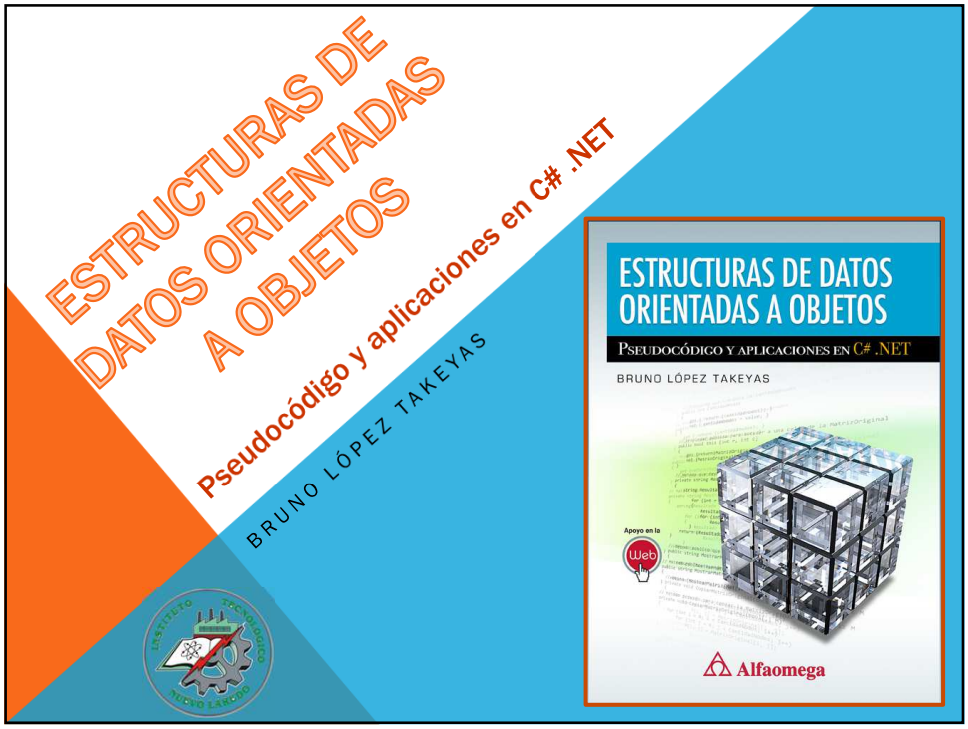

ESTRUCTURAS DE DATOS ORIENTADAS A OBJETOS
Pseudocódigo y aplicaciones en C# .NET
BRUNO LÓPEZ TAKEYAS
ESTRUCTURAS DE DATOS ORIENTADAS A OBJETOS
PSEUDOCÓDIGO Y APLICACIONES EN C# .NET
BRUNO LÓPEZ TAKEYAS
Apoyo en la Web
Alfaomega

1


RECURSIVIDAD
2

2

Preguntas detonadoras

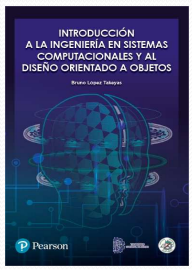
- ¿Qué es recursividad?
- ¿Qué características distinguen a los algoritmos recursivos?
- ¿Cuáles son las ventajas y desventajas de los algoritmos recursivos?
- ¿Qué es una variable local?
- ¿… y una global?
- ¿Cuál es la diferencia entre enviar un parámetro por valor y uno por referencia?
- ¿Cómo se diseña un algoritmo recursivo?
- ¿Se puede evitar la recursividad? ¿…cómo?

3

3

Tarea 3.01.- Repaso métodos

- *Leer Capítulo 8.- Métodos*
- *Contestar el cuestionario en MS Teams*
- *Se aceptará esta tarea si se obtiene calificación aprobatoria*



https://nlaredo.tecnm.mx/takeyas/Apuntes/Fundamentos%20de%20Programacion/Apuntes/05.-Metodos.pdf

4

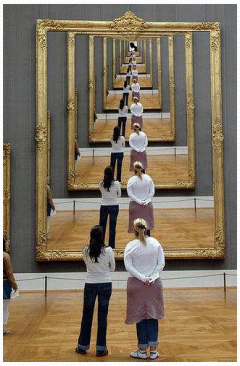
Recursividad

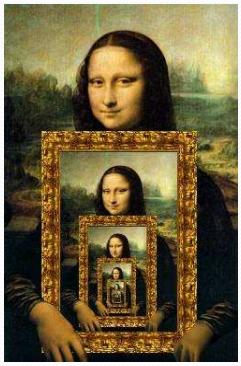
Es una forma de expresar un procedimiento en base a su propia definición, reduciéndolo de manera recurrente a la mínima expresión.

