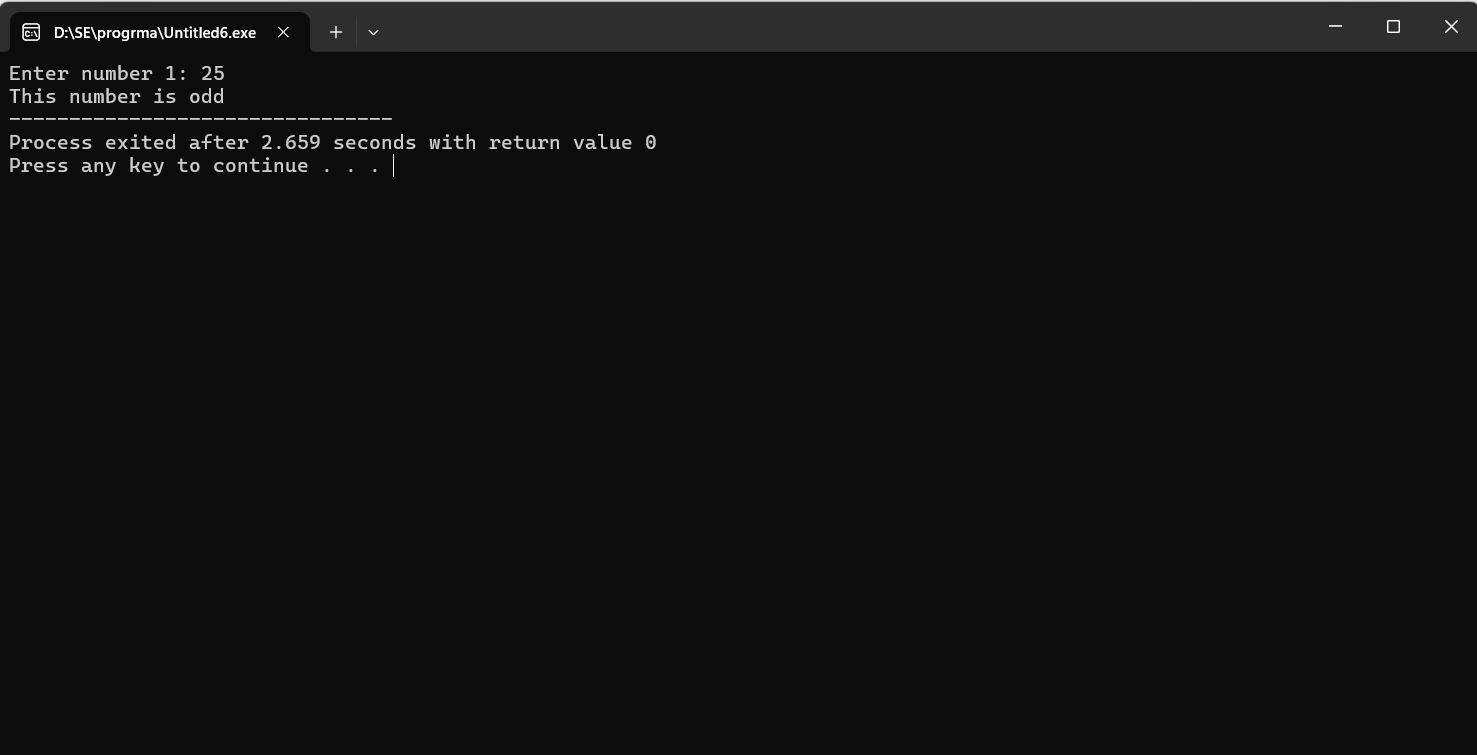
{

printf("This number is odd");

}

}

**OUTPUT:-**



**Q7:- Extend the program using a switch statement to display the month name based on the user’s input (1 for January, 2 for February, etc.).**

**ANS**

**CODE:-**

#include<stdio.h>

main(){

int choice;

printf("Enter your choice number between 1 to 12: ");

scanf("%d",&choice);

switch(choice){

case 1:

printf("month is jan");

break;

case 2:

printf("month is feb");

break;

case 3:

printf("month is mar");

break;

case 4:

printf("month is apr");

break;

case 5:

printf("month is may");

break;

case 6:

printf("month is jun");

break;

case 7:

printf("month is jul");

break;

case 8:

printf("month is aug");

break;

case 9:

printf("month is sept");

break;

case 10:

printf("month is oct");

break;

case 11:

printf("month is nov");

break;

case 12:

printf("month is dec");

break;

default:

