ESTRUCTURAS DE DATOS

LIC. REDES Y SERVICIOS DE CÓMPUTO

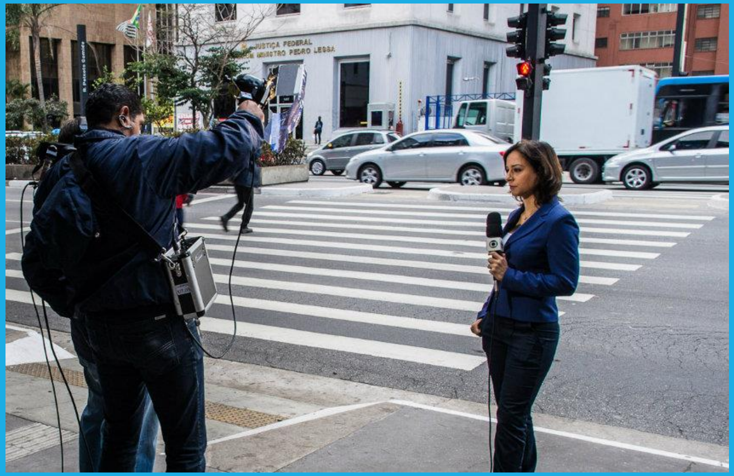
Clase 2

- Recursión

Recursión

Recursión

¿CÓMO LE HAGO?
HECHOS
6:26 JUE 15
Armando Contreras González
Azteca Noticias



POLICÍA
POLICÍA

¿CÓMO LE HAGO?
HECHOS
6:26 JUE 15
Armando Contreras González
Azteca Noticias

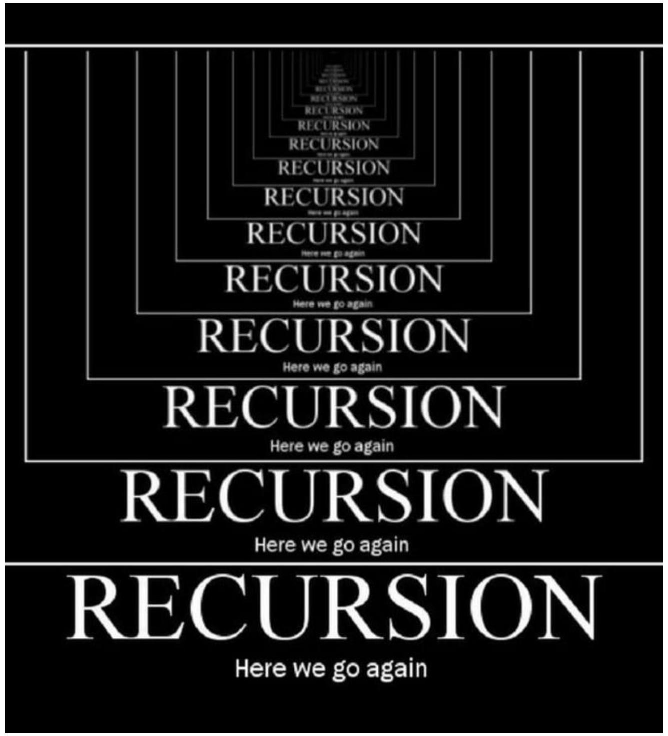
POLICÍA
POLICÍA

Recursión

Introducción

- Es un recurso poderoso para expresar soluciones simples y naturales a ciertos problemas.

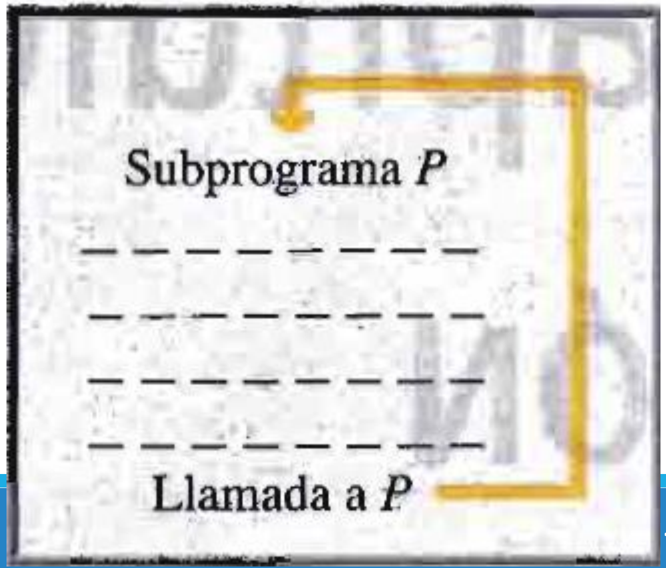
- Algunos problemas son naturalmente recursivos, otros no.