Un problema recursivo puede dividirse en instancias más pequeñas para resolverlas individualmente de manera repetitiva hasta encontrar una solución global

5

Ejemplos de recursividad en la vida cotidiana





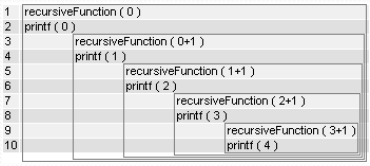
6

6

Algoritmos recursivos

- *Algoritmos que se llaman en cascada a sí mismos para resolver un problema*
- *Debe cuidarse que estas llamadas no sean infinitas*
- *Deben tener una condición de salida*

- *Métodos que …*
  - *se invocan a sí mismos.*
  - *tienen una condición de salida (caso base).*
- *Las llamadas recursivas dividen el problema y se aproximan al caso base.*



7

7

Ventajas

- ***Extensión***.- *Algoritmos más pequeños*
- ***Expresión***
  - *Elegante*
  - *Sencillo de escribir, de leer y de analizar.*
- ***División del problema.-*** *Subproblemas que recombinan sus soluciones.*
- ***Análisis y mantenimiento.-*** *Técnicas de inducción matemática*
- ***Ejecución***
  - *Se delega la responsabilidad del control de sus llamadas recursivas y los valores de sus variables al sistema*
  - *En otros algoritmos, el programador debe prever estas situaciones en el diseño.*

8

8

Desventajas

- ***Eficiencia***.- *Menos eficientes*
- ***Costo computacional:***
  - ***Velocidad***- *Más lentos*
  - ***Sobrecarga.***- *Demasiadas llamadas recursivas*
  - ***Memoria.***- *Ocupan más memoria*

9

9

Ámbito de las variables

Ámbito de variables

Locales – *Se declaran y utilizan dentro de un contexto específico. No se puede hacer referencia a ellas fuera de la sección de código donde se declara.*

Globales – *Se declaran fuera del cuerpo de cualquier método*

10

10

Parámetros recibidos por los métodos

Parámetros

- *Por valor* – *Se envía una copia del valor de la variable*
- *Por referencia* – *Se envía la dirección de la variable*

11

11

Envío de parámetros por valor

```
class Program
{
   static void Main(string[] args)
   {
         int x=10;
         HacerAlgo( x ); // Se envía el valor de x
         Console.Write("x=" + x); // x=10
   }
   static void HacerAlgo(int y)
   {
         y+=5;
         Console.Write("y=" + y); // y=15
   }
}
```

La variable "y" recibe el valor de la variable "x"

12

12

Envío de parámetros por referencia

```
static void Main(string[] args)
{
  int x = 10; // Se declara e inicializa x
  HacerAlgo(ref x);  // Se envía la referencia de x
  Console.WriteLine("x=" + x); // x=15
}

static void HacerAlgo(ref int y)
{
  y += 5;
  Console.WriteLine("y=" + y);  // y=15
}
```

¡ Se modificó el valor de la variable x !

13

Parámetros por referencia (apuntador)

Memoria RAM

Dirección	Valor	Variable
FA31:B278	5	x
...		
FA31:C45C	13	y
...		
FA31:D2A8	FA31:C45C	a
...		

14

14

Factorial de un número

$$n! = \prod_{1}^{n} n$$

4! = 4 * 3 * 2 * 1
3! = 3 * 2 * 1
2! = 2 * 1
1! = 1

Y por definición:
0! = 1 y

- *De ahí se deriva que:*

n! = n * (n-1) * (n-2) * …* 0

- *Por lo tanto,* n! = n * (n-1)!

15

15

Factorial de un número

$$n! = \prod_{1}^{n} n$$

Por lo tanto…
4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1!
1! = 1 * 0!