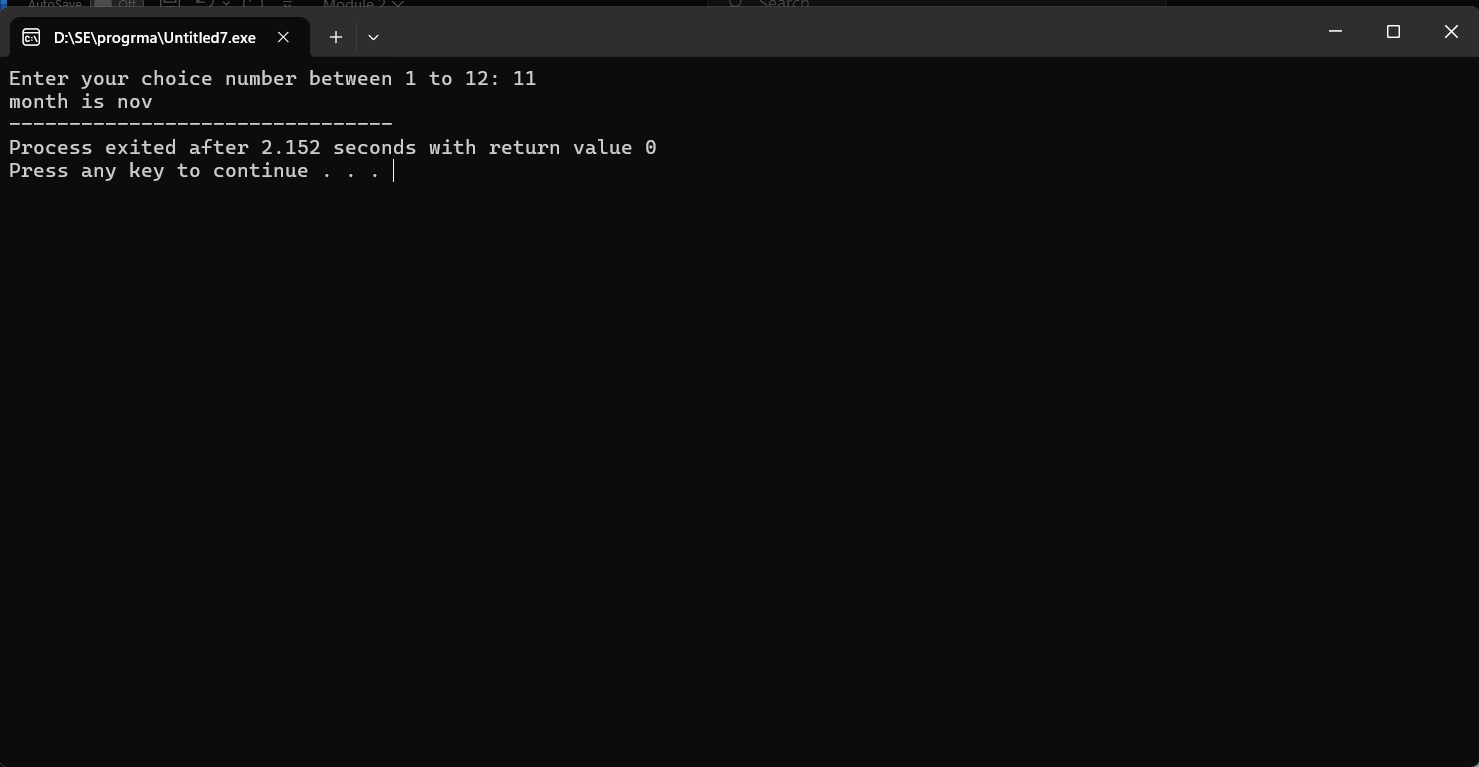
printf("Not a valid number.....");

break;

}

}

**OUTPUT:-**



**Q8:- Write a C program to print numbers from 1 to 10 using all three types of loops (while, for, do-while).**

**ANS**

**CODE:-**

#include <stdio.h>

main() {

int i;

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

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

printf("%d ", i);

}

printf("\n");

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

i = 1;

while (i <= 10) {

printf("%d ", i);

i++;

}

printf("\n");

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

i = 1;

do {

printf("%d ", i);

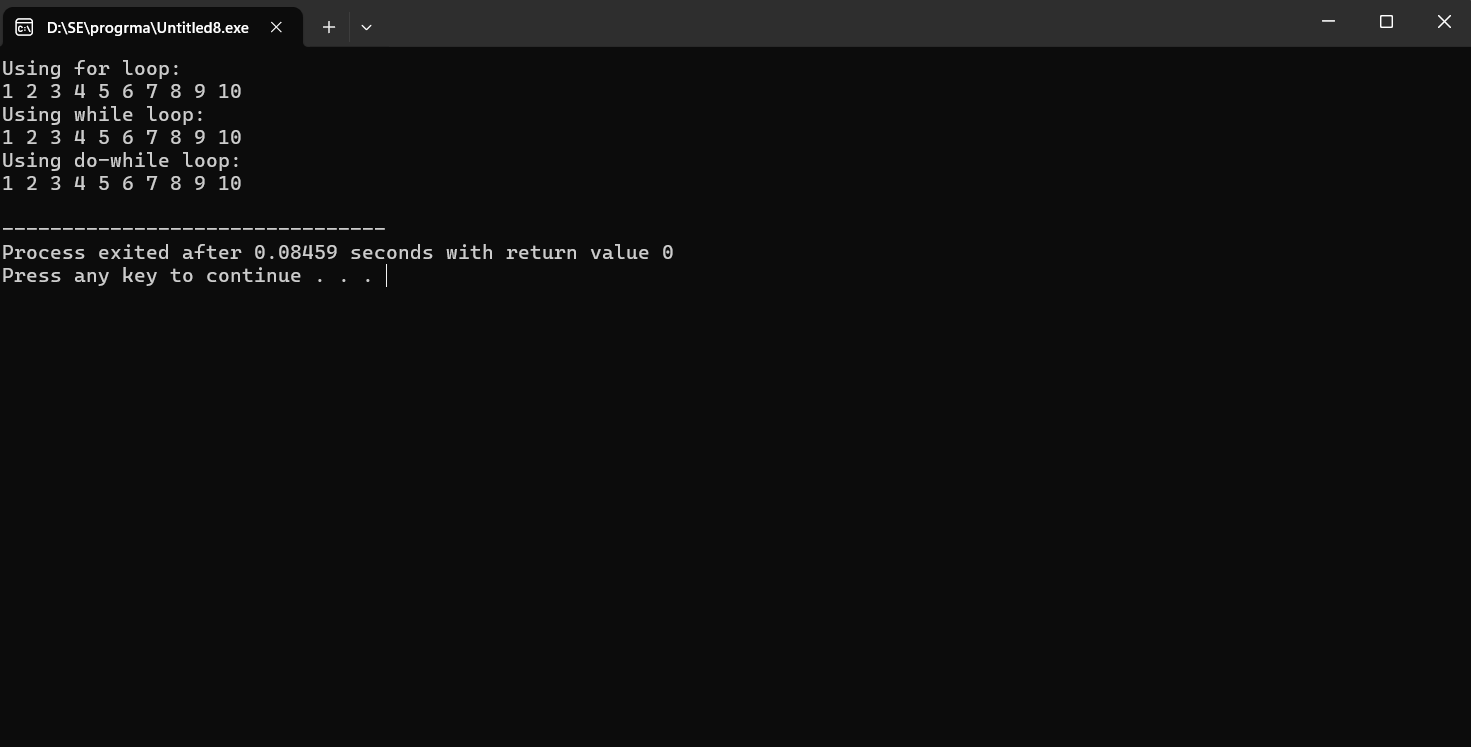
i++;

