Name:- Limbasiya Vishvash B.

Module 2 – Introduction to Programming

Overview of C Programming

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

C programming language was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.

Dennis Ritchie is known as the founder of the c language.

### Q.2:- 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.

### 1. Install a C Compiler (GCC)

Windows

* Download MinGW from [msys2.org](https://www.msys2.org/).
* Install it, then open MSYS2 terminal and run:
* Add the bin folder to your system PATH (e.g., C:\msys64\mingw64\bin).

macOS

* Open Terminal and type:
* xcode-select --install

Linux

* Open Terminal and run:
* sudo apt install gcc # For Debian/Ubuntu
* sudo dnf install gcc # For Fedora
* sudo pacman -S gcc # For Arch

2. Install an IDE

Option 1: Dev-C++

* Download from [SourceForge](https://sourceforge.net/projects/orwelldevcpp/).
* Install it and start writing C code directly.

Option 2: VS Code

* Download VS Code from [code.visualstudio.com](https://code.visualstudio.com/).
* Install the C/C++ Extension (Ctrl+Shift+X → Search "C/C++").
* Use the terminal in VS Code to compile code with gcc.

Option 3: Code::Blocks

* Download the version with MinGW from [codeblocks.org](https://www.codeblocks.org/).
* Install it, and the compiler will be set up automatically.

3. Test Your Setup

1. Write this program:
2. #include <stdio.h>
3. int main() {
4. printf("Hello, World!\n");
5. return 0;
6. }
7. Compile and run it in your IDE or terminal.

Q.3:- Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.

Basic Structure of a C Program

A C program typically consists of the following components:

1. Headers

* Header files provide predefined functions and macros.
* Commonly used header files include stdio.h (for input/output) and stdlib.h (for general utilities).
* Declared at the top of the program.

Example:

#include <stdio.h>

#include <conio.h>

2. Main Function

* The entry point of every C program.
* Written as int main() and returns an integer (usually 0 for success).

Example:

int main() {

return 0;

}

3. Comments

* Comments are used for documentation and are ignored by the compiler.
* Two types:
  + Single-line: //
  + Multi-line: /\* ... \*/

Example:

// This is a single-line comment

/\* This is

a multi-line comment \*/

4. Data Types

* Specify the type of data a variable can hold.
* Common types:
  + int: Integer
  + float: Floating-point number
  + char: Character
  + double: Double-precision floating-point number

Example:

int age = 25;

float pi = 3.14;

char grade = 'A';

double salary = 50000.50;

5. Variables

* Variables store data values.
* Declared with a data type and optionally initialized.

Example:

int a;

a = 10;

int b = 20;

#include <stdio.h>

int main() {

// Declare and initialize variables

int age = 25;

float height = 5.9;

char grade = 'A';

// Print variable values

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

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

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

Complete Example Program

}

Output of the Example

Age: 25

Height: 5.9

Grade: A

Q.4:- Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.

| Operator | Meaning of Operator |
| --- | --- |
| + | addition or unary plus |
| - | subtraction or unary minus |
| \* | multiplication |
| / | Division |
| % | remainder after division (modulo division) |

### Q.5:-Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.

### Decision-Making Statements in C

Decision-making statements control the flow of a program based on conditions.

1. if Statement

* Executes a block of code if a condition is true.

Syntax:

if (condition) {

}

Example:

#include <stdio.h>

int main() {

int num = 10;

if (num > 5) {

printf("The number is greater than 5.\n");

}

}

Output:

The number is greater than 5.

2. if-else Statement

* Executes one block if the condition is true, and another block if it is false.

Syntax:

if (condition) {

} else {

}

Example:

#include <stdio.h>

int main() {

int num = 3;

if (num > 5) {

printf("The number is greater than 5.\n");

} else {

printf("The number is 5 or less.\n");

}

}

Output:

The number is 5 or less.

3. Nested if-else Statement

* An if or if-else inside another if or else.

Syntax:

if (condition1) {

if (condition2) {

} else {

false

}

} else {

}

Example:

#include <stdio.h>

int main() {

int num = 15;

if (num > 10) {

if (num < 20) {

printf("The number is between 10 and 20.\n");

} else {

printf("The number is 20 or more.\n");

}

} else {

printf("The number is 10 or less.\n");

}

}

Output:

The number is between 10 and 20.

4. switch Statement

* Evaluates an expression and matches it against multiple cases.
* Executes the block associated with the matching case.

Syntax:

switch (expression) {

case value1:

break;

case value2:

break;

...

default:

}

Example:

#include <stdio.h>

int main() {

int day = 3;

switch (day) {

case 1:

printf("Monday\n");

break;

case 2:

printf("Tuesday\n");

break;

case 3:

printf("Wednesday\n");

break;

case 4:

printf("Thursday\n");

break;

case 5:

printf("Friday\n");

break;

default:

printf("Weekend\n");

}

}

Output:

Wednesday

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

When to Use Each Loop

1. while Loop

* Best for: Repeating a block of code as long as a condition is true, where the condition depends on external factors or dynamic data.
* Example Scenario: Reading data from a file until the end is reached.

Syntax:

while (condition) {

}

Example:

#include <stdio.h>

int main() {

int i = 0;

while (i < 5) {

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

i++;

}

}

2. for Loop

* Best for: Loops where the number of iterations is predefined or easily determined.
* Example Scenario: Printing numbers from 1 to 10 or iterating over an array.

Syntax:

for (initialization; condition; increment) {

}

Example:

#include <stdio.h>

int main() {

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

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

}

return 0;

}

3. do-while Loop

* Best for: Loops where the body must execute at least once, even if the condition is initially false.
* Example Scenario: Displaying a menu and prompting for input until a valid option is chosen.

Syntax:

do {

} while (condition);

Example:

#include <stdio.h>

int main() {

int num;

do {

printf("Enter a positive number: ");

scanf("%d", &num);

} while (num <= 0);

printf("You entered: %d\n", num);

}

Key Differences

| Aspect | While | for | do-while |
| --- | --- | --- | --- |
| Initialization | Outside the loop. | Inside the loop header. | Outside the loop. |
| Iteration Control | Manual. | Automated in header. | Manual. |
| Guaranteed Execution | No, condition must be true initially. | No, condition must be true initially. | Yes, executes at least once. |

Summary of Use Cases

* while Loop: Use when the number of iterations depends on dynamic conditions (e.g., reading input until the user types "exit").
* for Loop: Use for loops with counters or fixed ranges (e.g., iterating over arrays, running a loop 10 times).
* do-while Loop: Use when you need the loop to run at least once (e.g., prompting the user for input).

Q.7:- Explain the use of break, continue, and goto statements in C. Provide examples of each.

Break:-

#include <stdio.h>

int main() {

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

if (i == 3) {

break;

}

printf("%d ", i);

}

}

Continue:-

#include <stdio.h>

int main() {

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

if (i == 3) {

continue; // Skip the rest when i equals 3

}

printf("%d ", i);

}

return 0;

}

Goto:-

#include <stdio.h>

int main() {

int i = 1;

start:

if (i > 5) {

}

printf("%d ", i);

i++;

}

Q.8:- What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples

Functions in C

A function in C is a block of code that performs a specific task. Functions improve code modularity and reusability. A C program must have at least one function, the main() function, which is the entry point of the program.

1. Function Declaration

* Specifies the function's name, return type, and parameters.
* Acts as a "prototype" so the compiler knows how to call it.
* Syntax:
* return\_type function\_name(parameter\_list);
* Example:
* int add(int a, int b);

2. Function Definition

* Provides the body of the function (the actual implementation).
* Syntax:
* return\_type function\_name(parameter\_list) {
* // Function body
* }
* Example:
* int add(int a, int b) {
* return a + b;
* }

3. Function Call

* Executes the function by passing arguments.
* Syntax:
* function\_name(argument\_list);
* Example:
* int result = add(5, 10);

Complete Example

#include <stdio.h>

int add(int a, int b);

void greet();

int main() {

int num1, num2;

printf("Enter two numbers: ");

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

int sum = add(num1, num2);

printf("The sum is: %d\n", sum);

greet();

}

int add(int a, int b) {

return a + b;

}

(void return type)

void greet() {

printf("Hello! This is a simple C program.\n");

}

Explanation

1. Function Declaration:
   * Declares int add(int a, int b); and void greet();.
   * Informs the compiler about the functions before they are used.
2. Function Definition:
   * Implements add to compute the sum of two integers.
   * Implements greet to print a message.
3. Function Call:
   * add(num1, num2) computes the sum and returns it.
   * greet() prints a greeting message.

Q.9:- Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.