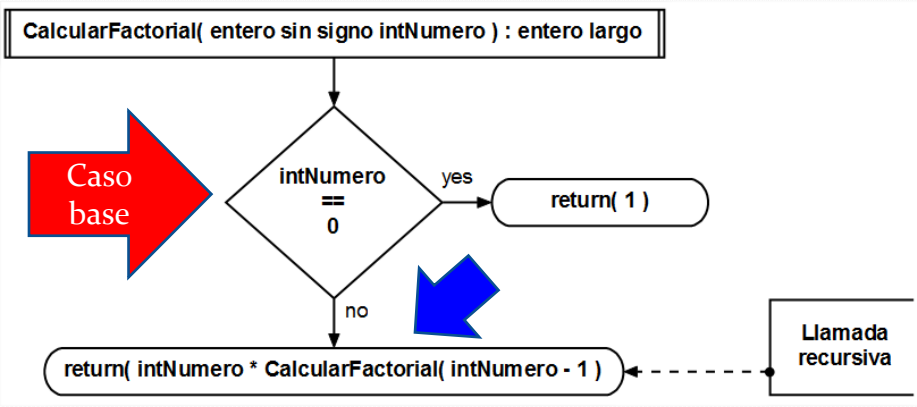
- *De ahí se deriva que:*
- n! = n * (n-1)!

Y por definición:
0! = 1

16

16

Algoritmo recursivo para calcular el factorial



17

17

Codificación del método

```
public static long CalcularFactorial(uint intNumero)
{
   if (intNumero == 0)
        return (1);
   else
        return (intNumero * CalcularFactorial(intNumero - 1));
}
```

18

18

Codificación del método mediante el operador condicional ?

```
public static long CalcularFactorial(uint intNumero)
{
   return (intNumero == 0 ? 1 : intNumero * CalcularFactorial(intNumero - 1);
}
```

19

19

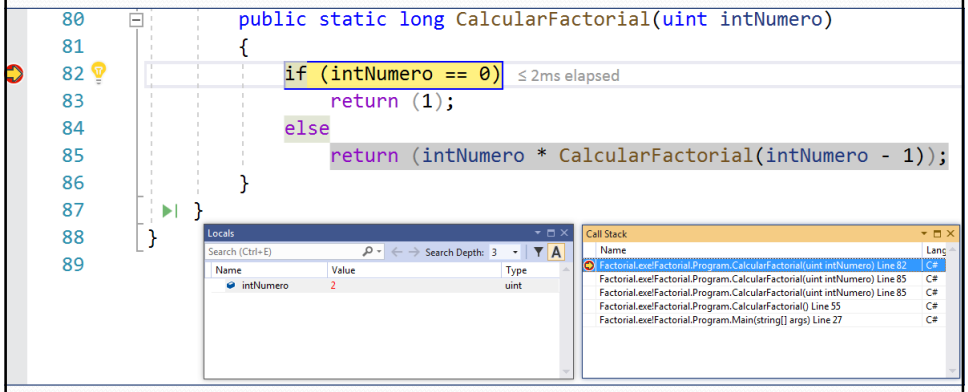
Calcular 4!

Llamada	Valor de intNumero	Cálculo	Devuelve
1	4	4 * Factorial(3)	
2	3	3 * Factorial(2)	
3	2	2 * Factorial(1)	
4	1	1 * Factorial(0)	
5	0		1
4	1	1 * 1	1
3	2	2 * 1	2
2	3	3 * 2	6
1	4	4 * 6	24

20

20

Vista de las variables locales y la pila de llamadas recursivas en Visual Studio



21

Riesgos …

```
CALCULAR EL FACTORIAL DE UN NUMERO POSITIVO ENTERO
Número? 26

26! = -1,569,523,520,172,457,984
Oprima cualquier tecla para continuar...
```

- *¿A qué se debe que el resultado es negativo?*
- *OverflowException .- Desborde aritmético*
- *StackOverflowException.- Desborde de la pila de llamadas*

22

22

Desbordes aritmético y de pila

- `OverflowException.-` *Ocurre cuando un dato "no cabe" en la variable*
- `StackOverflowException.-` *Ocurre cuando se satura la pila de llamadas*
- *Estas excepciones **NO** se disparan regularmente cuando ocurre un desborde*
- *Requieren la sentencia* `checked`

23

23

Verificación de desbordes

```
public static long CalcularFactorial(uint intNumero)
 {
    if (intNumero > 65) // Máx. aprox. de llamadas recursivas
       throw new Exception("Demasiadas llamadas recursivas ...");

    if (intNumero == 0)
      return (1);
    else
      return checked(intNumero*CalcularFactorial(intNumero-1));
 }
```

24

24

Uso de la sentencia checked

```
CALCULAR EL FACTORIAL DE UN NUMERO POSITIVO ENTERO
Número? 26
Resultado demasiado grande (desborde aritmético)
```

```
CALCULAR EL FACTORIAL DE UN NUMERO POSITIVO ENTERO
Número? 66
Demasiadas llamadas recursivas ...
```