} while (i <= 10);

printf("\n");

}

**OUTPUT:-**



**Q9:- Write a C program that uses the break statement to stop printing numbers when it reaches 5?**

**ANS**

**CODE:-**

#include<stdio.h>

main(){

int i;

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

if(i==5){

break;

}

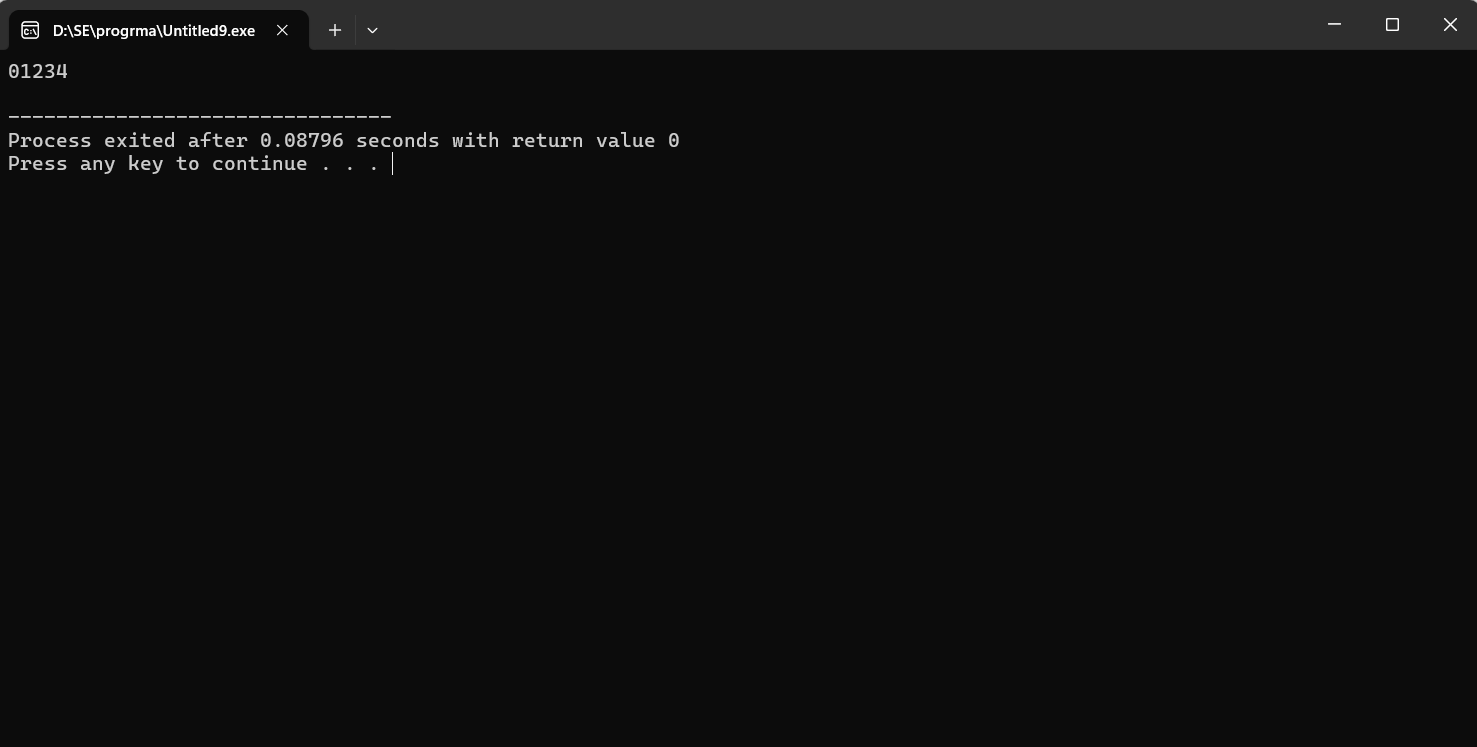
printf("%d",i);

}

printf("\n");

}

**OUTPUT:-**



**Q10:- Modify the program to skip printing the number 3 using the continue statement.**

**ANS**

**CODE:-**

#include<stdio.h>

main(){

int i;

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

if(i==3){

continue;