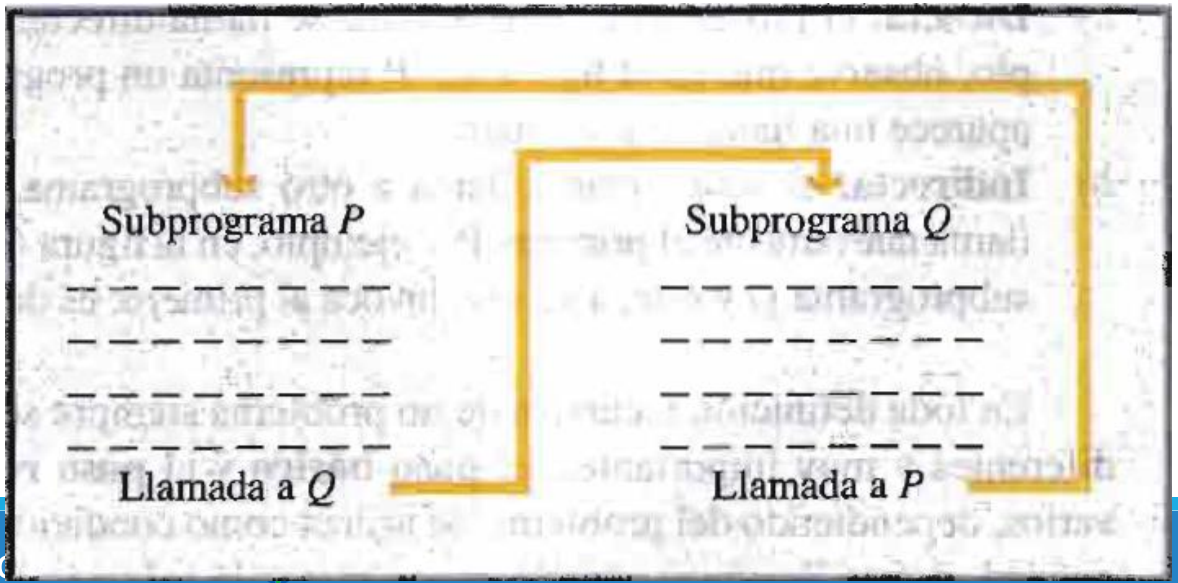
Recursión

Un objeto recursivo es aquel que aparece en la definición de sí mismo, así como el que se llama a sí mismo.

Recursión Directa: El programa o subprograma se llama directamente a sí mismo.



Recursión Indirecta: El subprograma llama a otro subprograma, y el segundo, en algún momento llama nuevamente al primero.



Recursión

En toda definición recursiva de un problema se deben establecer dos pasos:

- Paso Básico
- Paso Recursivo

Paso Básico

- Pueden ser uno o varios, dependiendo del problema.
- Se utiliza como condición de parada o fin de la recursividad.
- Se llega cuando se encuentra la solución del problema o cuando se decide que no se continuará porque no se tienen las condiciones para hacerlo.

Paso Recursivo

- Propicia la recursividad.
- Pueden presentarse uno o varios, dependiendo del problema.

Recursión

- Es muy importante definir con precisión los pasos básicos y recursivos.
- En cada vuelta del ciclo es importante que nos acerquemos cada vez más a la solución del problema (al paso básico). De otra forma esteremos en un ciclo extraño, con un problema mal definido.
- La computadora se quedaría ejecutando por tiempo indefinido el programa, hasta agotarse la memoria.

Recursión

Ejemplos

Factorial de un número

Sucesión de Fibonacci

Impresión de un arreglo

Suma de un arreglo

Recursión

Ejemplo. Factorial de un número.

¿Qué es el Factorial de un número?