Arrays in C

An array in C is a collection of elements of the same data type stored in contiguous memory locations. Arrays allow you to store and manage multiple values efficiently using a single variable.

Key Points About Arrays

1. Arrays are zero-indexed (i.e., the first element has index 0).
2. The size of the array must be specified during declaration or initialization.
3. Syntax for declaration:
4. data\_type array\_name[size];

1. One-Dimensional Arrays

A one-dimensional array represents a list of elements (like a row).

Declaration:

int arr[5];

Example:

#include <stdio.h>

int main() {

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

printf("Elements of the array:\n");

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

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

}

}

2. Multi-Dimensional Arrays

A multi-dimensional array stores data in rows and columns, like a matrix.

2D Array Declaration:

int arr[3][3]; // Declares a 2D array with 3 rows and 3 columns

* Example:

#include <stdio.h>

int main() {

int matrix[2][2] = {

{1, 2},

{3, 4}

};

printf("Elements of the 2D array:\n");

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

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

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

}

printf("\n");

}

return 0;

}

Output:

Elements of the 2D array:

1 2

3 4

Differences Between One-Dimensional and Multi-Dimensional Arrays

| Feature | One-Dimensional Array | Multi-Dimensional Array |
| --- | --- | --- |
| Definition | A single list of elements | Multiple lists organized into rows and columns |
| Declaration | int arr[5]; | int arr[3][3]; |
| Access | arr[i] | arr[i][j] |
| Storage | Linear (like a single row) | Tabular (row × column) |
| Example Use Case | Storing marks of students | Representing a matrix |

3D Array Example

For arrays with more dimensions, the concept extends naturally. For example, a 3D array:

int arr[2][2][2]; // Represents a cube of data

Summary:

* One-dimensional arrays are simpler and suitable for linear data.
* Multi-dimensional arrays are used for structured data like grids or matrices.

Q.10:- o Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

Pointers in C

A pointer is a variable that stores the memory address of another variable. Pointers are essential for dynamic memory allocation, efficient array handling, and function manipulation in C.

Declaration and Initialization

1. Declaration:
2. data\_type \*pointer\_name;

Example:

int \*ptr; // Pointer to an integer

1. Initialization:
2. pointer\_name = &variable\_name; // Assigns the address of the variable

Example:

int num = 10;

int \*ptr = &num; // Pointer stores the address of num

Why Pointers Are Important

1. Dynamic Memory Allocation: Enable allocation and deallocation of memory at runtime (using malloc, calloc, etc.).
2. Efficient Array and String Handling: Simplify operations on arrays and strings.
3. Pass by Reference: Allow functions to modify variables directly.
4. Access Hardware Resources: Used in low-level programming like interacting with memory or hardware.

Example

#include <stdio.h>

int main() {

int num = 10;

int \*ptr = &num; // Pointer to num

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

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

printf("Value stored in pointer ptr: %p\n", ptr);

printf("Value pointed to by ptr: %d\n", \*ptr); // Dereference pointer

return 0;

}

Q.11:- Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.

String Handling Functions in C

The string handling functions in C, provided by the <string.h> library, are used to manipulate and process strings efficiently.

1. strlen()

* Purpose: Returns the length of a string (excluding the null terminator \0).
* Syntax: size\_t strlen(const char \*str);
* Example:
* #include <stdio.h>
* #include <string.h>
* int main() {
* char str[] = "Hello, World!";
* printf("Length of the string: %zu\n", strlen(str));
* return 0;
* }

Output:

Length of the string: 13

* Use Case: Determine the length of a string for processing or validation.

2. strcpy()

* Purpose: Copies one string into another.
* Syntax: char \*strcpy(char \*destination, const char \*source);
* Example:
* #include <stdio.h>
* #include <string.h>
* int main() {
* char source[] = "OpenAI";
* char destination[20];
* strcpy(destination, source);
* printf("Copied string: %s\n", destination);
* return 0;
* }

Output:

Copied string: OpenAI

* Use Case: Initialize or duplicate strings.

3. strcat()

* Purpose: Concatenates (appends) one string to another.
* Syntax: char \*strcat(char \*destination, const char \*source);
* 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;
* }

Output:

Concatenated string: Hello, World!

* Use Case: Combine strings to form a single message or sentence.

4. strcmp()

* Purpose: Compares two strings lexicographically.
* Syntax: int strcmp(const char \*str1, const char \*str2);
  + Returns 0 if strings are equal.
  + Returns a negative value if str1 < str2.
  + Returns a positive value if str1 > str2.
* Example:
* #include <stdio.h>
* #include <string.h>
* int main() {
* char str1[] = "OpenAI";
* char str2[] = "OpenAI";
* char str3[] = "ChatGPT";
* printf("Comparison result (str1, str2): %d\n", strcmp(str1, str2));
* printf("Comparison result (str1, str3): %d\n", strcmp(str1, str3));
* return 0;
* }

Output:

Comparison result (str1, str2): 0

Comparison result (str1, str3): 1

* Use Case: Check if two strings are equal or sort strings.

5. strchr()

* Purpose: Finds the first occurrence of a character in a string.
* Syntax: char \*strchr(const char \*str, int character);
* Example:
* #include <stdio.h>
* #include <string.h>
* int main() {
* char str[] = "Hello, World!";
* char \*pos = strchr(str, 'W');
* if (pos) {
* printf("Character 'W' found at position: %ld\n", pos - str);
* } else {
* printf("Character not found.\n");
* }
* return 0;
* }

Output:

Character 'W' found at position: 7

* Use Case: Search for a specific character in a string for parsing or validation.

Summary of Uses

| Function | Purpose | Example Use Case |
| --- | --- | --- |
| strlen() | Get string length | Validating input string size |
| strcpy() | Copy a string | Initializing or duplicating a string |
| strcat() | Concatenate two strings | Forming sentences or messages |
| strcmp() | Compare two strings  Lexicographically | Sorting or checking equality of strings |
| strchr() | Find first occurrence of a character | Parsing or locating specific characters |

Q.12:- Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.

Structures in C

A structure in C is a user-defined data type that groups variables of different types under one name. It allows for organizing complex data more effectively.

Declaring a Structure

* Use the struct keyword to define a structure.
* Syntax:
* struct structure\_name {
* data\_type member1;
* data\_type member2;
* ...
* };
* Example:
* struct Student {
* int id;
* char name[50];
* float marks;
* };

Initializing a Structure

1. Direct Initialization:
2. struct Student student1 = {101, "John Doe", 89.5};
3. Individual Assignment:
4. struct Student student2;
5. student2.id = 102;
6. strcpy(student2.name, "Jane Smith");
7. student2.marks = 92.0;

Accessing Structure Members

* Use the dot (.) operator for accessing members of a structure variable.
* Syntax:
* structure\_variable.member\_name

Complete Example

#include <stdio.h>

#include <string.h>

// Structure declaration

struct Student {

int id;

char name[50];

float marks;

};

int main() {

// Direct initialization

struct Student student1 = {101, "John Doe", 89.5};

// Individual member assignment

struct Student student2;

student2.id = 102;

strcpy(student2.name, "Jane Smith");

student2.marks = 92.0;

// Access and print structure members

printf("Student 1:\n");

printf("ID: %d\nName: %s\nMarks: %.2f\n\n", student1.id, student1.name, student1.marks);

printf("Student 2:\n");

printf("ID: %d\nName: %s\nMarks: %.2f\n", student2.id, student2.name, student2.marks);

return 0;

}

Explanation

1. Declaration:
   * The Student structure contains an integer id, a character array name, and a float marks.
2. Initialization:
   * student1 is initialized directly during declaration.
   * student2 is initialized by assigning values to each member individually.
3. Access:
   * Members are accessed using the . operator (e.g., student1.id).

Output

Student 1:

ID: 101

Name: John Doe

Marks: 89.50

Student 2:

ID: 102

Name: Jane Smith

Marks: 92.00

Q.13:- Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files

C provides a number of build-in function to perform basic file operations:

* fopen() - create a new file or open a existing file
* fclose() - close a file
* getc() - reads a character from a file
* putc() - writes a character to a file
* fscanf() - reads a set of data from a file
* fprintf() - writes a set of data to a file
* getw() - reads a integer from a file
* putw() - writes a integer to a file
* fseek() - set the position to desire point
* ftell() - gives current position in the file
* rewind() - set the position to the beginning point

LAB EXERCISE

Q:-1 Write your first program to print "Hello, World!" and run it.