}

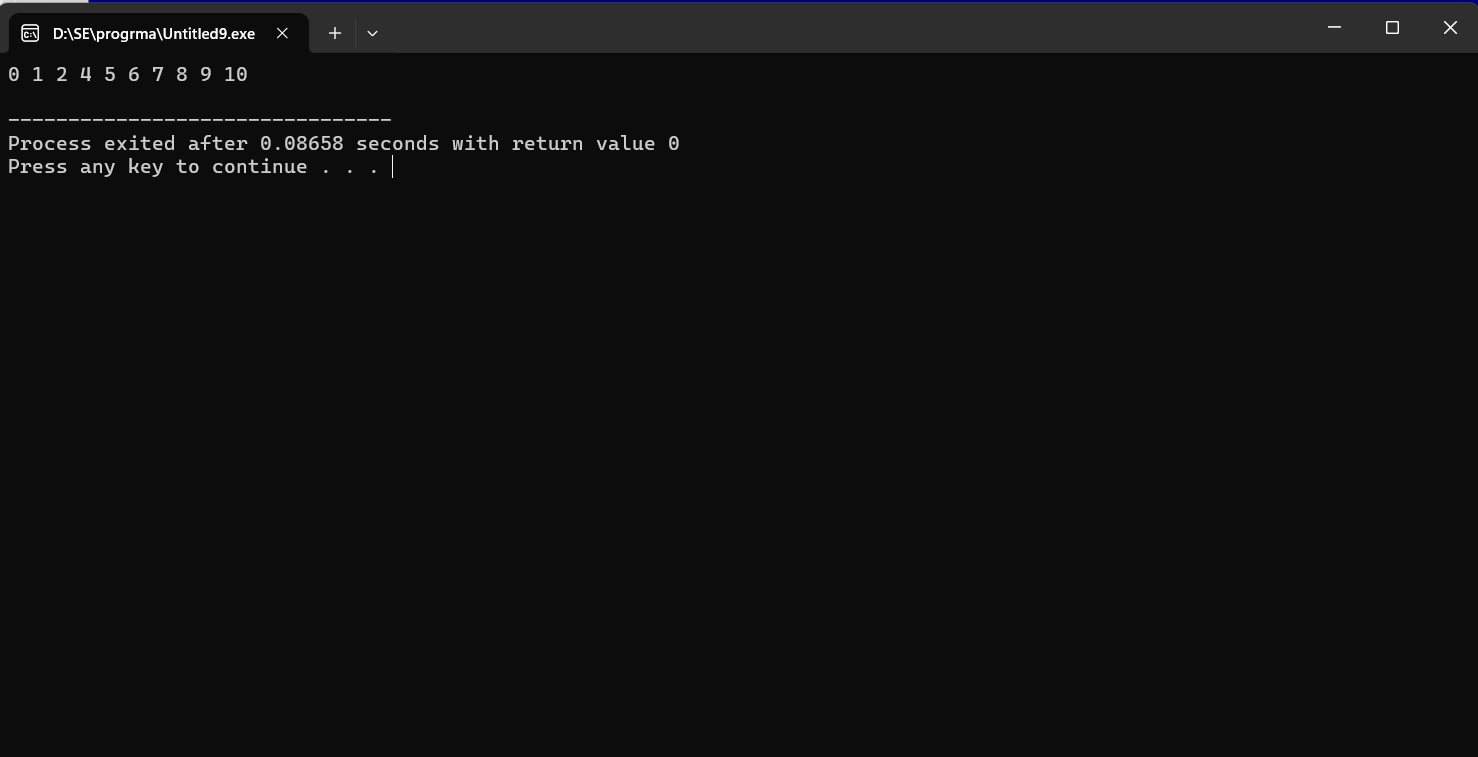
printf("%d ", i);

}

printf("\n");

}

**OUTPUT:-**



**Q11:- Write a C program that calculates the factorial of a number using a function. Include function declaration, definition, and call.**

**ANS**

**CODE:-**

#include <stdio.h>

main() {

int num, fact = 1;

printf("Enter a number: ");

scanf("%d", &num);

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

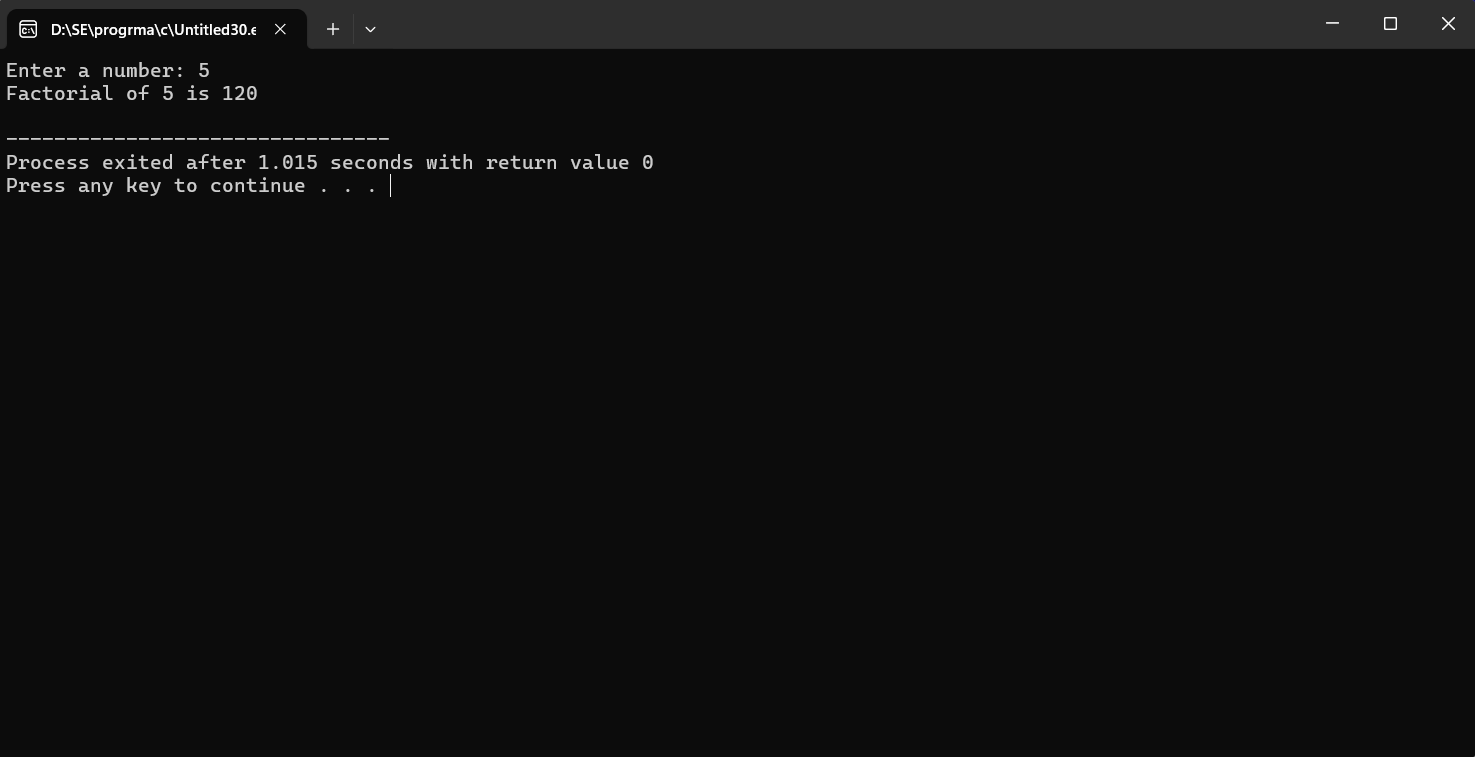
fact \*= i;

