Informática II

Recursividad

¿Qué es recursividad?

- Que puede repetirse.
- La recursividad es un concepto fundamental en matemáticas y en computación.
- Es una alternativa diferente para implementar estructuras de repetición (ciclos). Los módulos se hacen llamadas recursivas.
- Se puede usar en toda situación en la cual la solución pueda ser expresada como una secuencia de movimientos, pasos o transformaciones gobernadas por un conjunto de reglas no ambiguas.

Función recursiva

Se compone de:

- Caso base: una solución simple para un caso particular (puede haber más de un caso base). Es importante determinar un caso base, es decir un punto en el cual existe una condición por la cual no se requiera volver a llamar a la misma función.
- Caso recursivo: una solución que involucra volver a utilizar la función original, con parámetros que se acercan más al caso base. Los pasos que sigue el caso recursivo son los siguientes:
 - 1. La función se llama a sí misma
 - 2. El problema se resuelve, tratando el mismo problema pero de tamaño menor
 - 3. La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará.

Ejemplo: Calculo del factorial

☐ Calcule el factorial (!) de un entero no negativo.

0!	=1	=1	=1
1!	=1 * 0!	=1* 1	=1
2!	=2 * 1!	=2 * 1	=2
3!	=3 * 2!	=3 * 2 * 1	=6
4!	=4 * 3!	=4 * 3 * 2 * 1	=24
5!	=5 * 4!	=5 * 4 * 3 * 2 * 1	=120
N!	N * (N - 1)!		

El factorial de un número es la multiplicación de cada número desde 1 hasta ese número, entonces es muy sencillo crear un ciclo de 1 hasta el número pedido para hacer el cálculo.

Factorial (sin recursividad)

```
1 #include<iostream>
 2 #include<locale.h>
 4 using namespace std;
 5
 7⊝ int main(){
        int i,num,resultado=1;
10
        cout << "Calculo del factorial." << endl;</pre>
                                                                  Uso de la
11
        cin >> num;
                                                                 estructura
12
      for(i=1; i<=num; i++)</pre>
13
                                                               repetitiva FOR
14
            resultado = resultado * i;
15
16
        cout << "El factorial de "<<num<<" es: " << resultado;
17
        return 0;
18
19 }
```

Factorial (con recursividad)

Para el calculo de N!

$$N! = \begin{cases} si N = 0 \\ N*(N-1)! si N > 0 \end{cases}$$
 Caso base Caso Recursivo

Factorial (con recursividad)

```
#include<iostream>
   #include<locale.h>
   using namespace std;
   int calcularFactorial(int);
 8⊖ int main(){
10
     int num=0;
11
     cout<<"Introduce un numero para calcular el factorial: ";</pre>
12
13
     cin>>num;
     cout<<"El factorial de "<<num<<" es: "<<calcularFactorial(num);</pre>
14
15
     return 0;
16
17 }
     if(num==0)
                                                 Caso base
22
23
     return 1;
24
25
     else
26
                                                              Se llama así misma
27
28
                                                   Caso Recursivo
```

¿Por qué usar recursividad?

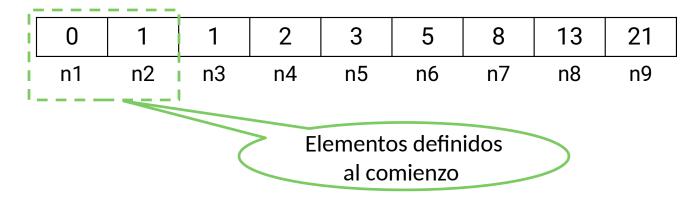
- Son mas cercanos a la descripción matemática.
- Permiten simplificar el código.
- Generalmente mas fáciles de analizar.
- Se adaptan mejor a las estructuras de datos recursivas.
- Los algoritmos recursivos ofrecen soluciones estructuradas, modulares y elegantemente simples.

¿Cuando no usar recursividad?

- Cuando las funciones usen arreglos largos.
- Cuando las funciones cambian de manera impredecible.
- U Cuando las iteraciones sean la mejor opción.

Ejemplo: Calculo de la Serie de Fibonacci

Los valores son:



Omienza con un 0, luego un 1 y a partir de ahí cada término de la serie suma los 2 anteriores. Fórmula recursiva:

Ejemplo: Calculo de la Serie de Fibonacci

```
#include<iostream>
   #include<locale.h>
   using namespace std;
   #define n1 0
   #define n2 1
   int calcularFibonacci(int);
10
11⊖ int main(){
12
13
     int num=0, i;
     cout<<"Introduce cuantos numeros de la serie Fibonacci desea ver: ":</pre>
14
15
     cin>>num;
      for(i=1;i<=num;i++)</pre>
16
17
          cout<<calcularFibonacci(i)<<" ";</pre>
18
19
20
      return 0;
21
22 }
23
24@int calcularFibonacci(int num)
25 {
                                            Casos bases
       if (num==1)
26
                                                                                  Caso
            return n1;
27
28
                                                                               Recursivo
       else if (num==2)
29
            return n2;
30
31
32
        else
            return calcularFibonacci(num-1) + calcularFibonacci(num-2);
33
34 }
```

Recursión vs iteración

Repetición

- √ Iteración: ciclo explícito (se expresa claramente)
- ✓ Recursión: repetidas invocaciones a una función

Terminación

- ✓ Iteración: el ciclo termina o la condición del ciclo falla
- ✓ Recursión: se reconoce el o los casos bases

En ambos casos podemos tener ciclos infinitos