Ans:-

#include<stdio.h>

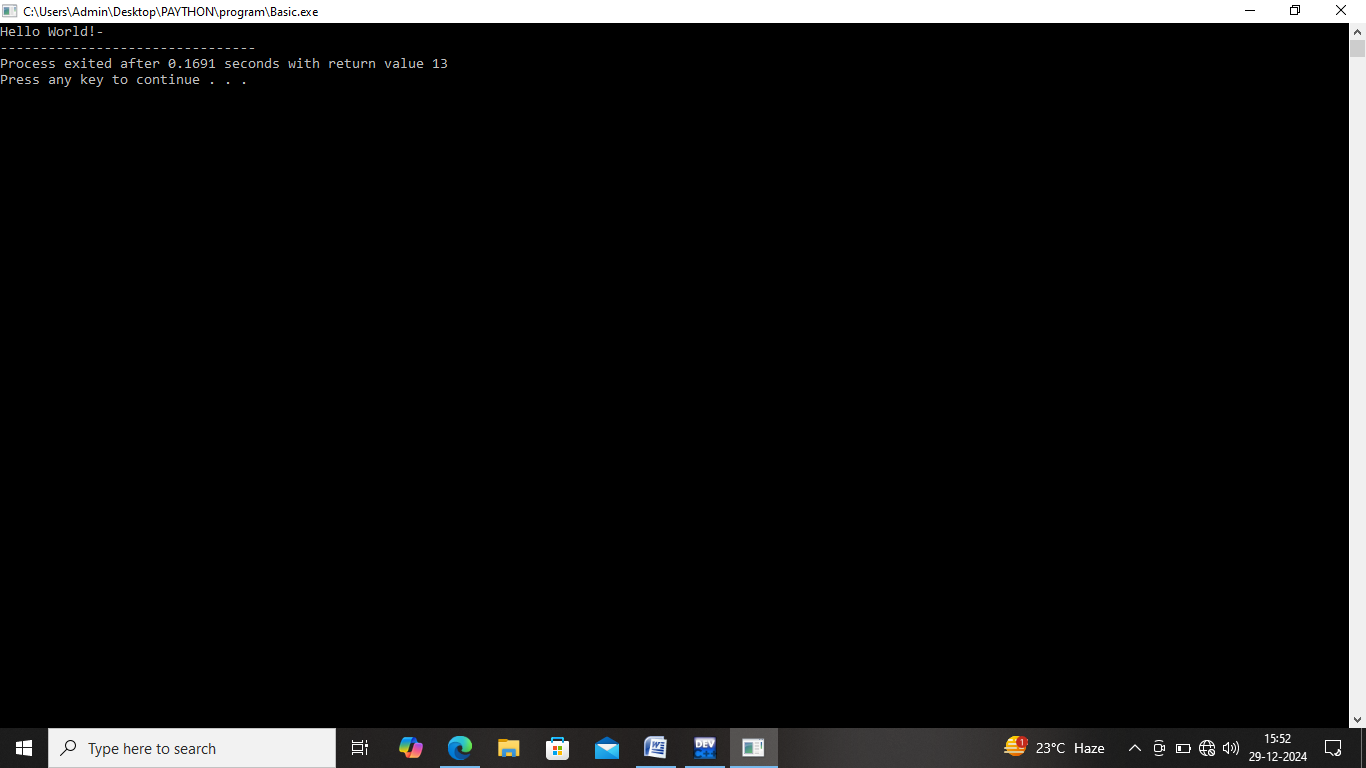
main()

{

printf("Hello World!");

}

Output:-



Q.2:- Write a C program that includes variables, constants, and comments. Declare and use different data types (int, char, float) and display their values.

Ans:-

#include<stdio.h>

main()

{

int n=10;

char V='V';

float pr=3.14;

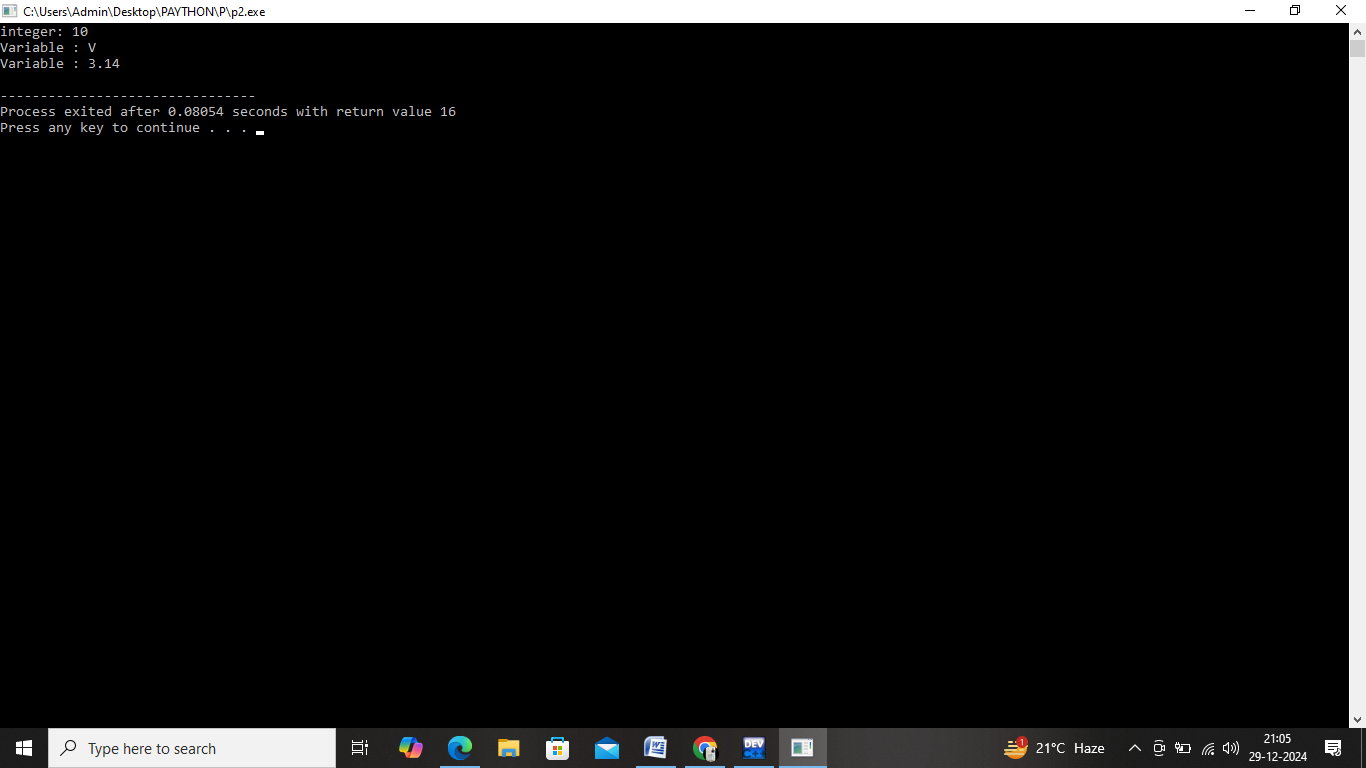
printf("integer: %d\n", n);

printf("Variable : %c\n", V);

printf("Variable : %.2f\n",pr);

}

Output:-



Q.3:- Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results.

Ans:-

Arithmetic Operation:

#include <stdio.h>

main()

{

int num1, num2;

printf("Enter two integers: ");