}

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

}

**OUTPUT:-**



**Q12:- Write a C program that stores 5 integers in a one-dimensional array and prints them.**

**ANS**

**CODE:-**

#include<stdio.h>

main(){

int arr[5];

printf("Enter 5 integers in a one-dimensional array:\n\n");

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

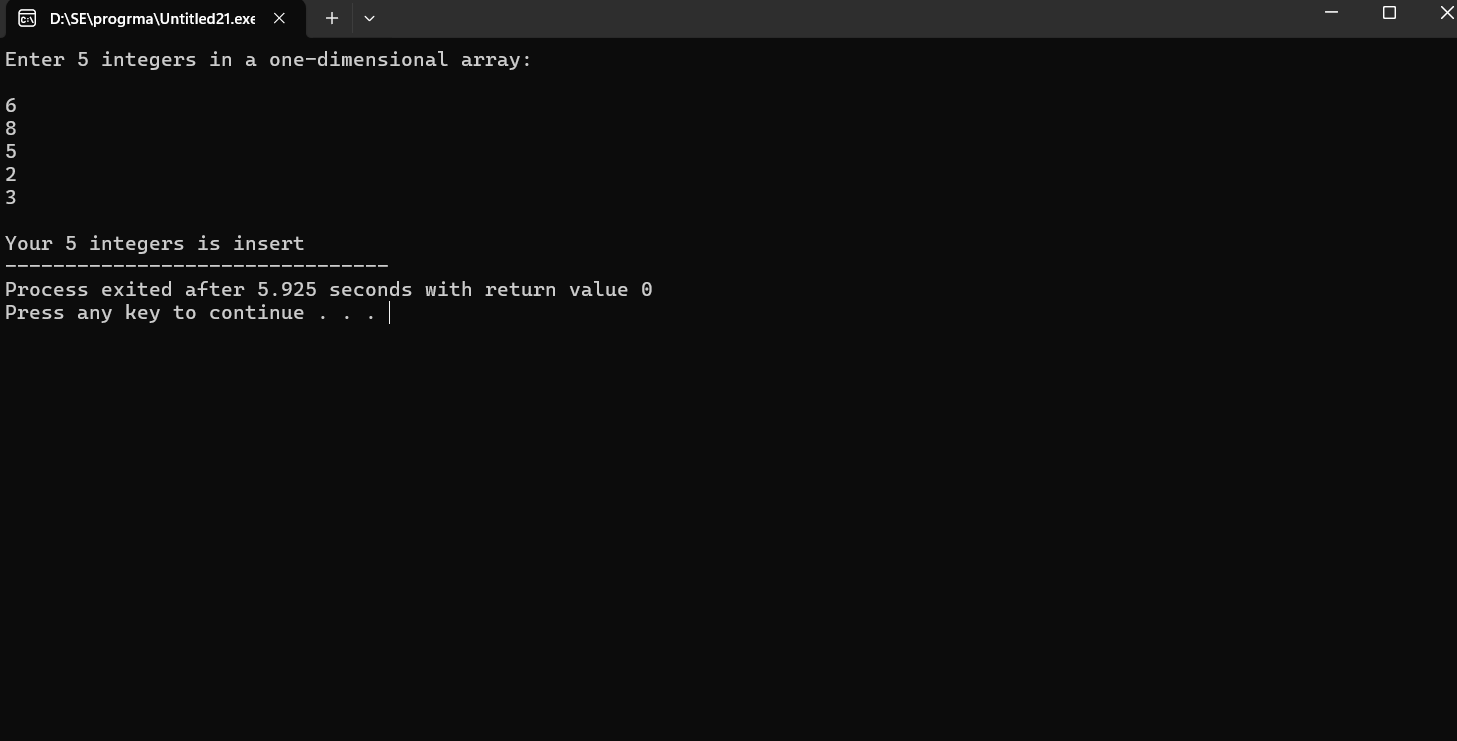
scanf("%d",&arr[i]);