El factorial de un número entero positivo **n** se define como el producto de los números comprendidos entre **1** y ***n***. También como **n** por el factorial de (***n* – 1**).

A continuación se ilustra el ejemplo:

La expresión **!*n*** simbolizará el factorial de ***n***.

Recursión

Ejemplo. Factorial de un número.

¿Qué es el Factorial de un número?

El factorial de un número entero positivo = n por el factorial de (n – 1).

Así, el factorial de 3 se define como:

3! = 3 * 2!

Lo que es igual a:

3 * 2 * 1

Y el factorial de 4 se define como:

4! = 4 * 3!

Lo que es igual a:

4 * 3 * 2 * 1

Recursión

Ejemplo. Factorial de un número.

El factorial de 5 se define como:

5! = 5 * 4!

Lo que es igual a:

5 * 4 * 3 * 2 * 1

….

…

..

n! = n * (n – 1)! = n * (n – 1)! * ….* 4 * 3 * 2 * 1

Así, tenemos que para calcular el factorial de 5 decimos que 5! Es igual a 5 * 4!

Y el factorial de 4 se calcula 4 * 3!

El factorial de 3, se calcula 3 * 2!

Y así sucesivamente hasta llegar a un paso básico que detiene la recursividad.

Definiendo el factorial de un número *n*, en términos del factorial del número (n – 1)!

Recursión

Ejemplo. Factorial de un número.

Por definición:

0! = 1 → Paso Básico

1! = 1 → Paso Básico

n! = n * (n – 1)! = n * (n – 1)! * ….* 4 * 3 * 2 * 1 → Paso Recursivo

Se deduce la fórmula:

$$n! \begin{cases} 1 & \text{Si } n = 0 \text{ o } n = 1 \quad \textbf{Paso básico} \\ n * (n-1)! & \text{Si } n > 1 \quad \textbf{Paso recursivo} \end{cases}$$

Recursión

Ejemplo. Factorial de un número.

Y el algoritmo para calcularlo es el siguiente:

- Calcula el factorial de un número *N* en forma recursiva, donde *N* es un valor numérico entero, positivo o nulo.

Factorial_rec (*N*)

1. *Si* ($N = 0$)

    entonces

        Hacer Factorial_rec ← 1 {Paso básico}

    si no

        Hacer Factorial_rec ← *N* * Factorial_rec ($N - 1$)

            {Llamada —paso— recursiva}

2. {Fin del condicional del paso 1}

Recursión

Ejemplo. Factorial de un número.

Factorial_rec (*N*)

1. *Si* $(N = 0)$
   entonces
   Hacer Factorial_rec ← 1 {Paso básico}
   si no
   Hacer Factorial_rec ← N * Factorial_rec $(N - 1)$
   {Llamada —paso— recursiva}
2. {Fin del condicional del paso 1}

		Factorial rec (*N*)		
Valor inicial	→	5		
[1]	→	4		
[2]	→	3		
[3]	→	2		
[4]	→	1		
[5]	→	0	→	1

		Pila			
1 = 1 * 1	[5]	1 * Factorial_rec	(0)	←	1
2 = 2 * 1	[4]	2 * Factorial_rec	(1)	←	1
6 = 3 * 2	[3]	3 * Factorial_rec	(2)	←	2
24 = 4 * 6	[2]	4 * Factorial_rec	(3)	←	6
120 = 5 * 24	[1]	5 * Factorial_rec	(4)	←	24

Utiliza una pila, donde se van almacenando las instrucciones a ejecutar con los valores de las constantes y las variables en ese momento.
Cuando se termina la ejecución se llega al estado básico, se toma la instrucción del tope de la pila y se continua ejecutando; hasta que la pila queda vacía.

```cpp
/* Funci[on recursiva Factorial */
#include <iostream>
#include <iomanip>
using namespace std;

unsigned long factorial( unsigned long ); // prototipo de función

int main()
{
    // calcula los factoriales del 0 al 10
    for ( int contador = 0; contador <= 10; contador++ )
        cout << setw( 2 ) << contador << "! = " << factorial( contador )
            << endl;

    return 0; // indica que terminó correctamente
} // fin de main

// definición recursiva de la función factorial
unsigned long factorial( unsigned long numero )
{
    if ( numero <= 1 ) // evalúa el caso base
        return 1; // casos base: 0! = 1 y 1! = 1
    else // paso recursivo
        return numero * factorial( numero - 1 );
} // fin de la función factorial
```