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

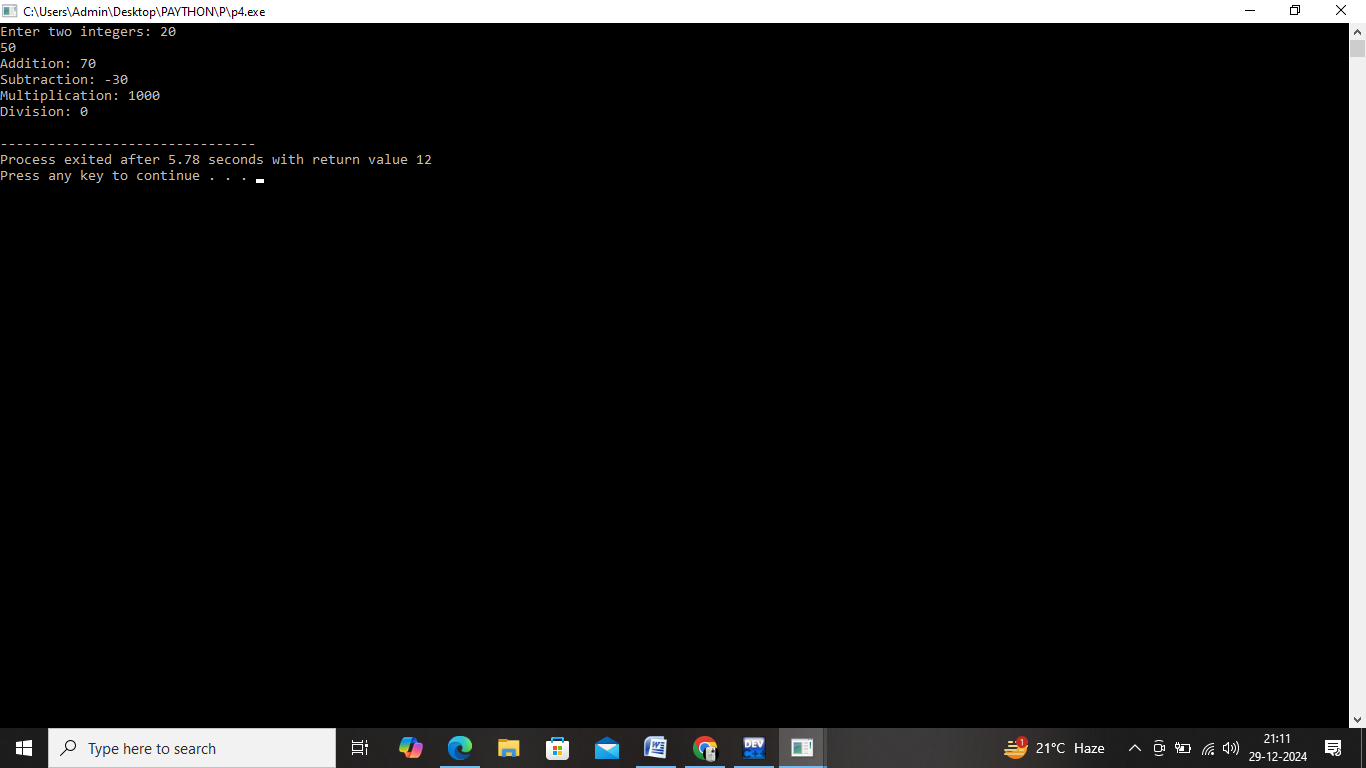
printf("Addition: %d\n", num1 + num2);

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

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

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

}



Q.4:- Write a C program to check if a number is even or odd using an if-else statement.

Ans:-

#include<stdio.h>

main(){

int num1;

printf("Enter number 1: ");

scanf("%d",&num1);

if(num1%2==0)

{

printf("This number is even");

}

else

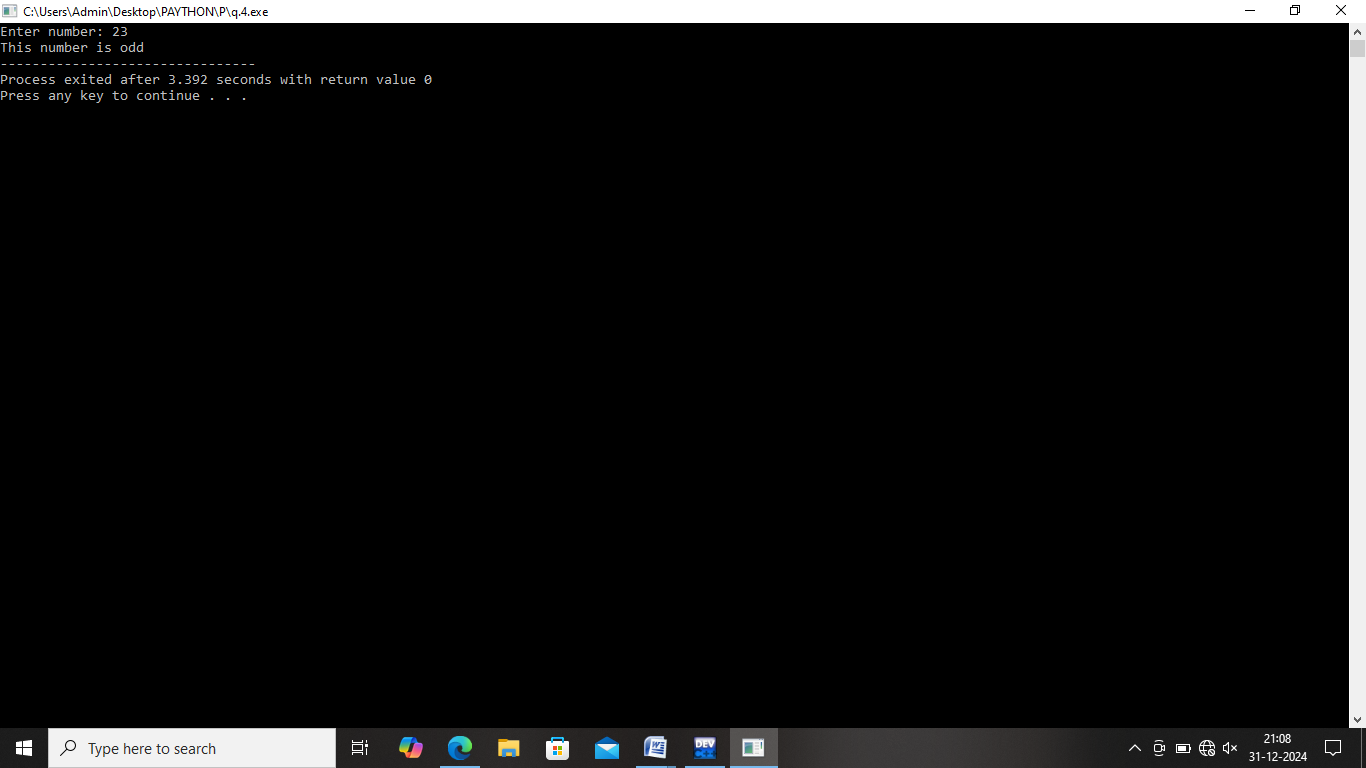
{

printf("This number is odd");

}

}

Output:-



Q.5:- Extend the program using a switch statement to display the month name based on the user’s input (1 for January, 2 for February, etc.).

Ans:-

#include<stdio.h>

main(){

int choice;

printf("1 to 12: ");

scanf("%d",&choice);

switch(choice){

case 1:

printf("month is january");

break;

case 2:

printf("month is febuary");

break;

case 3:

printf("month is march");

break;

case 4:

printf("month is april");

break;

case 5:

printf("month is may");

break;

case 6:

printf("month is june");

break;

case 7:

printf("month is july");

break;

case 8:

printf("month is august");

break;

case 9:

printf("month is september");

break;

case 10:

printf("month is october");

break;

case 11:

printf("month is november");

break;

case 12:

printf("month is december");

break;

default:

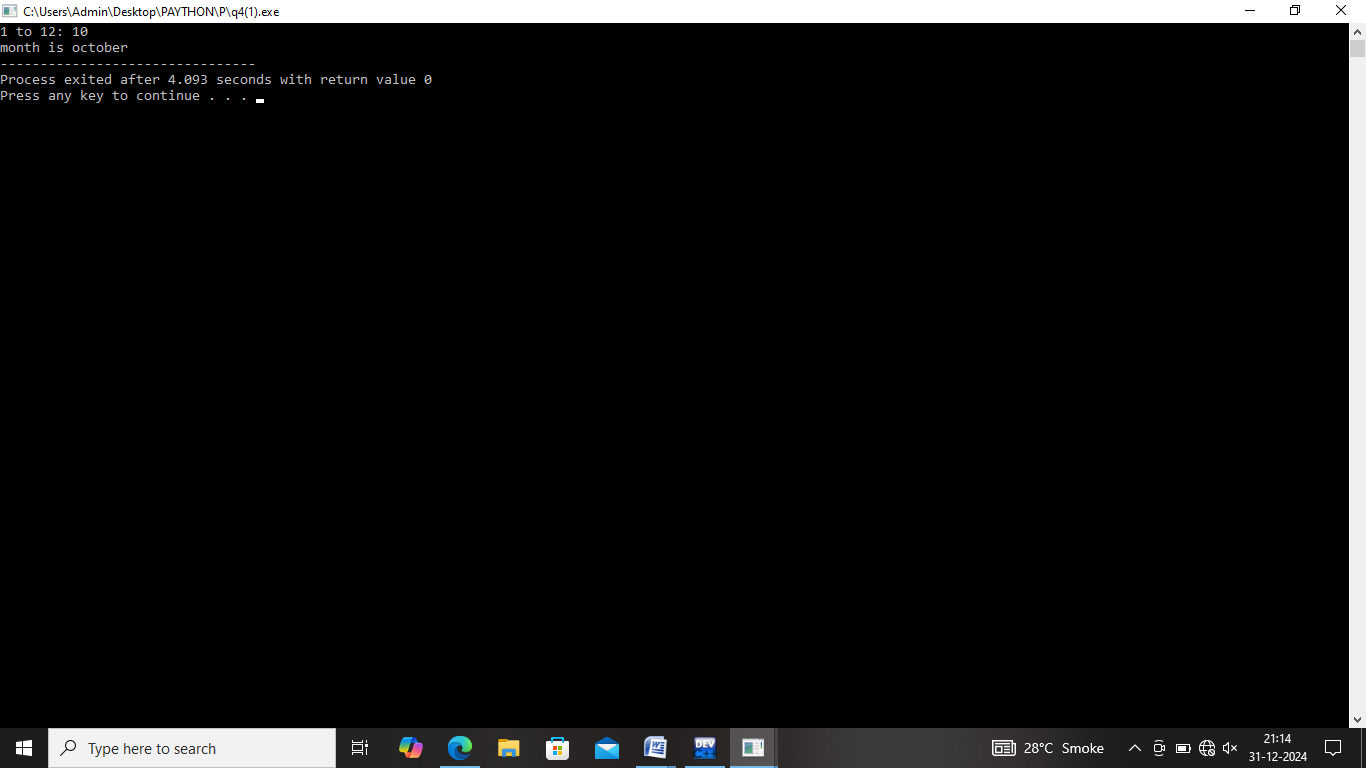
printf("Error!");