}

printf("\nYour 5 integers is insert");

}

**OUTPUT:-**



**Q13:- Extend this to handle a two-dimensional array (3x3 matrix) and calculate the sum of all elements.**

**ANS**

**CODE:-**

#include <stdio.h>

main() {

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

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

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

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

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

}

}

printf("\nTwo-dimensional array (3x3 matrix) elements:\n");

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

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

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

}

printf("\n");

}

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

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

sum += matrix[i][j];

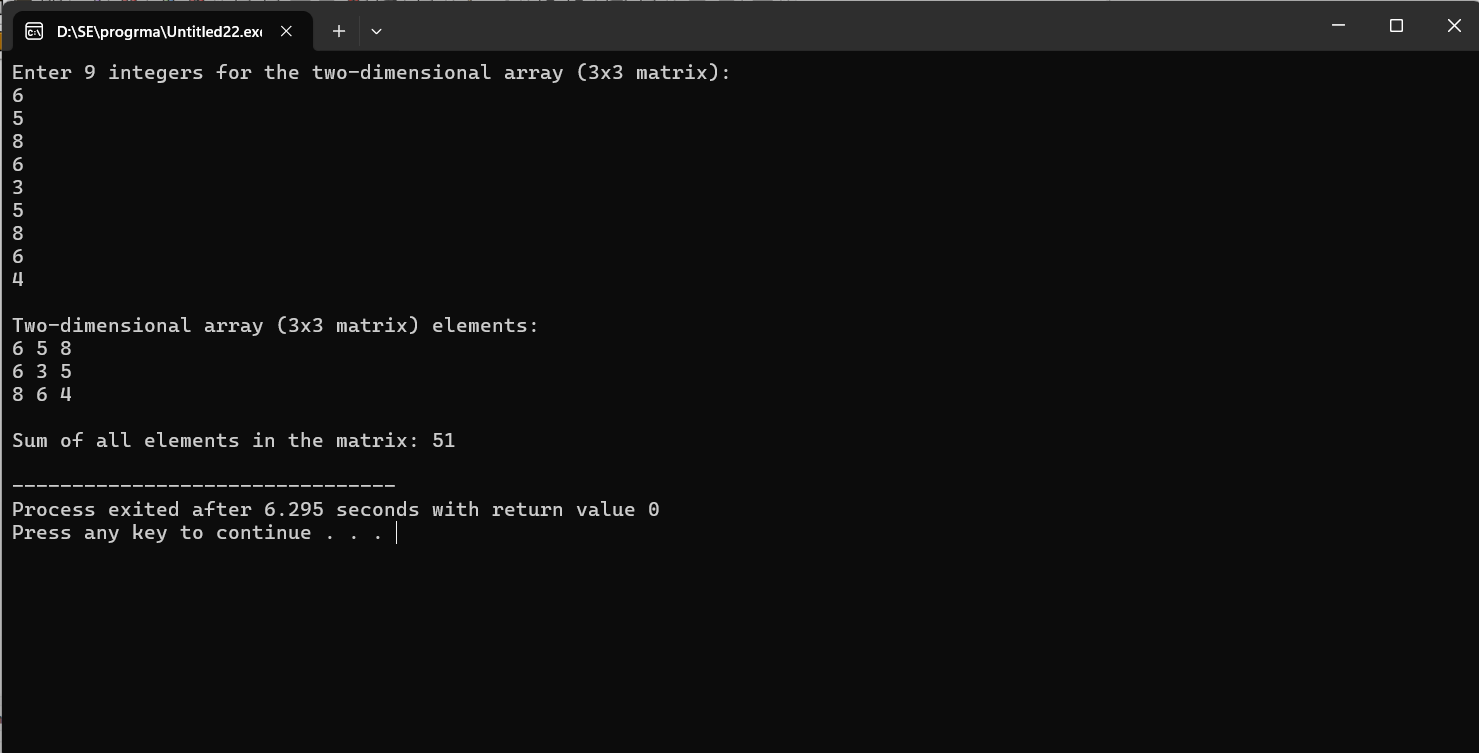
}

}

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

}

**OUTPUT:-**



**Q14:- Write a C program to demonstrate pointer usage. Use a pointer to modify the value of a variable and print the result.**

**ANS**

**CODE:-**

#include <stdio.h>

main() {

int num, \*ptr;

printf("Enter an integer value for num: ");

scanf("%d", &num);

ptr = &num;