C:\Users\Mamß\Documents\Ago21-Ene22\EstructurasDeDatos\Ejercicios\Recursion\factorialRecursion.exe

```
 0! = 1
 1! = 1
 2! = 2
 3! = 6
 4! = 24
 5! = 120
 6! = 720
 7! = 5040
 8! = 40320
 9! = 362880
10! = 3628800

--------------------------------
Process exited after 0.05313 seconds with return value 0
Presione una tecla para continuar . . .
```

Recursión

Actividad.

Elabora el algoritmo, estableciendo el paso básico y el paso recursivo, para calcular un número Fibonacci N en forma recursiva. Donde N es un número entero positivo o nulo.

Después de establecer el paso básico y el paso recursivo, desarrolla el programa en C++.

La serie de Fibonacci,

0, 1, 2, 3, 4, 5, 6, 7, 8 →**N**

0, 1, 1, 2, 3, 5, 8, 13, 21, …→**Fibonacci correspondiente**

empieza con 0 y 1, y tiene la propiedad de que cada número subsiguiente de Fibonacci es la suma de los dos números Fibonacci anteriores.

Recursión

Actividad.

El Fibonacci de un número se obtiene de la suma de los dos números Fibonacci anteriores:

$$
\begin{aligned}
&\text{Fibonacci}(0) = 0 && \rightarrow \textbf{Paso básico}\\
&\text{Fibonacci}(1) = 1 && \rightarrow \textbf{Paso básico}\\
&\text{Fibonacci}(2) = \text{Fibonacci}(1) + \text{Fibonacci}(0)\\
&\qquad\qquad\quad = 1 + 0 = 1\\
&\text{Fibonacci}(3) = \text{Fibonacci}(2) + \text{Fibonacci}(1)\\
&\qquad\qquad\quad = 1 + 1 = 2\\
&\text{Fibonacci}(4) = \text{Fibonacci}(3) + \text{Fibonacci}(2)\\
&\qquad\qquad\quad = 2 + 1 = 3\\
&\ldots\\
&\text{Fibonacci}(n) = \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2) && \rightarrow \textbf{Paso recursivo}
\end{aligned}
$$

Recursión

Actividad.

El Fibonacci de un número se obtiene de la suma de los dos números Fibonacci anteriores:

$$\text{Fibonacci}(n) = \begin{cases} n & \text{si} \quad (n = 0) \text{ o } (n = 1 \\ \text{Fibonacci } (n-1) + \text{Fibonacci } (n-2) & \text{si} \quad n > 1 \end{cases}$$

El paso básico se presenta cuando n=0 o n=1.

El paso recursivo utiliza el concepto de Fibonacci aplicado a (n - 1) y (n - 0)