break;

}

}

Output:-



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

Ans:-

For Loop:

#include <stdio.h>

main() {

int i;

printf("for loop:\n");

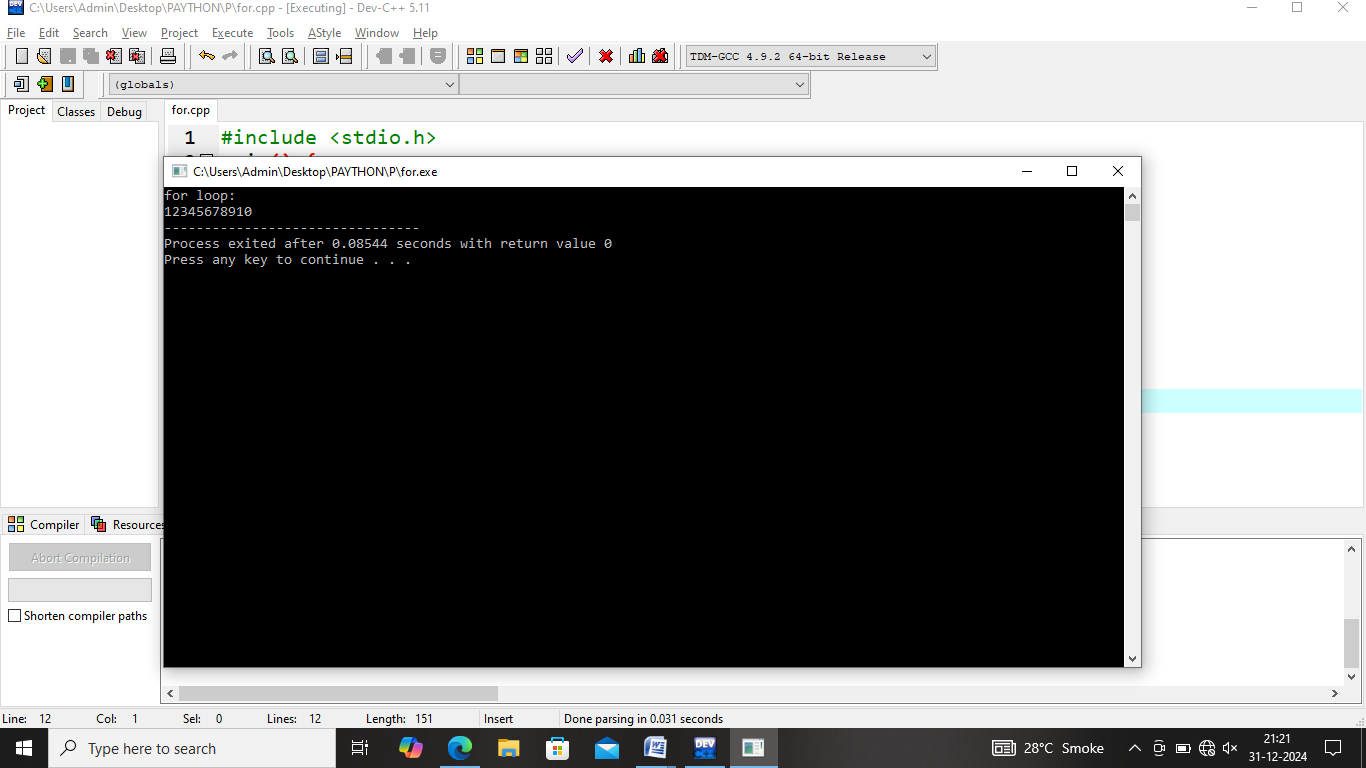
for (i=1; i<=10; i++)

{

printf("%d", i);

}

}

Output:-

While Loop:

#include <stdio.h>

main() {

int i;

printf("while loop:\n");

i = 1;

while (i <= 10)

{

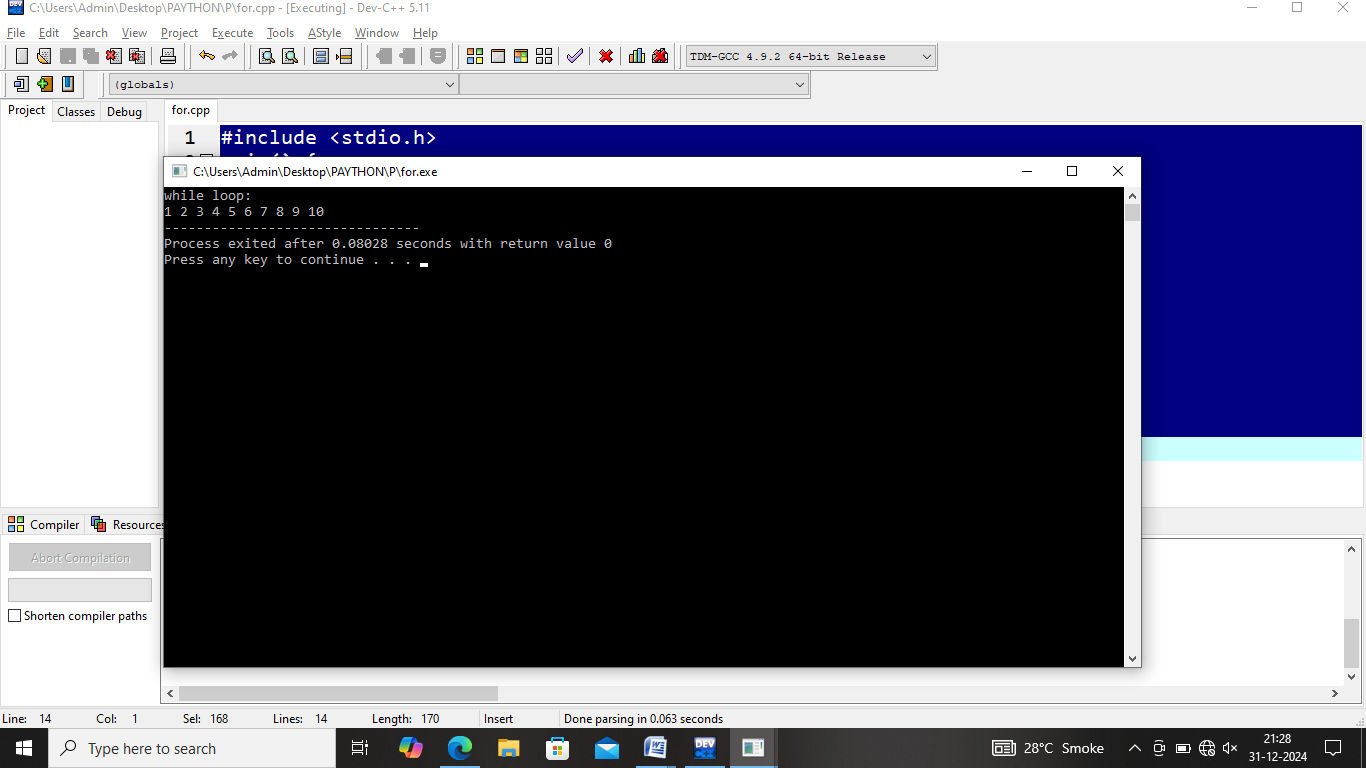
printf("%d ", i);

i++;

