

Investigación sobre recursividad

Ruiz Pérez Brian Luis

06/03/2025

Recursividad

Definición de recursividad:

La recursividad es una técnica de programación en la que una función se llama así misma para resolver un problema. Se basa en dividir un problema en subproblemas más pequeños y similares al original, hasta alcanzar un caso base que pueda resolverse directamente.

Una función recursiva generalmente consta de dos partes esenciales:

1. Caso base: Es la condición de parada que evita que la función se ejecute indefinidamente.
2. Caso recursivo: Es la parte donde la función se llama así misma con una versión reducida del problema original.

¿Para qué se utiliza la recursividad?

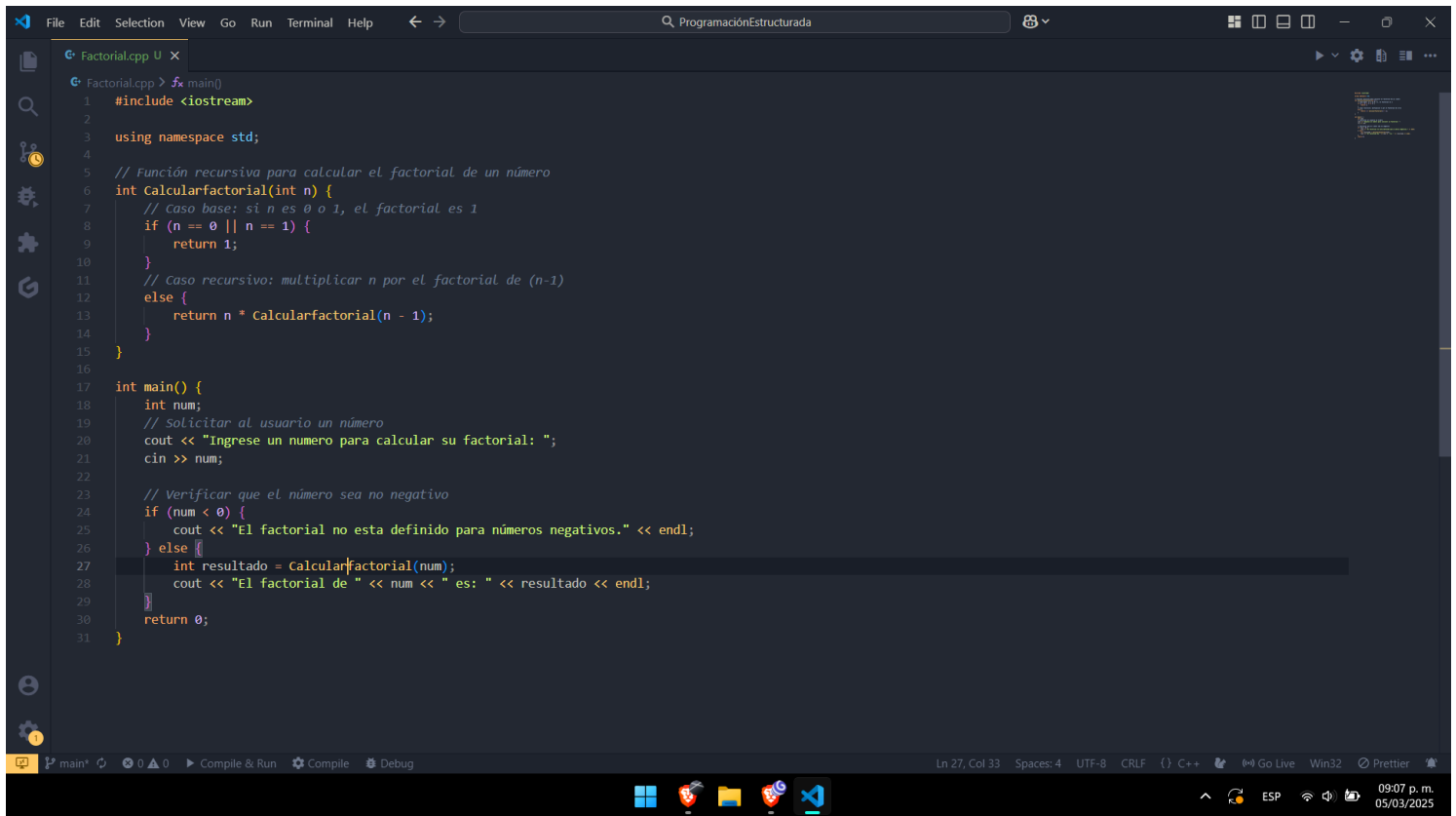
La recursividad se utiliza principalmente en situaciones donde los problemas pueden descomponerse en subproblemas más pequeños del mismo tipo. Algunos casos de uso incluyen:

- **División y conquista:** Algoritmos como QuickSort y MergeSort usan recursividad para ordenar listas.
- **Cálculo de secuencias matemáticas:** Como la serie de Fibonacci o el factorial de un número.
- **Búsqueda y exploración de caminos:** Algoritmos de backtracking como la resolución del problema de las Torres de Hanoi o la búsqueda en laberintos.

Diferencia entre recursividad y ciclos

Característica	Recursividad	Ciclos (for, while)
Definición	Una función se llama así misma para resolver un problema.	Se repite un bloque
Uso	Útil cuando el problema puede dividirse en subproblemas más pequeños del mismo tipo.	Se usa cuando el número de iteraciones es conocido o se basa en una condición.
Eficiencia	Puede consumir más memoria debido al uso de la pila de llamadas.	Más eficiente en términos de uso de memoria y procesamiento.
Complejidad	Puede ser más difícil de entender y depurar.	Más fácil de seguir en muchos casos.
Ejemplo típico	Cálculo del factorial, Fibonacci, recorrido de árboles	Iteraciones sobre listas o matrices, cálculos repetitivos simples.

Ejemplo de recursividad



```
Factorial.cpp U X
Factorial.cpp > fx main()
1  #include <iostream>
2
3  using namespace std;
4
5  // Función recursiva para calcular el factorial de un número
6  int Calcularfactorial(int n) {
7      // Caso base: si n es 0 o 1, el factorial es 1
8      if (n == 0 || n == 1) {
9          return 1;
10     }
11     // Caso recursivo: multiplicar n por el factorial de (n-1)
12     else {
13         return n * Calcularfactorial(n - 1);
14     }
15 }
16
17 int main() {
18     int num;
19     // Solicitar al usuario un número
20     cout << "Ingrese un numero para calcular su factorial: ";
21     cin >> num;
22
23     // Verificar que el número sea no negativo
24     if (num < 0) {
25         cout << "El factorial no esta definido para números negativos." << endl;
26     } else {
27         int resultado = Calcularfactorial(num);
28         cout << "El factorial de " << num << " es: " << resultado << endl;
29     }
30     return 0;
31 }
```

Explicación:

Si el usuario ingresa el número 5, el programa ejecutará la siguiente secuencia:

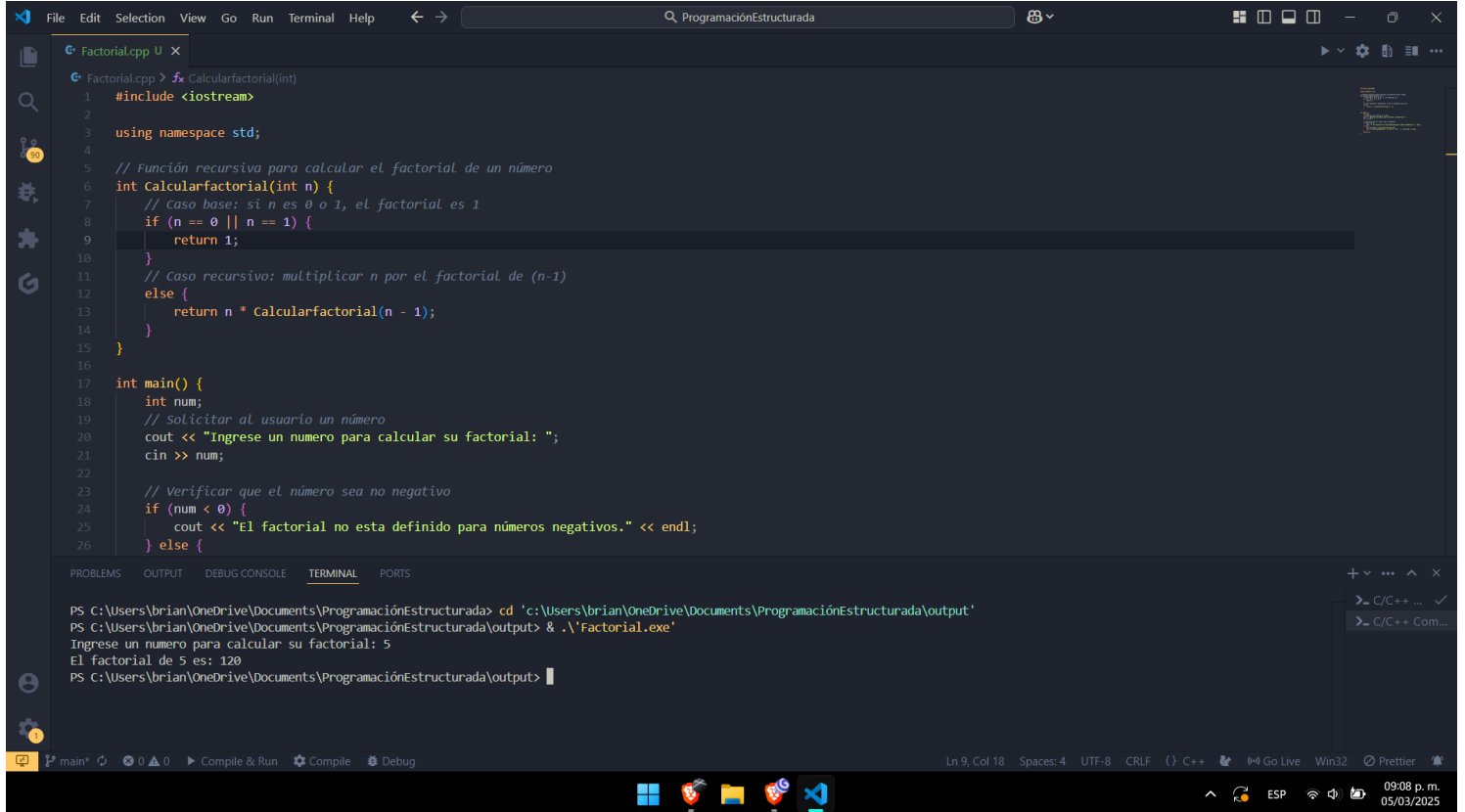
1. $\text{factorial}(5) = 5 * \text{factorial}(4)$
2. $\text{factorial}(4) = 4 * \text{factorial}(3)$
3. $\text{factorial}(3) = 3 * \text{factorial}(2)$
4. $\text{factorial}(2) = 2 * \text{factorial}(1)$
5. $\text{factorial}(1) = 1$ (caso base)

Luego la pila de llamadas se resuelve multiplicando los valores obtenidos:

- $\text{factorial}(2) = 2 * 1 = 2$
- $\text{factorial}(3) = 3 * 2 = 6$
- $\text{factorial}(4) = 4 * 6 = 24$
- $\text{factorial}(5) = 5 * 24 = 120$

Ingeniería en Tecnologías de la Información e Innovación Digital

Salida esperada con el ejemplo del número 5:



The screenshot shows a Visual Studio Code editor with a C++ file named 'Factorial.cpp'. The code implements a recursive function 'Calcularfactorial' to calculate the factorial of a number 'n'. The base case is when 'n' is 0 or 1, returning 1. The recursive case multiplies 'n' by the factorial of 'n-1'. The 'main' function prompts the user for a number, checks if it's non-negative, and then calls the recursive function. The terminal output shows the program running successfully, calculating the factorial of 5 as 120.