Recursión

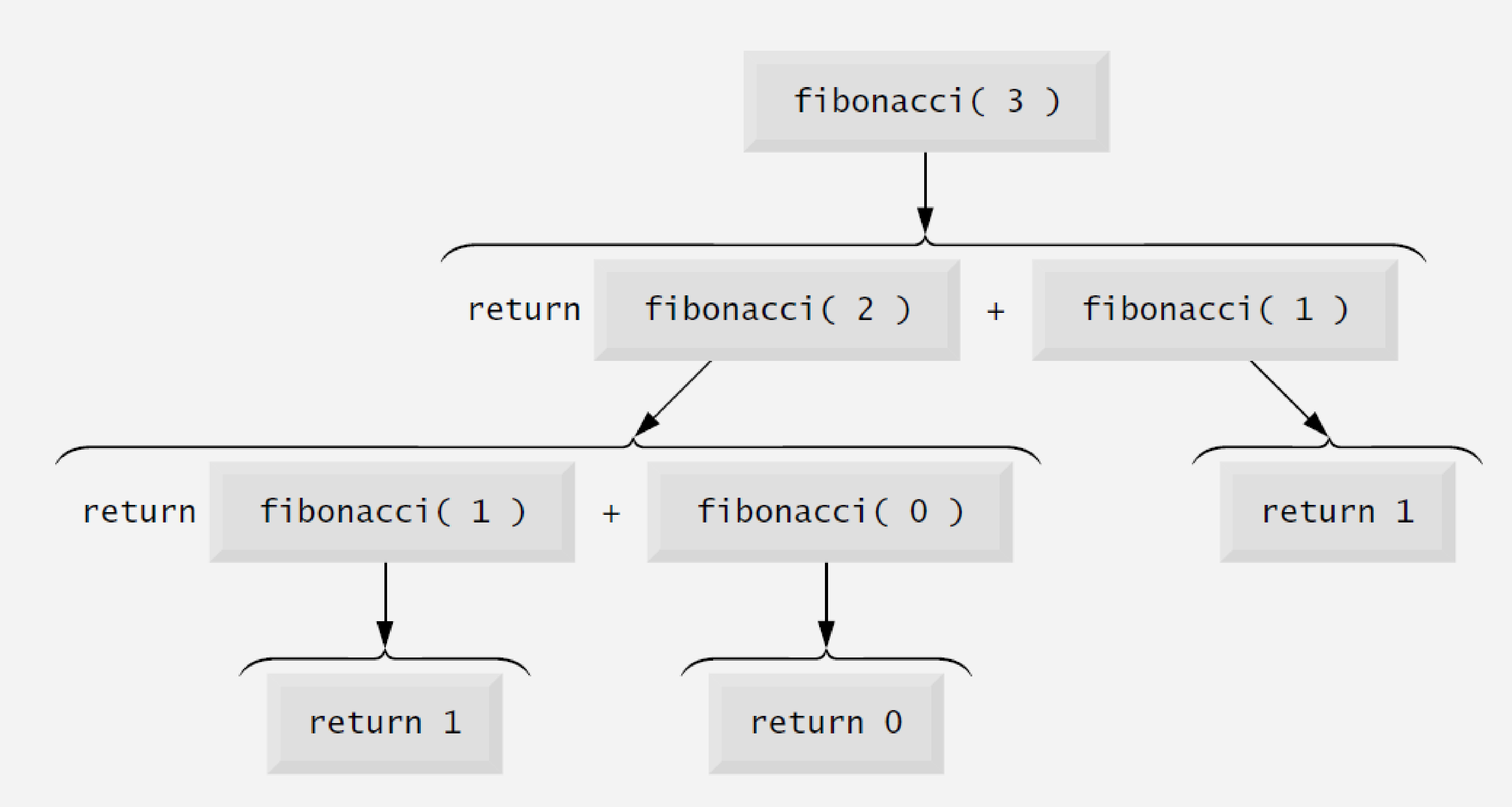
Actividad.

Fibonacci_rec (N)