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

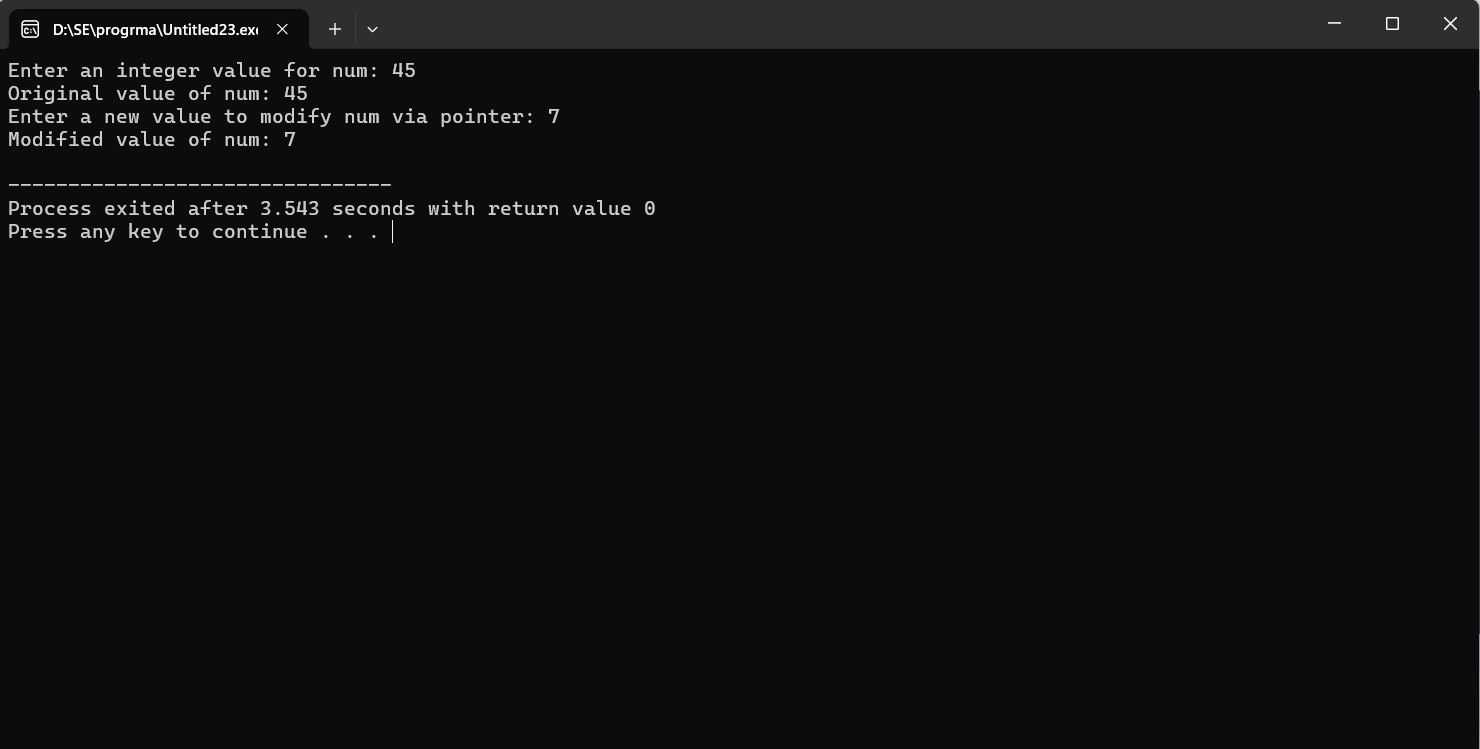
printf("Enter a new value to modify num via pointer: ");

scanf("%d", ptr);

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

}

**OUTPUT:-**



**Q15:- 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**

**CODE:-**

#include <stdio.h>

#include <string.h>

main() {

char str1[200], str2[100];

printf("Enter the first string: ");

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

printf("Enter the second string: ");

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

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

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

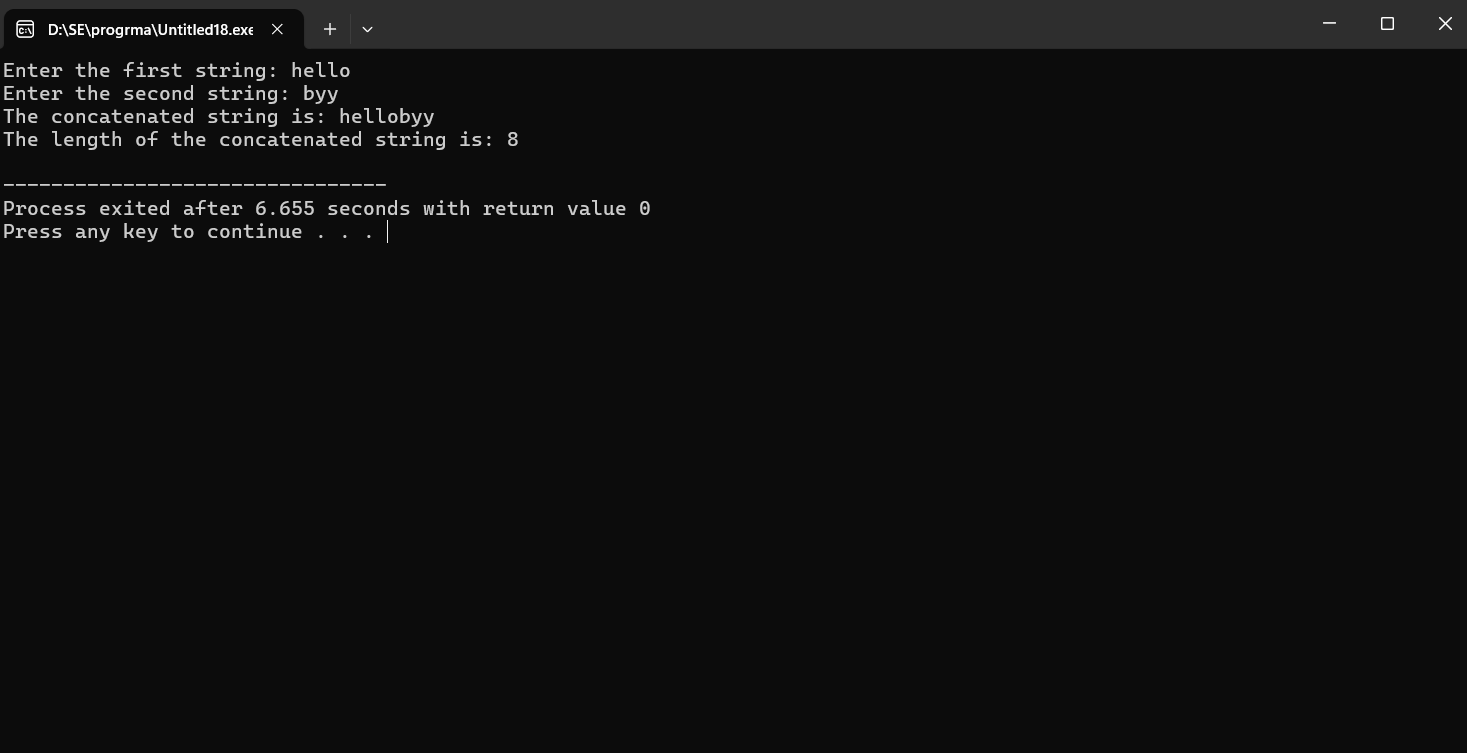
strcat(str1, str2);

printf("The concatenated string is: %s\n", str1);

printf("The length of the concatenated string is: %zu\n", strlen(str1));

