

RECURSIVIDAD

RECUERDA

Algunos de los problemas más habituales de los métodos recursivos son los siguientes:

- No finalización del método: puede ocurrir que siempre, o sólo en el caso de determinados parámetros de entrada, el método recursivo no alcance nunca un caso no recursivo y, por lo tanto, no finalice.
- Desbordamiento de pila: cuanto mayor sea el número de invocaciones anidadas en un momento dado, mayor es el tamaño que ocupa la pila, porque necesita almacenar más datos. Existe el riesgo de que, para determinados parámetros, se sobrepase el tamaño máximo de esta pila, lo cual desemboca en una cancelación abrupta del programa (“stack overflow”).
- Cálculo repetido de los mismos datos: un algoritmo recursivo mal programado puede dar lugar a que se realice muchas veces el mismo cálculo, provocando que el número de invocaciones recursivas realizadas crezca exponencialmente. Por ejemplo, la sucesión de Fibonacci.

EJERCICIOS

Escoge diez de los siguientes ejercicios, y ponte manos a la obra. Tendrás un punto por cada ejercicio resuelto correctamente.

1. ¿Cuál es el resultado del siguiente programa si llamo al procedimiento *metodo(6)*? Dibuja las llamadas.

```
void metodo(int n) {
    if (n<2)
        printf("X");
    else {
        metodo(n-1);
        printf("O");
    }
}
```

2. ¿Cuál es el resultado del siguiente programa si llamo al procedimiento *metodo1(6)*? ¿Y si llamo al procedimiento *metodo1(7)*? Dibuja las llamadas.

```
void metodo1(int n) {
    if (n == 0)
        printf("En metodo 1 con N: %d\n", n);
    else
        metodo2(n);
}

void metodo2(int n) {
    printf("En metodo 2 con N: %d\n", n);
    metodo3(n-1);
}

void metodo3(int n) {
    printf("En metodo 3 con N: %d\n", n);
    metodo1(n-1);
}
```

1. Determina qué calcula la siguiente función recursiva. Escribe una función iterativa que realice la misma tarea.

```
int func(int n) {
    if (n == 0)
        return(0);
    else
        return( n+func(n-1) );
}
```

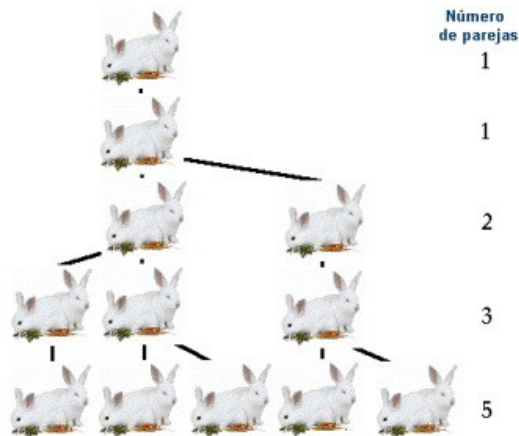
2. Programa una función recursiva y otra iterativa para calcular el máximo común divisor de dos números enteros aplicando las siguientes propiedades recurrentes:

$\text{mcd}(a, b) = \text{mcd}(a-b, b)$ si $a > b$
 $\text{mcd}(a, b) = \text{mcd}(a, b-a)$ si $a < b$
 $\text{mcd}(a, b) = a$ si $a = b$

3. Calcula el número de llamadas que se generan para calcular el número de Fibonacci mediante el algoritmo recursivo básico, para un cierto número n , pasado por parámetro. Programa el método de cálculo de la sucesión de Fibonacci con un algoritmo iterativo. (Si el método iterativo ya estuviera visto en clase, programa un método recursivo que, para evitar cálculos repetidos, aceptase como parámetro un vector de resultados.)

$\text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2)$, para $n > 1$
 $\text{Fib}(n) = 1$, para $n = 1$ ó $n = 0$

La famosa serie de Fibonacci responde a la siguiente pregunta: “Una pareja de conejos tarda un mes en alcanzar la edad fértil, a partir de ese momento cada vez engendra una pareja de conejos, que a su vez, tras ser fértiles engendrarán cada mes una pareja de conejos. ¿Cuántos conejos habrá al cabo de un determinado número de meses?”.



4. Programa un método recursivo que transforme un número entero positivo a notación binaria.
5. Programa un método recursivo que transforme un número expresado en notación binaria a número entero.