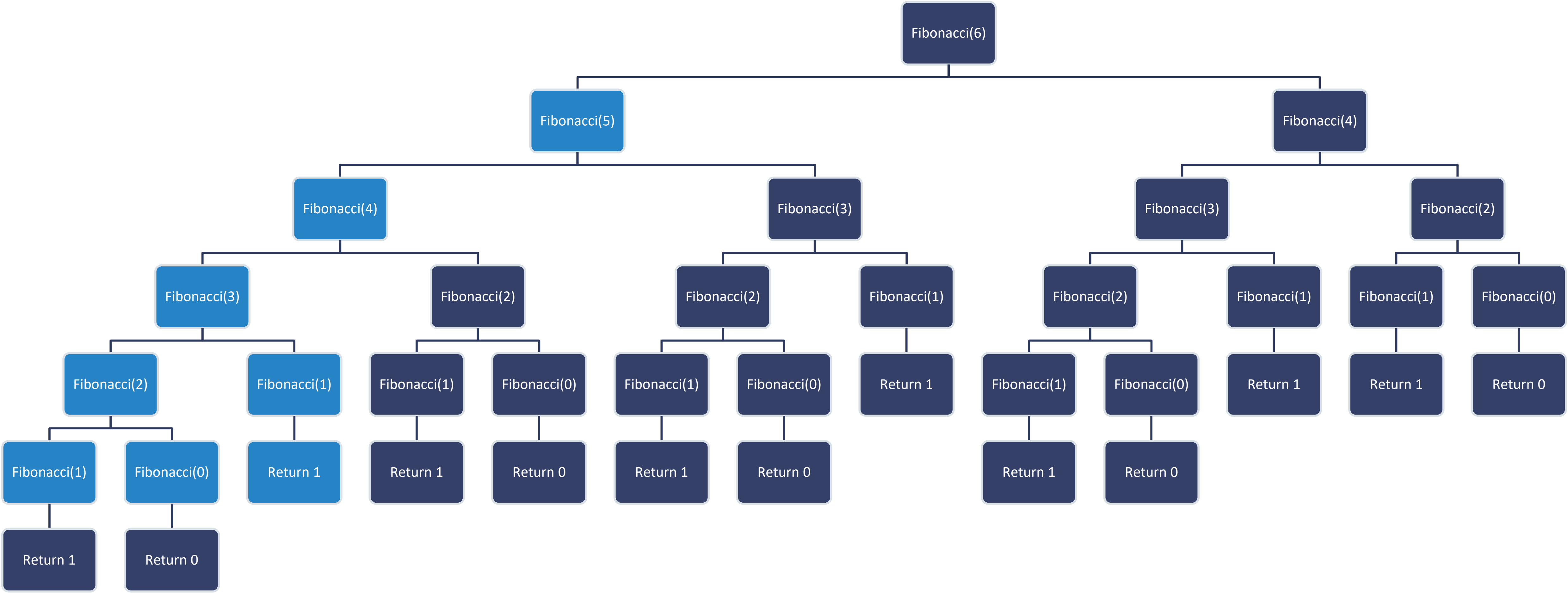
Calcula el número Fibonacci correspondiente a N de forma recursiva. Donde N es un valor numérico, entero, positivo o nulo.

1. *Si* (($N = 0$) o ($N = 1$))
    entonces
        Hacer Fibonacci_rec ← N {Paso básico}
    si no
        Hacer Fibonacci_rec ← Fibonacci_rec($N - 1$) + Fibonacci_rec($N - 2$)
            {Llamadas recursivas}
2. {Fin del condicional del paso 1}

Recursión


fibonacci(3)
return fibonacci(2) + fibonacci(1)
return fibonacci(1) + fibonacci(0)
return 1
return 1
return 0

Recursión


Fibonacci(6)
Fibonacci(5)
Fibonacci(4)
Fibonacci(4)
Fibonacci(3)
Fibonacci(3)
Fibonacci(2)
Fibonacci(3)
Fibonacci(2)
Fibonacci(2)
Fibonacci(1)
Fibonacci(2)
Fibonacci(1)
Fibonacci(1)
Fibonacci(0)
Fibonacci(2)
Fibonacci(1)
Fibonacci(1)
Fibonacci(0)
Fibonacci(1)
Fibonacci(0)
Return 1
Fibonacci(1)
Fibonacci(0)
Return 1
Return 1
Return 0
Fibonacci(1)
Fibonacci(0)
Return 1
Return 1
Return 0
Return 1
Return 0
Return 1
Return 0
Return 1
Return 0

```cpp
/* Funci[on recursiva Fibonacci */
#include <iostream>

using namespace std;

unsigned long fibonacci( unsigned long ); // prototipo de función

int main()
{
    // calcula los valores de fibonacci del 0 al 10
    for ( int contador = 0; contador <= 10; contador++ )
        cout << "fibonacci( " << contador << " ) = "
            << fibonacci( contador ) << endl;

        // muestra valores de fibonacci mayores
        cout << "fibonacci( 20 ) = " << fibonacci( 20 ) << endl;
        cout << "fibonacci( 30 ) = " << fibonacci( 30 ) << endl;
        cout << "fibonacci( 35 ) = " << fibonacci( 35 ) << endl;
        return 0; // indica que terminó correctamente
} // fin de main

// método fibonacci recursivo
unsigned long fibonacci( unsigned long numero )
{
    if ( ( numero == 0 ) || ( numero == 1 ) ) // casos base
        return numero;
    else // paso recursivo
        return fibonacci( numero - 1 ) + fibonacci( numero - 2 );
} // fin de la función fibonacci
```

Recursión

Actividad.