}

}

Output:



Do While loop:

#include <stdio.h>

main() {

int i;

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

i = 1;

do {

printf("%d ", i);

i++;

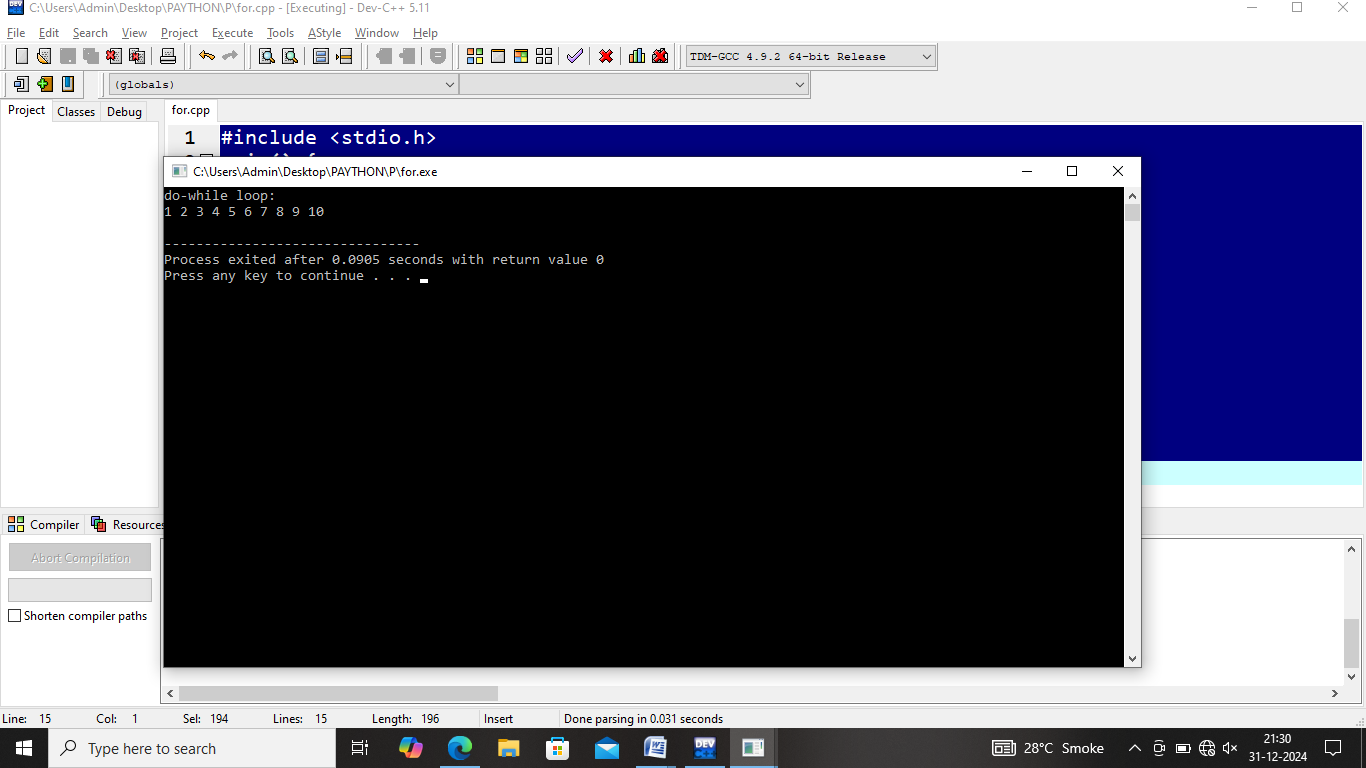
}

while (i <= 10);

printf("\n");

}

Output:



Q.7:- Modify the program to skip printing the number 3 using the continue statement.

Ans:-

#include<stdio.h>

main()

{

int i;

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

{

if(i==3)

{

continue;

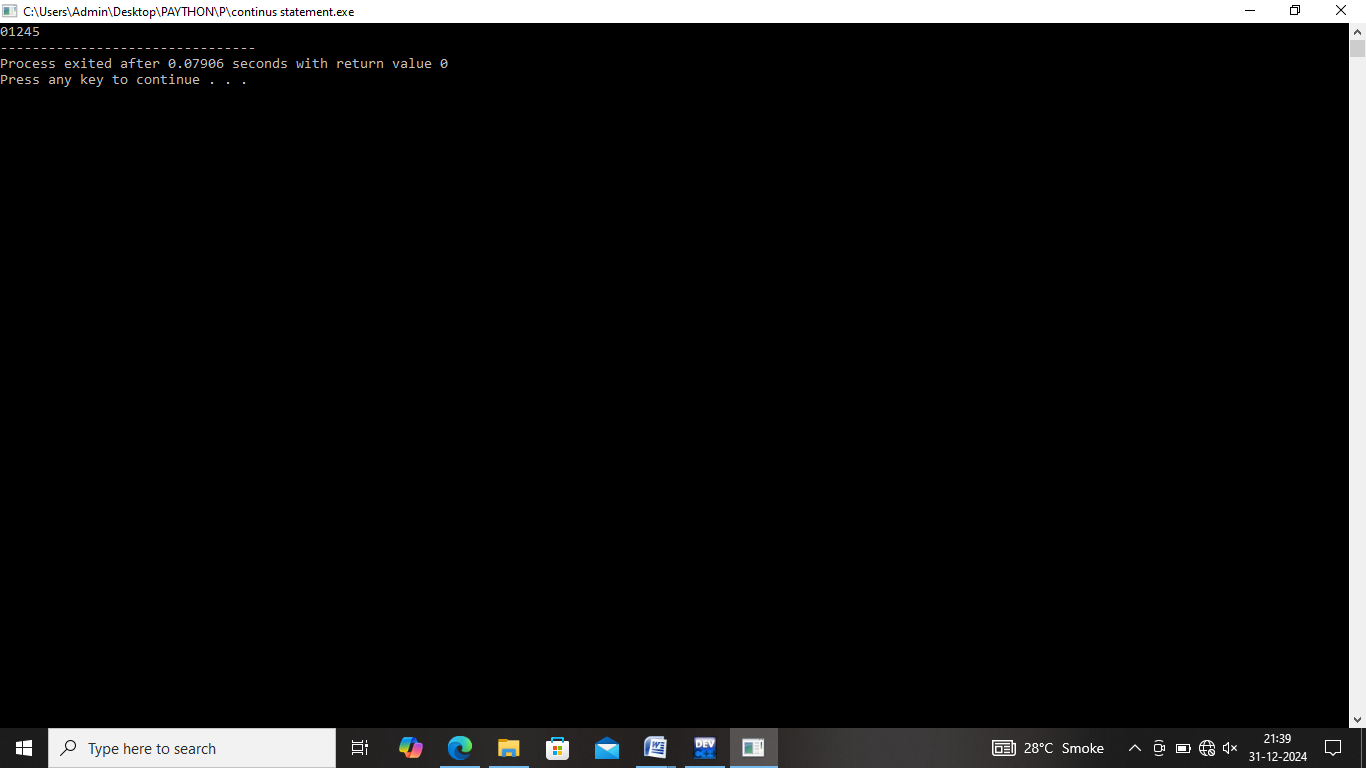
}

printf("%d",i);

}

}

Output:



Q.8:- Write a C program that uses the break statement to stop printing numbers when it reaches 5.

