**F. Y. B. Tech Academic Year 2021-22**

**Subject:** Programming and Problem Solving **Trimester:** 2

**Name:** Shreerang Mhatre  **Division:** 11

**Roll No:** 111056  **Batch:** K3

**ASSIGNMENT NO: 6**

**AIM**: Write a C function to compute factorial of the given number using

recursion.

**OBJECTIVE:**

1. To learn and understand Functions in C.

2. To learn and understand Recursive Function.

**THEORY:**

**User Defined Functions**

A function is a block of code that performs a specific task. C allows you to define functions according to your need. These functions are known as user-defined functions. For example: Suppose, you need to create a circle and color it depending upon the radius and color. You can create two functions to solve this problem:

**createCircle() function**

**color() function**

* **Decleration:** A function declaration is a frame (prototype)of function that contains the function’s name, list of parameters and return type and ends with the semicolon. but it doesn't contain the function body.
* **Definition:** C programming language allows coders to define functions to perform

special tasks. As functions are defined by users, they are called user-defined functions. user-defined functions have contained the block of statements which are written by the user to perform a task.

* **Call to Function Syntax:**  A function is a group of statements that together perform a task. Every C program has at least one function, which is main (), and all the most trivial programs can define additional functions. You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task. A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.
* **Example:** Here is an example to add two integers. To perform this task, we have created an user-defined.

#include <stdio.h>

int addNumbers(int a, int b); // function prototype

int main()

{

int n1,n2,sum;

printf("Enters two numbers: ");

scanf("%d %d",&n1,&n2);

sum = addNumbers(n1, n2); // function call

printf("sum = %d",sum);

return 0;

}

int addNumbers(int a, int b) // function definition

{

int result;

result = a+b;

return result; // return statement

}

* **Use of Return Statement:** A return statement ends the execution of a function, and returns control to the calling function. Execution resumes in the calling function at the point immediately following the call. A return statement can return a value to the calling function.

**What is Formal and Actual argument?**

**Formal arguments**- Arguments which are mentioned in the definition of the function is

called formal arguments. Formal arguments are very similar to local variables inside the

function. Just like local variables, formal arguments are destroyed when the function

ends.

int factorial (int n)

{

// write logic here

}

Here n is the formal argument. Things to remember about actual and formal arguments.

**Actual arguments**- Arguments which are mentioned in the function call is known as the

actual argument. For example:

func1(12, 23);

here 12 and 23 are actual arguments.

Actual arguments can be constant, variables, expressions etc.