25

25

Juego recursivo: Torres de Hanoi

http://www.uterra.com/juegos/torre_hanoi.php

26

26

Representación

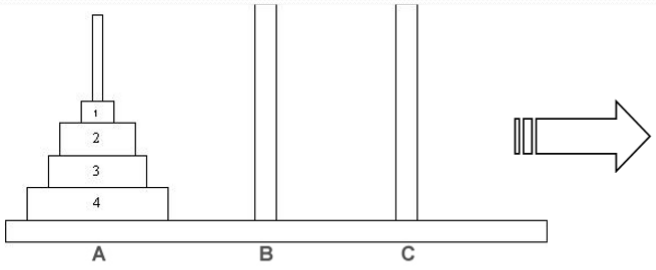
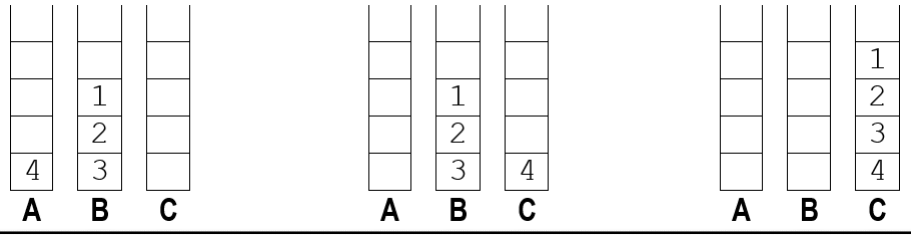


Fig. 7.9. Representación con pilas de las torres de Hanoi.



27

Algoritmo Torres de Hanoi

Algoritmo Torres de Hanói (Complejidad $\Theta(2^n)$)

Entrada: Tres pilas de números *origen*, *auxiliar*, *destino*, con la pila *origen* ordenada

Salida: La pila *destino*

1. **si** *origen* $==\{1\}$ **entonces**
    1. **mover** el disco 1 de pila origen a la pila destino (insertarlo arriba de la pila destino)
    2. **terminar**
2. **si no**
    1. *hanoi*($\{1,...,n-1\}$,*origen*,*destino*, *auxiliar*) //mover todas las fichas menos la más grande (*n*) a la varilla auxiliar
3. **mover** disco *n* a *destino* //mover la ficha grande hasta la varilla final
4. *hanoi* (*auxiliar*, *origen*, *destino*) //mover todas las fichas restantes, 1...*n*–1, encima de la ficha grande (*n*)
5. **terminar**

28

28

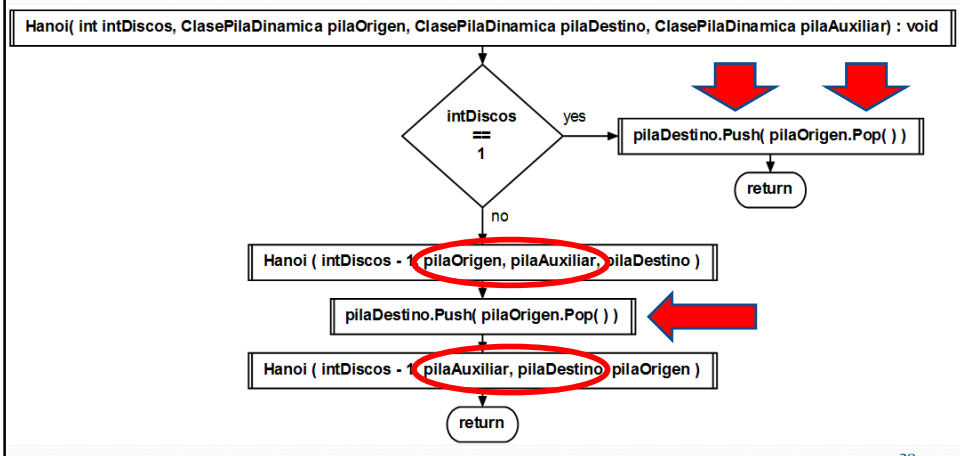
Pseudocódigo

```
Hanoi(Discos, Origen, Destino, Auxiliar): nulo
1.- SI Discos == 1 ENTONCES  // Caso base
      1.1.1. IMPRIMIR "Mover disco de "+Origen+" a "+Destino
    SINO
      1.2.1. Hanoi(Discos-1, Origen, Auxiliar, Destino)
      1.2.2. IMPRIMIR "Mover disco de "+Origen+" a "+Destino
      1.2.3. Hanoi(Discos-1, Auxiliar, Destino, Origen)
2.- {FIN DE LA CONDICIONAL DEL PASO 1}
3.- RETURN
```