Elabora el algoritmo, estableciendo el paso básico y el paso recursivo, para escribir el contenido de las casillas de un arreglo de izquierda a derecha.

Posteriormente, desarrolla el programa en C++.

Recursión

Actividad.

Escribe el contenido de un arreglo A de tipo entero y tamaño N de izquierda a derecha.

1. *Si* ($N \neq 0$) *entonces*
   Arreglo_imp (A, $N - 1$)
   Escribir $A[N]$
2. {Fin del condicional del paso 1}

Recursión

Actividad.

Escribe el contenido de un arreglo A de tipo entero y tamaño N de izquierda a derecha.

A	8	12	21	27	31	44	48

1. *Si* ($N \neq 0$) *entonces*
   Arreglo_imp (A, $N-1$)
   Escribir $A[N]$
2. {Fin del condicional del paso 1}

Arreglo_imp (A, N)	Impresión de resultados	
A 7	8	E [1]
A 6	12	E [2]
A 5	21	E [3]
A 4	27	E [4]
A 3	31	E [5]
A 2	44	E [6]
A 1	48	E [7]
A 0		

Pila
Escribir A [1]
Escribir A [2]
Escribir A [3]
Escribir A [4]
Escribir A [5]
Escribir A [6]
Escribir A [7]

```cpp
#include <iostream>
using namespace std;

// funcion recursiva
void Arreglo_imp( int A[], int N )
{
    if( N != 0 )
    {
        // Usa N-1 para acceder al valor del arreglo
        // porque en C++ los arreglos empiezan en 0
        Arreglo_imp(A, N-1);
        // imprime la posicion y el valor del elemento en el arreglo
        cout << "A[" << N-1 << "] = " << A[N-1] << endl ;
    }
}

int main()
{
    int N = 10;
    int A[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    // funcion recursiva
    Arreglo_imp( A, N );

    return 0;
}
```

Recursión

Actividad.

Elabora el algoritmo, estableciendo el paso básico y el paso recursivo, para sumar los elementos de un arreglo.

Posteriormente, desarrolla el programa en C++.

Recursión

Actividad.

Suma los elementos de un arreglo
A de tipo entero y tamaño N.

1. *Si* $(N = 0)$

    entonces

        Hacer Arreglo_sum $\leftarrow 0$

    si no

        Hacer Arreglo_sum $\leftarrow A[N]$ + Arreglo_sum$(A, N - 1)$

2. {Fin del condicional del paso 1}

```cpp
#include <iostream>
using namespace std;

// funcion recursiva
int Arreglo_sum( int A[], int N )
{
    if( N == 0 )
    {
        return 0;
    }
    else
    {
        // Usa N-1 para acceder al valor del arreglo
        // porque en C++ los arreglos empiezan en 0
        return A[N-1] + Arreglo_sum( A, N-1 );
    }
}

int main()
{
    int N = 10;
    // La suma de los valores es 55
    int A[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    // funcion recursiva
    cout << "Suma de los valores del arreglo: " << Arreglo_sum(A, N);

    return 0;
}
```

1. *Si* (N = 0)

    entonces

        Hacer Arreglo_sum ← 0

    si no

        Hacer Arreglo_sum ← $A[N]$ + Arreglo_sum(A, N – 1)

2. {Fin del condicional del paso 1}

A	8	12	21	27	31	44	48

Arreglo_sum	(A, N)
	A 7
[1]	A 6
[2]	A 5
[3]	A 4
[4]	A 3
[5]	A 2
[6]	A 1
[7]	A 0 → 0

8 + 0 = 8

12 + 8 = 20

21 + 20 = 41

27 + 41 = 68

31 + 68 = 99

44 + 99 = 143

48 + 143 = 191

Pila	
$A[1]$ + Arreglo_sum (A, 0)	[7]
$A[2]$ + Arreglo_sum (A, 1)	[6]
$A[3]$ + Arreglo_sum (A, 2)	[5]
$A[4]$ + Arreglo_sum (A, 3)	[4]
$A[5]$ + Arreglo_sum (A, 4)	[3]
$A[6]$ + Arreglo_sum (A, 5)	[2]
$A[7]$ + Arreglo_sum (A, 6)	[1]

Recursión

Mayor común divisor recursivo

El mayor común divisor, mcd, de los enteros *x* y *y* es el entero más grande que se puede dividir entre x y y de manera uniforme.