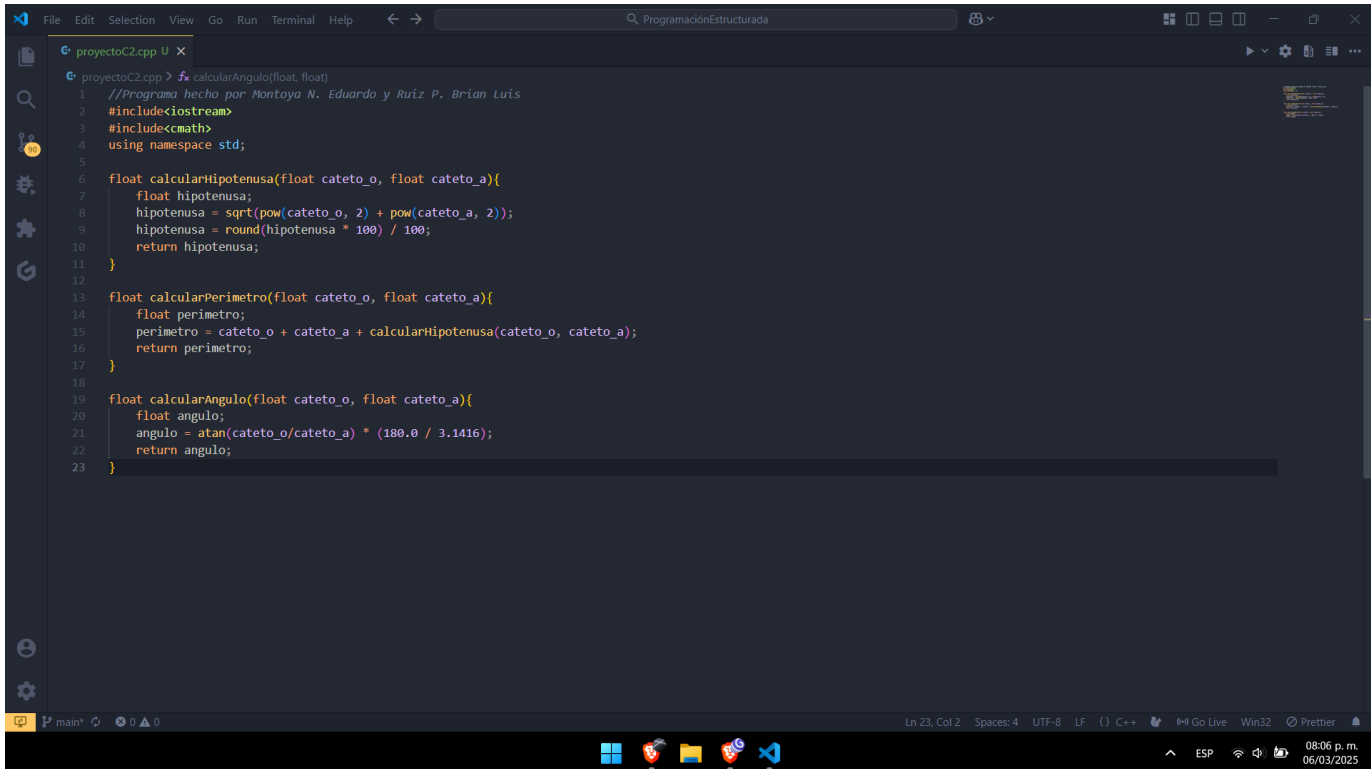
```
Factorial.cpp
1 #include <iostream>
2
3 using namespace std;
4
5 // Función recursiva para calcular el factorial de un número
6 int Calcularfactorial(int n) {
7     // Caso base: si n es 0 o 1, el factorial es 1
8     if (n == 0 || n == 1) {
9         return 1;
10    }
11    // Caso recursivo: multiplicar n por el factorial de (n-1)
12    else {
13        return n * Calcularfactorial(n - 1);
14    }
15 }
16
17 int main() {
18     int num;
19     // Solicitar al usuario un número
20     cout << "Ingrese un numero para calcular su factorial: ";
21     cin >> num;
22
23     // Verificar que el número sea no negativo
24     if (num < 0) {
25         cout << "El factorial no esta definido para números negativos." << endl;
26     } else {
```

```
PS C:\Users\brian\OneDrive\Documents\ProgramaciónEstructurada> cd 'c:\Users\brian\OneDrive\Documents\ProgramaciónEstructurada\output'
PS C:\Users\brian\OneDrive\Documents\ProgramaciónEstructurada\output> & .\Factorial.exe
Ingrese un numero para calcular su factorial: 5
El factorial de 5 es: 120
PS C:\Users\brian\OneDrive\Documents\ProgramaciónEstructurada\output>
```