Ans:-

#include<stdio.h>

main(){

int i;

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

{

if(i==5)

{

break;

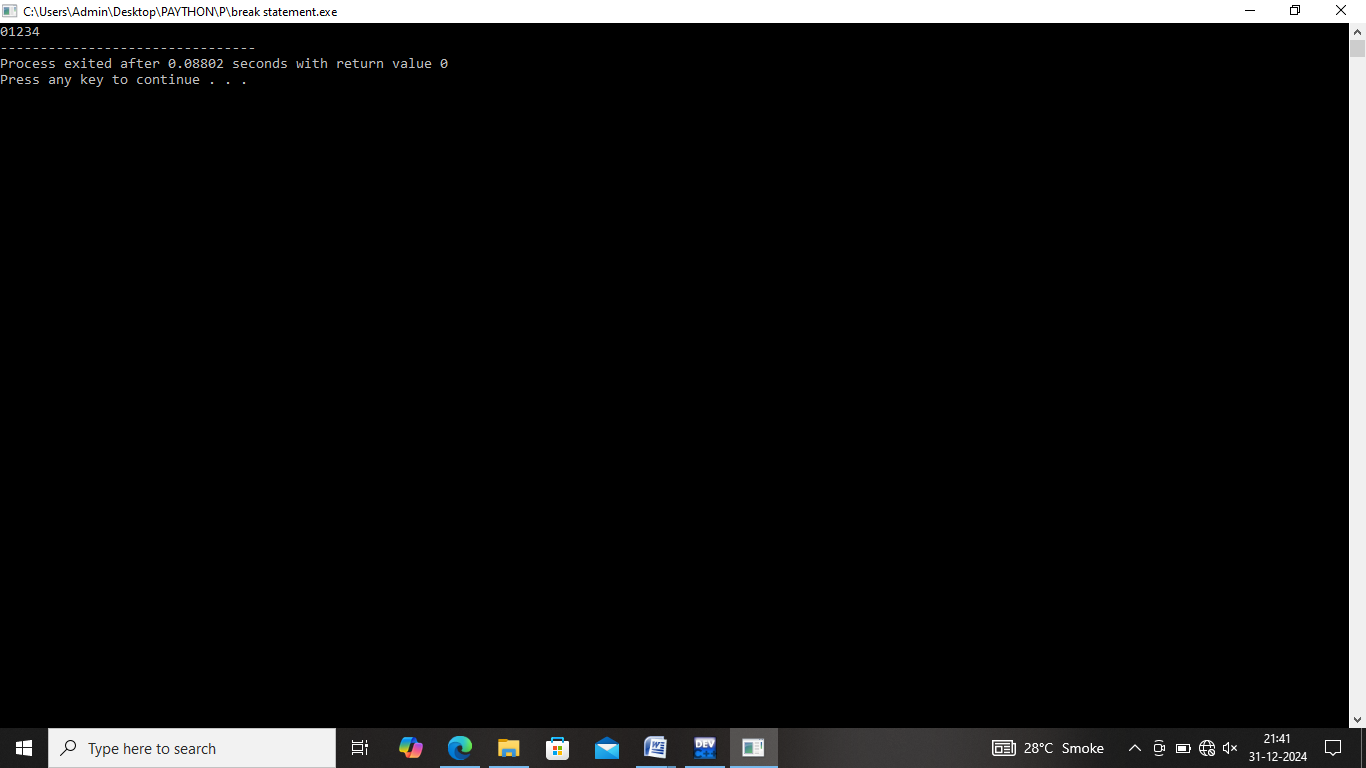
}

printf("%d",i);

}

}

Output:-



Q.9:- Write a C program that stores 5 integers in a one-dimensional array and prints them.

Ans:-

#include<stdio.h>

main()

{

int arr[5];

printf("Enter 5 integers array:\n\n");

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

{

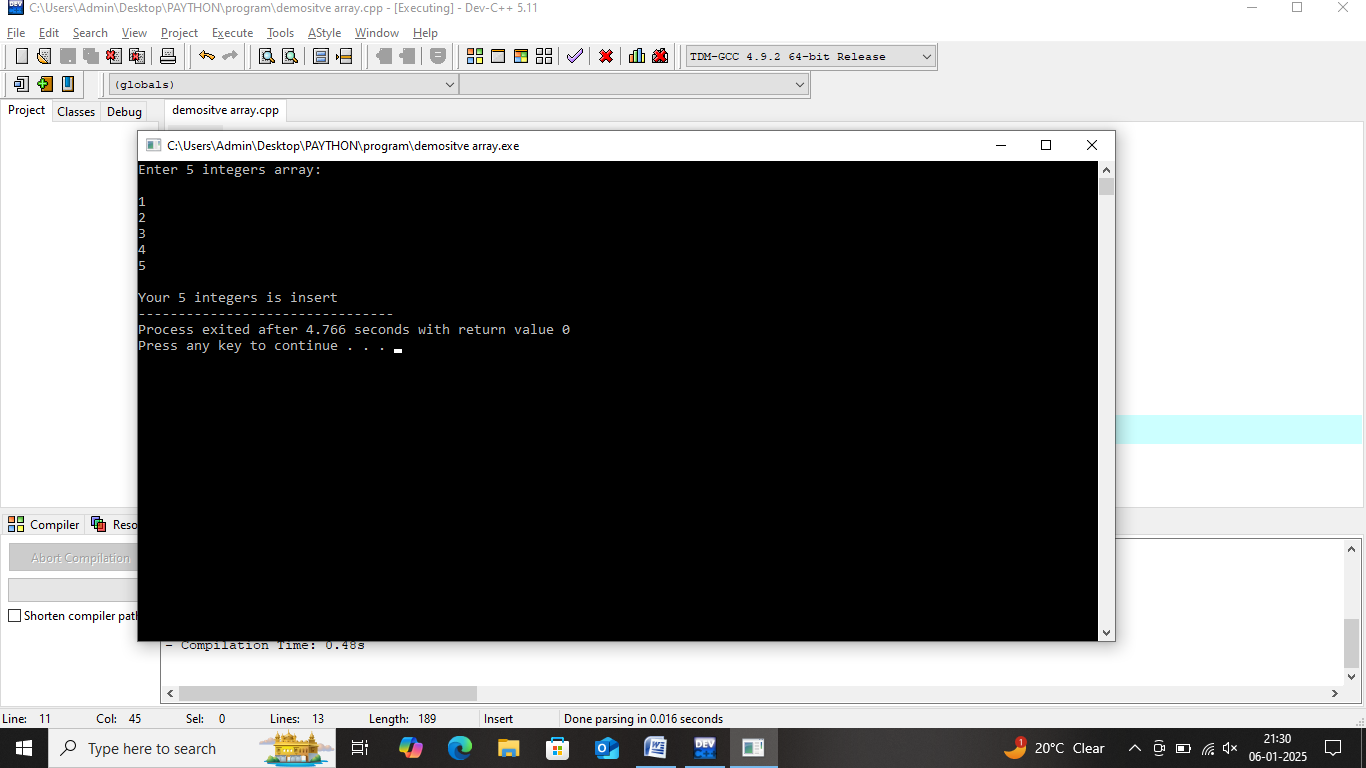
scanf("%d",&arr[i]);

}

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

}

Output:-



Q.10:- . Extend this to handle a two-dimensional array (3x3 matrix) and calculate the sum of all elements.

Ans:-

#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];

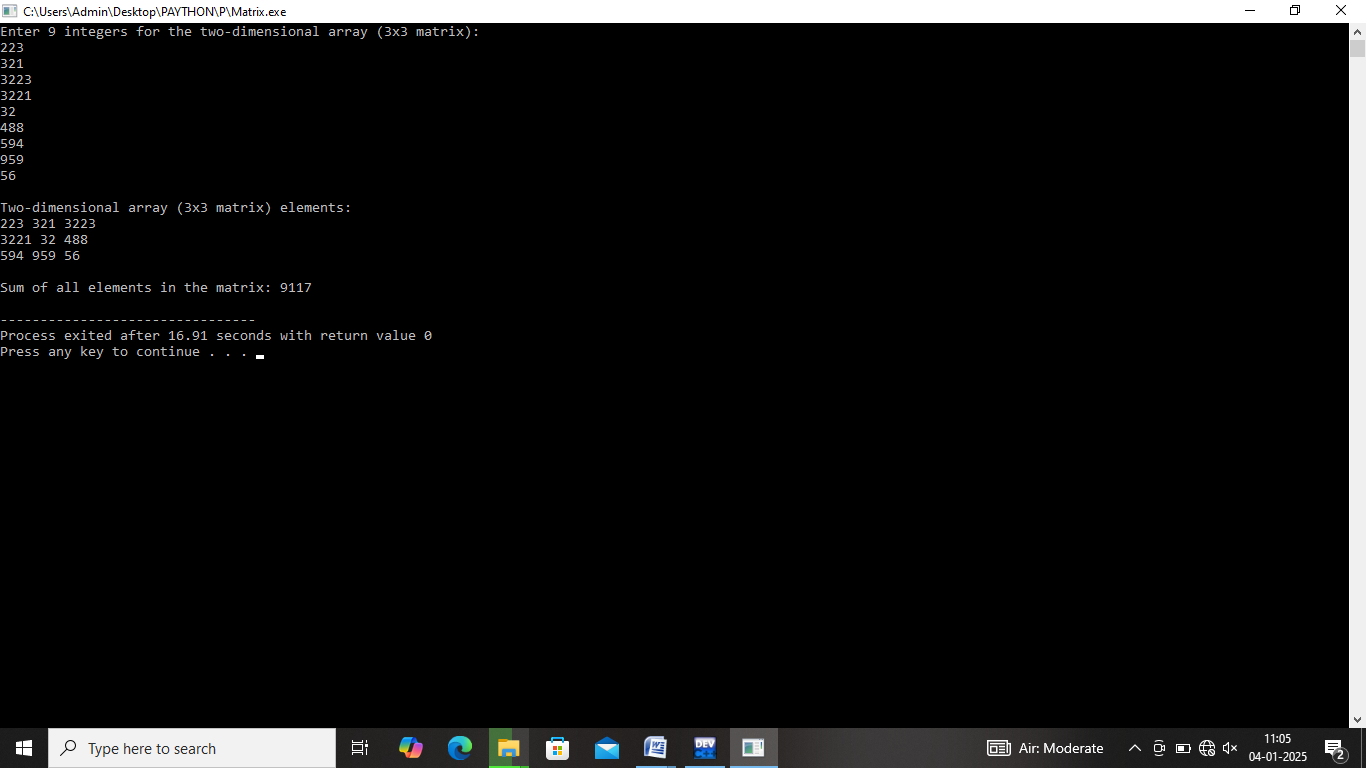
}

}

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

}

Output:-



Q.11:- 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>

main()