El algoritmo de Euclides es un método eficaz para encontrar el mcd entre dos números enteros positivos. El algoritmo consiste en varias divisiones euclidianas sucesivas.

En la primera división se toma como dividendo el mayor de los números y como divisor el otro.

Luego, el divisor y el resto sirven de dividendo y divisor, respectivamente, de la siguiente división. El proceso se detiene cuando se obtiene un resto nulo.

Escribe una función recursiva llamada *Euclides*, que devuelva el mayor común divisor de *x* y *y*, definida mediante la recursividad, de la siguiente manera:

si *y* es igual a 0, entonces mcd(x, y) es *x*; en caso contrario,

mcd(x, y) es mcd(y, x % y), donde % es el operador módulo.

[Nota: para este algoritmo, x debe ser mayor que y.]

Recursión

Mayor común divisor recursivo

El máximo común divisor (MCD) de dos enteros A y B es el entero más grande que divide tanto a A como a B.

El algoritmo de Euclides es una técnica para encontrar rápidamente el MCD de dos enteros.

Encontrar el MCD de 270 y 192.

A=270, B=192.

División para encontrar que 270/192 = 1 con un residuo de 78.

MCD(192,78)

Mayor común divisor recursivo

MCD(192,78)

A=192, B=78.

División para encontrar que 192/78 = 2 con un residuo de 36.

MCD(78,36),

A=78, B=36.

División para encontrar que 78/36 = 2 con un residuo de 6.

MCD(36,6),

A=36, B=6.

División larga para encontrar que 36/6 = 6 con un residuo de 0.

A=6, B=0.

A ≠0

B =0, MCD(6,0)=6.

MCD(270,192) = 6.

Recursión

Mayor común divisor recursivo

$$\text{Euclides}\,(M, N) \longrightarrow \begin{cases} M & \text{Si } N = 0 \\ \text{Euclides}\,(N, M \text{ MOD } N) & \text{En cualquier otro caso} \end{cases}$$

Recursión

Mayor común divisor recursivo

1. *Si* $(N = 0)$

   entonces

   Hacer Euclides $\leftarrow M$

   si no

   Hacer Euclides $\leftarrow$ Euclides $(N, M \text{ MOD } N)$

2. {Fin del condicional del paso 1}

Recursión

Mayor común divisor recursivo

1. *Si* ($N = 0$)
   entonces
   Hacer Euclides ← M
   si no
   Hacer Euclides ← Euclides (N, M MOD N)
2. {Fin del condicional del paso 1}

Euclides	(M, N)	
2 353	1 651	
1 651	702	[1]
702	247	[2]
247	208	[3]
208	39	[4]
39	13	[5]
13	0	[6] → **13**

Euclides	(M, N)	
25 680	11 892	
11 892	1 896	[1]
1 896	516	[2]
516	348	[3]
348	168	[4]
168	12	[5]
12	0	[6] → **12**

```cpp
#include <iostream>
using namespace std;

// El maximo común divisor de dos numeros a y b
// es el numero mas grande que divide a a y divide a b.

/* El algoritmo de Euclides es un procedimiento para calcular el máximo común divisor (m.c.d.) de dos números. ...
Si la división es exacta, el m.c.d. es el número menor. Si la división no es exacta, entonces se toma el residuo,
y se divide tantas veces como haga falta para llegar a una división sin residuo.
*/

// funcion recursiva. La M es de modulo o residuo
int Euclides( int M, int N )
{
    if( N == 0 )
    {
        return M;
    }
    else
    {
        return Euclides( N, M % N );
    }
}

int main()
{
    int M, N;

    cout << "Escribe el primer numero entero a comparar: ";
    cin >> M;
```

```cpp


    cout << "Escribe el segundo numero entero a comparar: ";
    cin >> N;

    // funcion recursiva
    cout << "El maximo comun divisor es: " << Euclides(M, N);

    return 0;
}
```

Cairó, O. y Guardati, S. (2002). Estructuras de Datos, 2da. Edición. McGraw-Hill.

Joyanes, L. (2006). Programación en C++: Algoritmos, Estructuras de datos y objetos. McGraw-Hill.