Conclusión

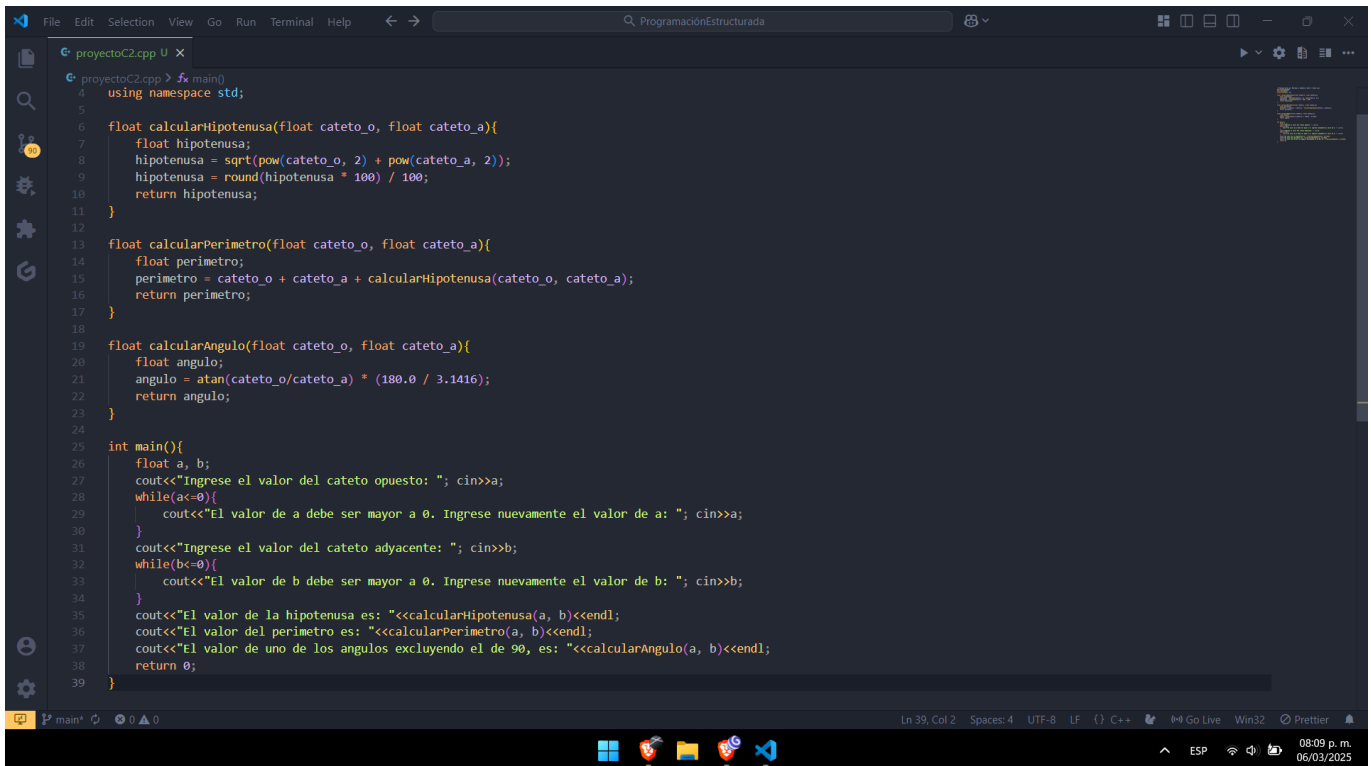
Este programa demuestra el uso de la recursividad en C++. La función `Calcularfactorial()` se llama así misma hasta llegar al caso base (`n == 1`), lo que evita una recursión infinita. Además, el programa incluye una validación para evitar números negativos.