6. Programa un método recursivo para calcular la integral de una función (por ejemplo, el $\sin(x)$) en un intervalo dado, dividiendo este intervalo en subintervalos de longitud no menor que un determinado valor e :

$$\text{si } b-a \geq e \quad \int_a^b f(x)dx = \int_a^m f(x)dx + \int_m^b f(x)dx \quad \text{donde } m = \frac{a+b}{2}$$

$$\text{si } b-a < e \quad \int_a^b f(x)dx \cong (b-a)f(m) \quad \text{donde } m = \frac{a+b}{2}$$

7. Programa un método recursivo que calcule la suma de un vector de números enteros.
8. Programa un método recursivo que invierta el orden de un vector de números enteros.
9. Programa un método que realice búsqueda binaria recursivamente en un vector de números enteros ordenado de menor a mayor.
10. Diseña la versión iterativa del siguiente algoritmo recursivo:

```
void rec(int n) {  
    if (!f(n)) {  
        /* cualquier grupo de sentencias que no modifiquen el valor de n */  
        rec(g(n));  
    }  
}
```

donde las cabeceras de f y g se definen como:

```
boolean f(int n);  
int g(int n);
```

11. Considera la siguiente función recursiva:

```
int p(int x) {  
    if (x < 3)  
        return(x);  
    else  
        return( p(x-1)*p(x-3) );  
}
```

Sea $A(n)$ el número de productos realizados al ejecutar la función p cuando se llama con el argumento n . Diseña la función recursiva de $A(n)$.

12. Dada la función recursiva:

```
int f(int x) {  
    if (x > 100)  
        return( x-10 );  
    else  
        return( f(f(x+11)) );  
}
```

Estudia cuál es su comportamiento. ¿Podrías diseñar f de una manera más sencilla?

13. Considera la siguiente función recursiva:

$$\begin{aligned} \text{Acker}(m, n) &= n + 1 \quad \text{si } m = 0 \\ \text{Acker}(m, n) &= \text{Acker}(m - 1, 1) \quad \text{si } n = 0 \\ \text{Acker}(m, n) &= \text{Acker}(m - 1, \text{Acker}(m, n - 1)) \quad \text{en otro caso} \end{aligned}$$

Esta función, llamada función de Ackermann, es interesante porque crece rápidamente con respecto a los valores m y n . ¿Qué vale $\text{Acker}(1, 2)$? ¿Cuántas llamadas recursivas se hacen a la función de Ackermann cuando queremos evaluar $\text{Acker}(1, 2)$?

14. Escribe un algoritmo recursivo que ordene un vector siguiendo el método “Mergesort”:
 - a) Sea k el índice del elemento mitad del vector.
 - b) Ordena los elementos hasta $a[k]$, incluyéndolo.
 - c) Ordena los elementos siguientes.
 - d) Mezcla los dos subvectores en un único vector ordenado.

15. Diseña un algoritmo recursivo que, dado un vector de enteros, devuelva el k -ésimo elemento más pequeño del vector, es decir, si el vector contiene los elementos (4, 7, 3, 6, 8, 1, 9, 2) y $k=5$, el algoritmo debería devolver 6. (Pista: estudia el método de ordenación “Quicksort”.)

16. Consideremos palíndromos sobre un alfabeto formado sólo por letras minúsculas. Sea $P(n)$ el número de palíndromos de longitud n . Diseña una función recursiva para $P(n)$.

17. Diseña un algoritmo recursivo que lea una secuencia de caracteres de longitud arbitraria terminada en un punto, e imprima la suma de todos los dígitos junto con la sucesión en orden inverso de entrada.

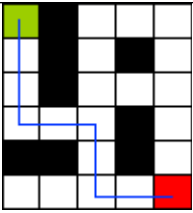
18. Diseña tres algoritmos recursivos para determinar si una cadena de caracteres (que se les pasa como parámetro) es:
 - a) un identificador válido.
 - b) un palíndromo.
 - c) una palabra perteneciente al lenguaje $L = \{w / w = a^n b^n, n \geq 0\}$

EJERCICIOS EXTRA

Cuando hayas acabado los diez ejercicios anteriores, si resuelves alguno de los siguientes ejercicios, tendrás una nota extra de 10.

19. Imaginad un laberinto formado por una matriz cuadrada de dimensiones nm , con k celdas marcadas que no se pueden atravesar. Diseñad un programa que encuentre el camino más corto para viajar de la esquina (1,1) a la esquina (n, m) sin pasar por las celdas marcadas.
 El usuario introducirá los valores de n , de m y de k , así como las coordenadas de las k celdas marcadas. La salida serán las coordenadas de las celdas que forman el camino (tened en cuenta que puede no haber camino o haber más de uno).

Por ejemplo:

 <p>Laberinto</p>	<p><u>Entrada</u></p> <p>6 5 8</p> <p>1 2</p> <p>2 2</p> <p>2 4</p> <p>3 2</p> <p>4 4</p> <p>5 1</p> <p>5 2</p> <p>5 4</p>	<p><u>Salida</u></p> <p>1 1</p> <p>2 1</p> <p>3 1</p> <p>4 1</p> <p>4 2</p> <p>4 3</p> <p>5 3</p> <p>6 3</p> <p>6 4</p> <p>6 5</p>
----------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------