{

int num, \*pr;

printf("Enter integer value: ");

scanf("%d", &num);

pr = &num;

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

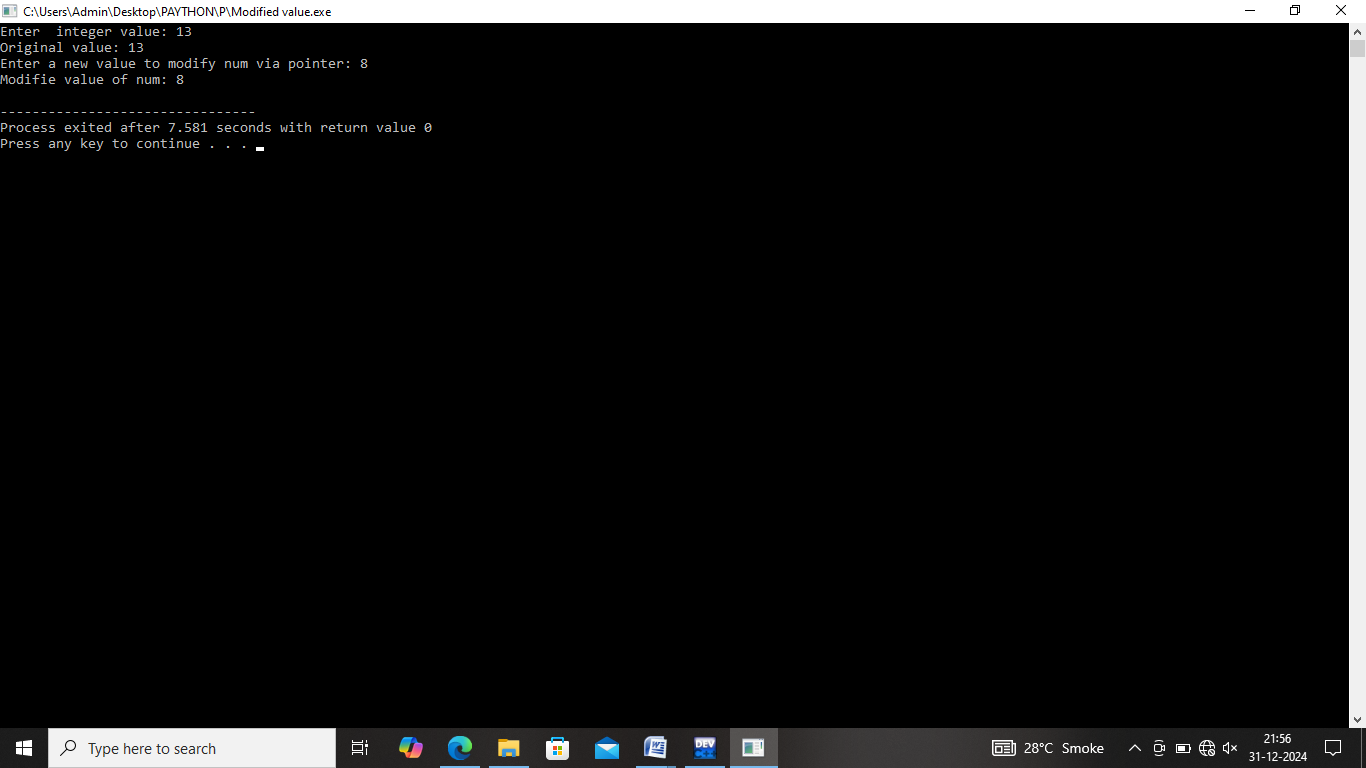
printf("Enter a new value to modify num via pointer: ");

scanf("%d", pr);

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

}

Output:-



Q.12:- 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>

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';

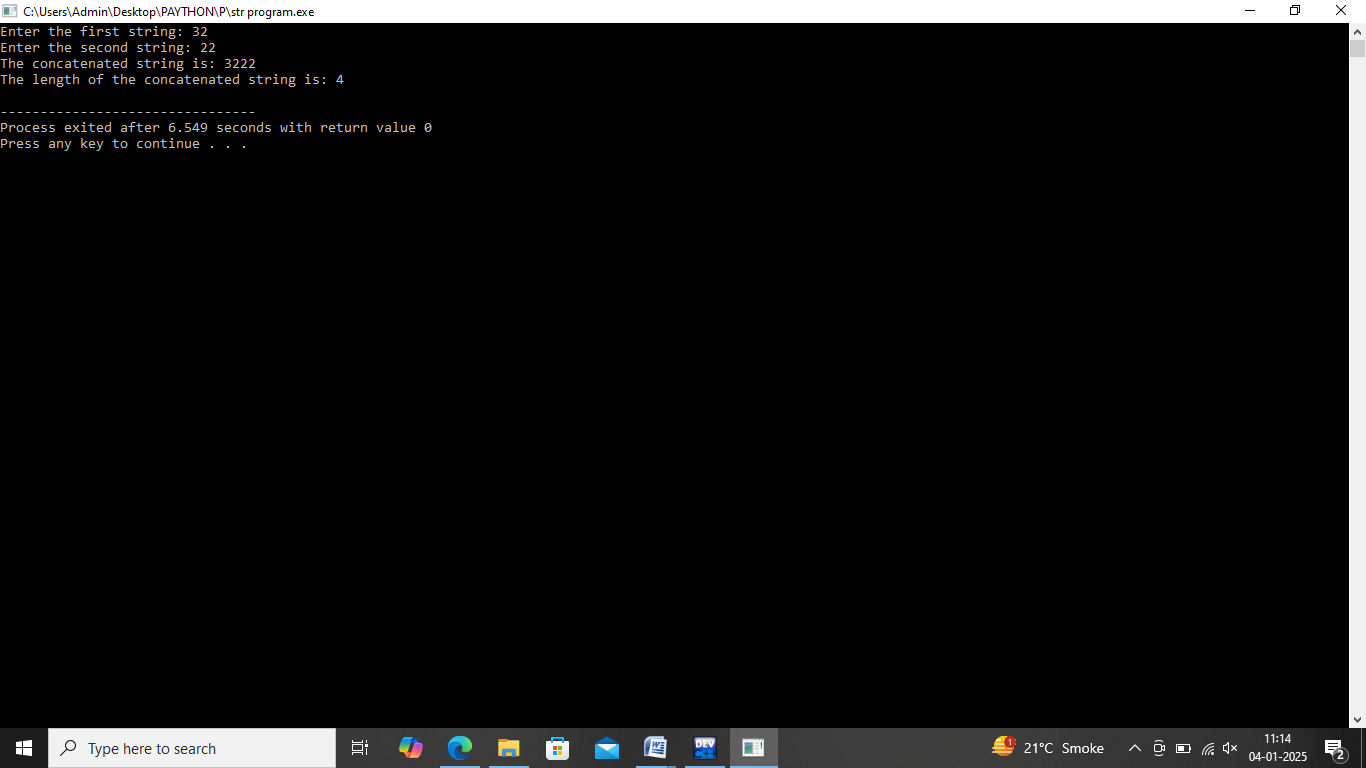
strcat(str1, str2);

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

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

}

Output:-



Q.13:- 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>

struct stdata

{

char nm[10];

int roll;

float mark;

}st[10];

main(){

int i,j;

printf("Enter how many students: ");

scanf("%d",&j);

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

{

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

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

printf("\nEnter student's id: ");

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

printf("\nEnter student's Marks: ");

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

}

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

{

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n)";

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

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

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

}

}

Output:-