Uso de recursividad en el proyecto del corte 1:



```
File Edit Selection View Go Run Terminal Help
proyectoC2.cpp
1 //Programa hecho por Montoya N. Eduardo y Ruiz P. Brian Luis
2 #include<iostream>
3 #include<cmath>
4 using namespace std;
5
6 float calcularHipotenusa(float cateto_o, float cateto_a){
7     float hipotenusa;
8     hipotenusa = sqrt(pow(cateto_o, 2) + pow(cateto_a, 2));
9     hipotenusa = round(hipotenusa * 100) / 100;
10    return hipotenusa;
11 }
12
13 float calcularPerimetro(float cateto_o, float cateto_a){
14     float perimetro;
15     perimetro = cateto_o + cateto_a + calcularHipotenusa(cateto_o, cateto_a);
16     return perimetro;
17 }
18
19 float calcularAngulo(float cateto_o, float cateto_a){
20     float angulo;
21     angulo = atan(cateto_o/cateto_a) * (180.0 / 3.1416);
22     return angulo;
23 }
```

La recursividad se implementará para poder realizar los cálculos de la hipotenusa, el perímetro y el área del triángulo, las cuales se mandarán a llamar para poder realizar los cálculos según los datos que ingrese el usuario:



```
File Edit Selection View Go Run Terminal Help
proyectoC2.cpp
4 using namespace std;
5
6 float calcularHipotenusa(float cateto_o, float cateto_a){
7     float hipotenusa;
8     hipotenusa = sqrt(pow(cateto_o, 2) + pow(cateto_a, 2));
9     hipotenusa = round(hipotenusa * 100) / 100;
10    return hipotenusa;
11 }
12
13 float calcularPerimetro(float cateto_o, float cateto_a){
14     float perimetro;
15     perimetro = cateto_o + cateto_a + calcularHipotenusa(cateto_o, cateto_a);
16     return perimetro;
17 }
18
19 float calcularAngulo(float cateto_o, float cateto_a){
20     float angulo;
21     angulo = atan(cateto_o/cateto_a) * (180.0 / 3.1416);
22     return angulo;
23 }
24
25 int main(){
26     float a, b;
27     cout<<"Ingrese el valor del cateto opuesto: "; cin>>a;
28     while(a<=0){
29         cout<<"El valor de a debe ser mayor a 0. Ingrese nuevamente el valor de a: "; cin>>a;
30     }
31     cout<<"Ingrese el valor del cateto adyacente: "; cin>>b;
32     while(b<=0){
33         cout<<"El valor de b debe ser mayor a 0. Ingrese nuevamente el valor de b: "; cin>>b;
34     }
35     cout<<"El valor de la hipotenusa es: "<<calcularHipotenusa(a, b)<<endl;
36     cout<<"El valor del perimetro es: "<<calcularPerimetro(a, b)<<endl;
37     cout<<"El valor de uno de los angulos excluyendo el de 90, es: "<<calcularAngulo(a, b)<<endl;
38     return 0;
39 }
```