}

**OUTPUT:-**

****

**Q16:- 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**

**CODE:-**

#include<stdio.h>

struct stdata{

char nm[10];

int roll;

float mrark;

}st[10];

main(){

int i,j;

printf("Enter how many student's you have: ");

scanf("%d",&j);

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

printf("Enter student's name: ");

scanf("%s",st[i].nm);

printf("Enter student's id: ");

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

printf("Enter student's Marks: ");

scanf("%f",&st[i].mrark);

}

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

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

printf("Enter student's name is:%s\n ",st[i].nm);

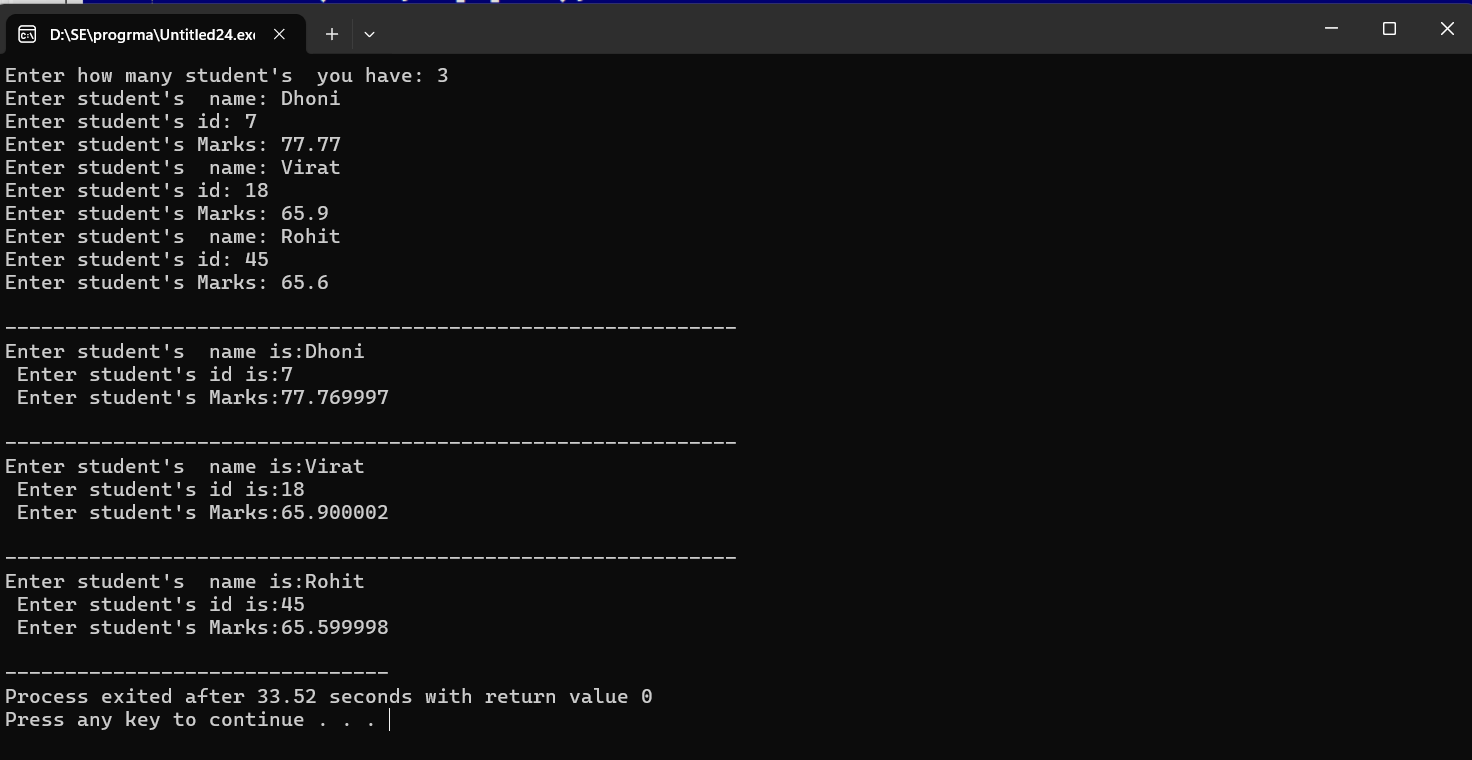
printf("Enter student's id is:%d\n ",st[i].roll);

printf("Enter student's Marks:%f\n ",st[i].mrark);

}

}

**OUTPUT:-**



**Q17:- 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**

**CODE:-**

#include <stdio.h>

main() {

FILE \*file;

char content[] = "Hello, World!";

char buffer[100];

file = fopen("file.txt", "w");

fprintf(file, "%s", content);

fclose(file);

file = fopen("file.txt", "r");

fgets(buffer, sizeof(buffer), file);

printf("File contains: %s\n", buffer);

fclose(file);

} **OUTPUT:-**