**Implementation:**

**Platform: 64 –**bit Windows 10**.**

**Technology:** Open Source Visual Studio Code

**Algorithm:**

**Step1) START**

**Step2) Ask the user to enter an integer to find the factorial**

**Step3) Read the integer and assign it to a variable**

**Step4) From the value of the integer up to 1, multiply each digit and update the final**

**value**

**Step5) The final value at the end of all the multiplication till 1 is the factorial**

**Step6) STOP**

**Pseudocode of C Program for Factorial:**

**Using the above algorithm, we can create pseudocode for the C program to find**

**factorial of a number, such as:**

**Start**

**procedure fact(num)**

**until num=1**

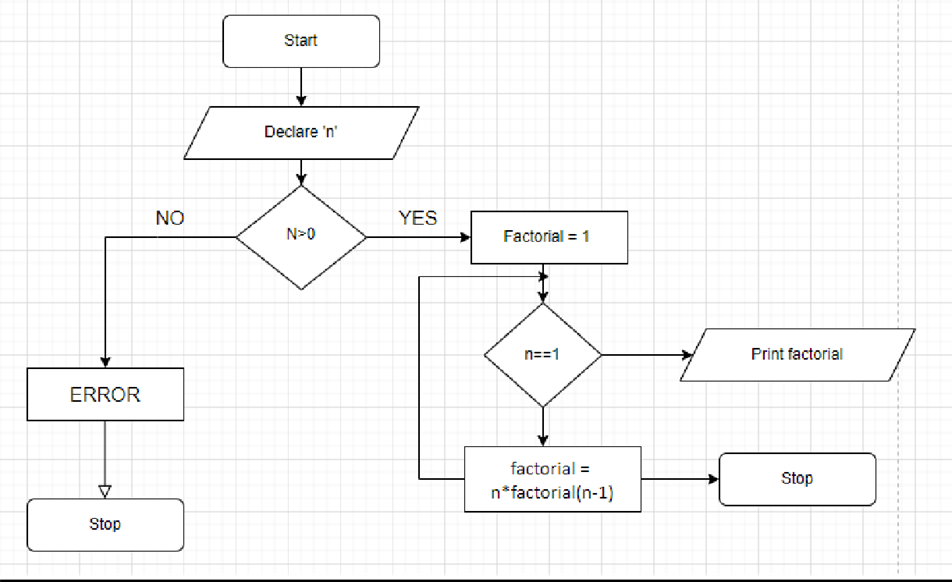
**fact = fact\*(num-1)**

**Print fact**

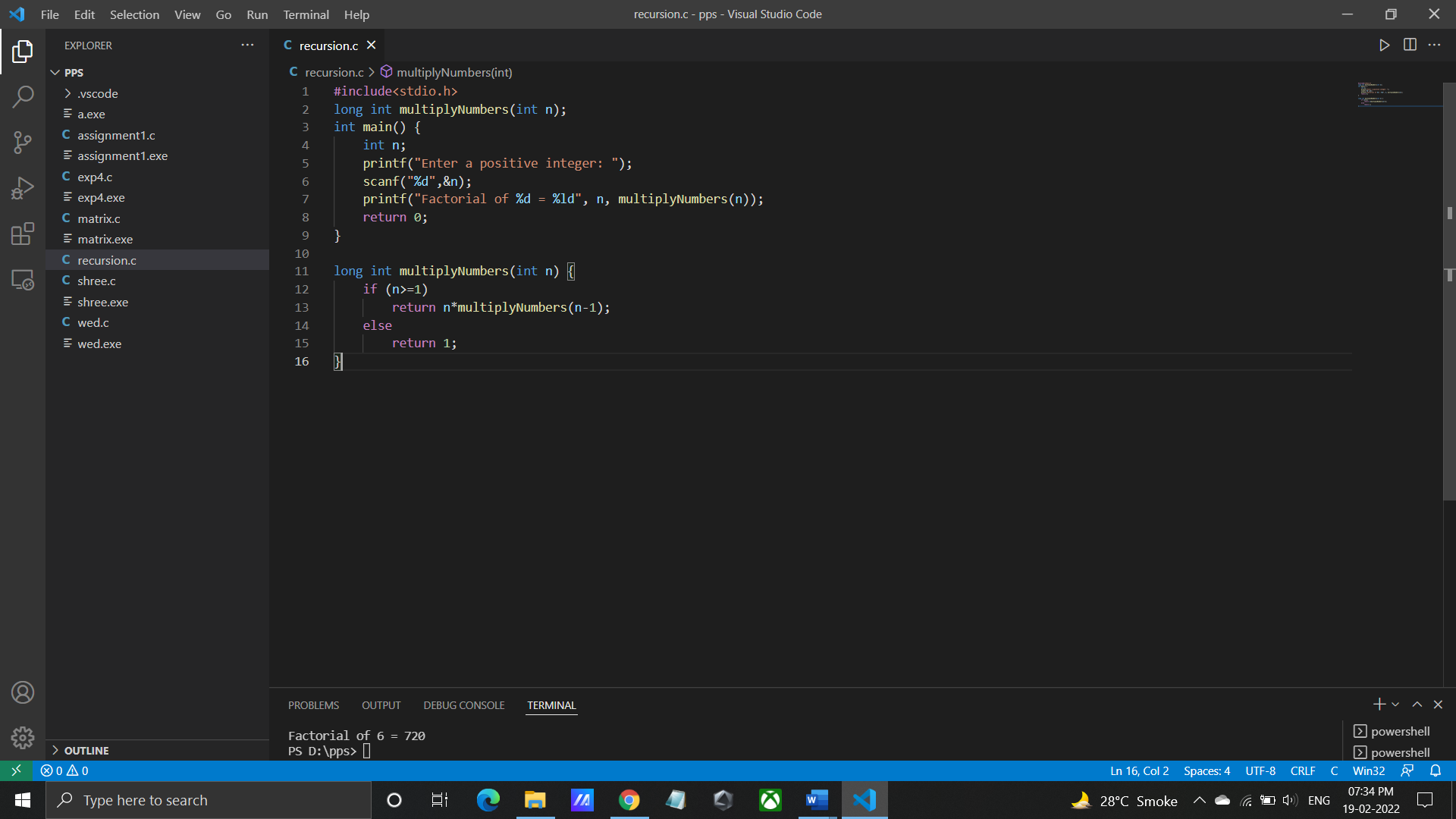
**end procedure**

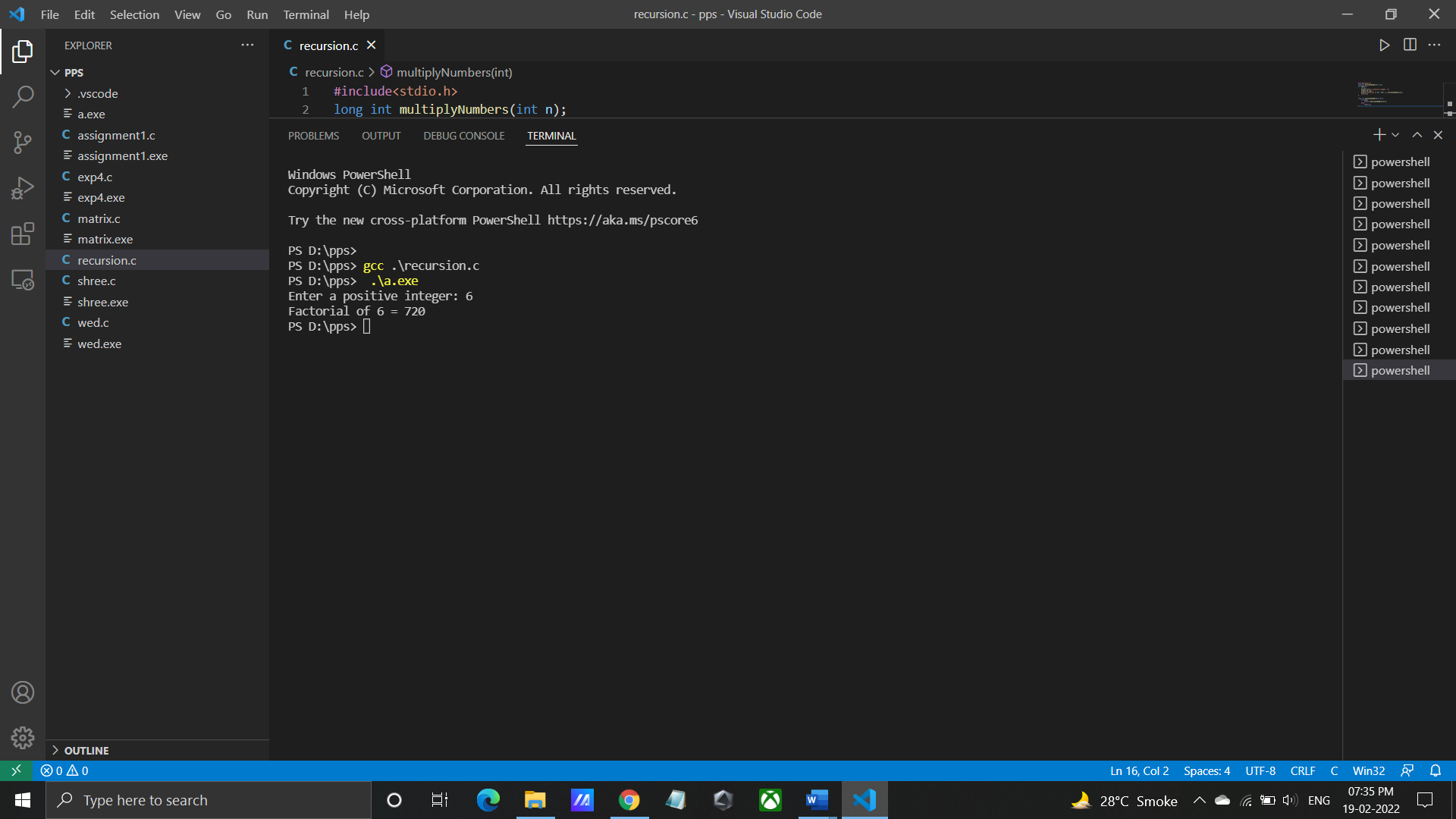
**Stop**

**Flowchart:**

****

**Visual Studio Code :**

**Code for given program:**

**Output for given program:**

**INPUT:** Enter first matrix elements: 1 4 3 6 7 4 0 8 4

Enter second matrix elements: 2 4 5 9 6 7 0 3 4

**OUTPUT:** Addition of two matrices is:

3 8 8

15 13 11

**Program/ Code:**

#include<stdio.h>

long int multiplyNumbers(int n);

int main() {

int n;

printf("Enter a positive integer: ");

scanf("%d",&n);

printf("Factorial of %d = %ld", n, multiplyNumbers(n));

return 0;

}

long int multiplyNumbers(int n) {

if (n>=1)

return n\*multiplyNumbers(n-1);

else

return 1;

}

**INPUT:** Any input number (e.g., 6)

**OUTPUT:** The Factorial of 6 is 720

**CONCLUSION:** Thus implemented the program to compute factorial using recursion.

**FAQ:**

1. What is Recursion Function?

ANS: A recursive function is one which calls itself. This is another complicated idea which

you are unlikely to meet frequently. Recursive functions are useful in evaluating certain

types of mathematical function. You may also encounter certain dynamic data structures

such as linked lists or binary trees. Recursion is a very useful way of creating and accessing these structures.

Example- int fact (int n)

{ if (n < = 1) // base case return 1; else

return n\*fact(n-1); }

In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

1. What are local variables?

ANS: Local variables are declared within a function. They are created anew each time the function is called, and destroyed on return from the function. Values passed to the

function as arguments can also be treated like local variables (Pass by value).

Example: if else block, loop block etc.