29

29

Diagrama de flujo: Torres de Hanoi "pensadas" con objetos



30

30

Demo: Torres de Hanoi

Prog. 7.5.- Torres de Hanoi Formas

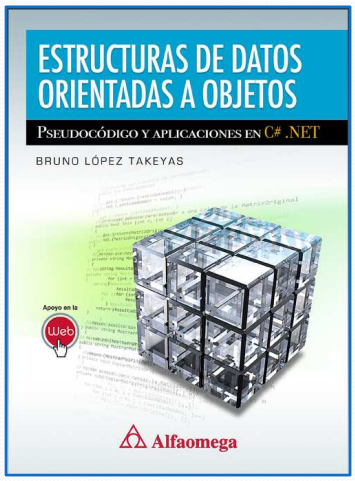
31

31

```
 private void Hanoi(int Discos, ClasePilaDesordenadaLista<ClaseNodoPilaLista> Origen,
ClasePilaDesordenadaLista<ClaseNodoPilaLista> Destino, ClasePilaDesordenadaLista<ClaseNodoPilaLista> Auxiliar, ref int
CantidadMovimientos)
        {
            // Creación del objeto para el disco
            ClaseNodoPilaLista Disco= new ClaseNodoPilaLista();

            if (Discos == 1)  // Caso base
            {
                Disco = Origen.Pop();  // Saca un disco de la torre origen
                Destino.Push(Disco);  // Inserta el disco en la torre destino
                CantidadMovimientos++;  // Incrementa la cantidad de movimientos
                MostrarTorres();  // Actualiza los listBoxes

                if(radioButton1.Checked)
                    MessageBox.Show("Disco "+Disco.Dato.ToString()+" de "+Origen.Nombre+" a "+Destino.Nombre,
"MOVIMIENTO DE DISCO", MessageBoxButtons.OK, MessageBoxIcon.Information);
                else
                    System.Threading.Thread.Sleep(1000);  // Retardo
            }
            else
            {
                // LLamada recursiva
                Hanoi(Discos - 1, Origen, Auxiliar, Destino, ref CantidadMovimientos);

                Disco = Origen.Pop(); // Saca un disco de la torre origen
                Destino.Push(Disco);  // Inserta el disco en la torre destino
                CantidadMovimientos++;  // Incrementa la cantidad de movimientos
                MostrarTorres();  // Actualiza los listBoxes

                if (radioButton1.Checked)
                    MessageBox.Show("Disco " + Disco.Dato.ToString() + " de " + Origen.Nombre + " a " + Destino.Nombre,
"MOVIMIENTO DE DISCO", MessageBoxButtons.OK, MessageBoxIcon.Information);
                else
                    System.Threading.Thread.Sleep(1000);  // Retardo

                // LLamada recursiva
                Hanoi(Discos - 1, Auxiliar, Destino, Origen, ref CantidadMovimientos);
            }
        }
```

32

Lectura



Para reforzar este tema se recomienda la lectura de:

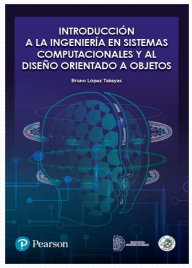
Capítulo 7.- Recursividad

33

33

Tarea 3.01.- Repaso métodos

- *Leer Capítulo 8.- Métodos*
- *Contestar el cuestionario en MS Teams*
- *Se aceptará esta tarea si se obtiene calificación aprobatoria*

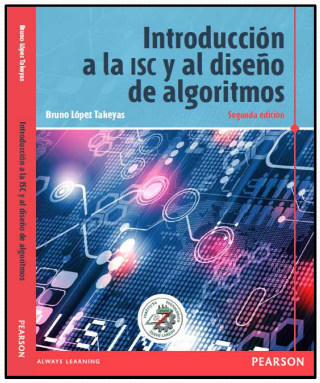


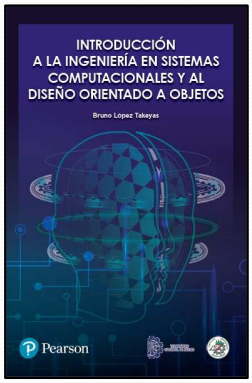
https://nlaredo.tecnm.mx/takeyas/Apuntes/Fundamentos%20de%20Programacion/Apuntes/05.-Metodos.pdf

34

Otros libros del autor

https://nlaredo.tecnm.mx/takeyas/Libro





bruno.lt@nlaredo.tecnm.mx

Bruno López Takeyas

35