PROGRAMACIÓN ESTRUCTURADA

Trabajo Práctico N° 8

Recursividad

08

Apellido y Nombre: …….. *Fecha: ……/……/……*

CONCEPTOS TEÓRICOS A TENER EN CUENTA

La recursividad consiste en realizar una definición de un concepto en términos del propio concepto que se está definiendo. Para que una definición recursiva sea válida, la referencia a sí misma debe ser relativamente más sencilla que el caso considerado.

Una definición recursiva indica cómo obtener nuevos conceptos a partir del concepto que se intenta definir. Esto es, la recursividad expresa un concepto complejo en función de las formas más simples del mismo concepto.

Un razonamiento recursivo tiene dos partes:

- caso base y
- regla recursiva de construcción

El caso base no es recursivo y es el punto tanto de partida como de terminación de la definición.

La regla recursiva de construcción intenta resolver una versión más simple (pequeña) del problema original.

Por lo tanto, para resolver problemas en forma recursiva debe considerarse:

1. la división sucesiva del problema original en uno o varios problemas más pequeños, del mismo tipo que el inicial;
2. la resolución de los problemas más sencillos, y
3. la construcción de las soluciones de los problemas complejos a partir de las soluciones de los problemas más sencillos.

Teniendo en cuenta todo esto, un algoritmo recursivo se caracteriza por:

1. el algoritmo debe contener una llamada a sí mismo,
2. el problema planteado puede resolverse atacando el mismo problema pero de tamaño menor,
3. la reducción del tamaño del problema eventualmente permite alcanzar el caso base, que puede resolverse directamente sin necesidad de otro llamado recursivo.

En resumen, un algoritmo recursivo consta de una parte recursiva, otra iterativa o no recursiva y una condición de terminación. La parte recursiva y la condición de terminación siempre existen. En cambio la parte no recursiva puede coincidir con la condición de terminación. Es importante destacar que siempre debe alcanzarse el caso base, si esto no se cumple el algoritmo nunca se finalizará.

<u>Ventajas e inconvenientes</u>

La principal ventaja es la simplicidad de comprensión y su gran potencia, favoreciendo la resolución de problemas de manera natural, sencilla y elegante; y facilidad para comprobar y convencerse de que la solución del problema es correcta.

Los algoritmos que por naturaleza son recursivos y donde la solución iterativa es complicada y debe manejarse explícitamente una pila para emular las llamadas recursivas, deben resolverse por métodos recursivos.

En general, las soluciones recursivas son menos eficientes que las iterativas (coste mayor en tiempo y memoria). Tenga en cuenta que por cada invocación se crea una copia en memoria del programa recursivo con los parámetros indicados en el llamado.

Cuando haya una solución obvia al problema por iteración, debe evitarse la recursividad.

<u>Tipos de Recursividad</u>

La recursividad puede ser de 2 tipos:

- Recursividad Directa (simple): un algoritmo se invoca a sí mismo una o más veces directamente.
- Recursividad Indirecta (mutua): un algoritmo *A* invoca a otro algoritmo *B* y éste a su vez invoca al algoritmo *A*.

Problemas resueltos mediante recursividad

Un ejemplo clásico de un problema que se resuelve utilizando recursividad es el cálculo del factorial de un número. Suponga que se calcula el factorial de 5, éste es:

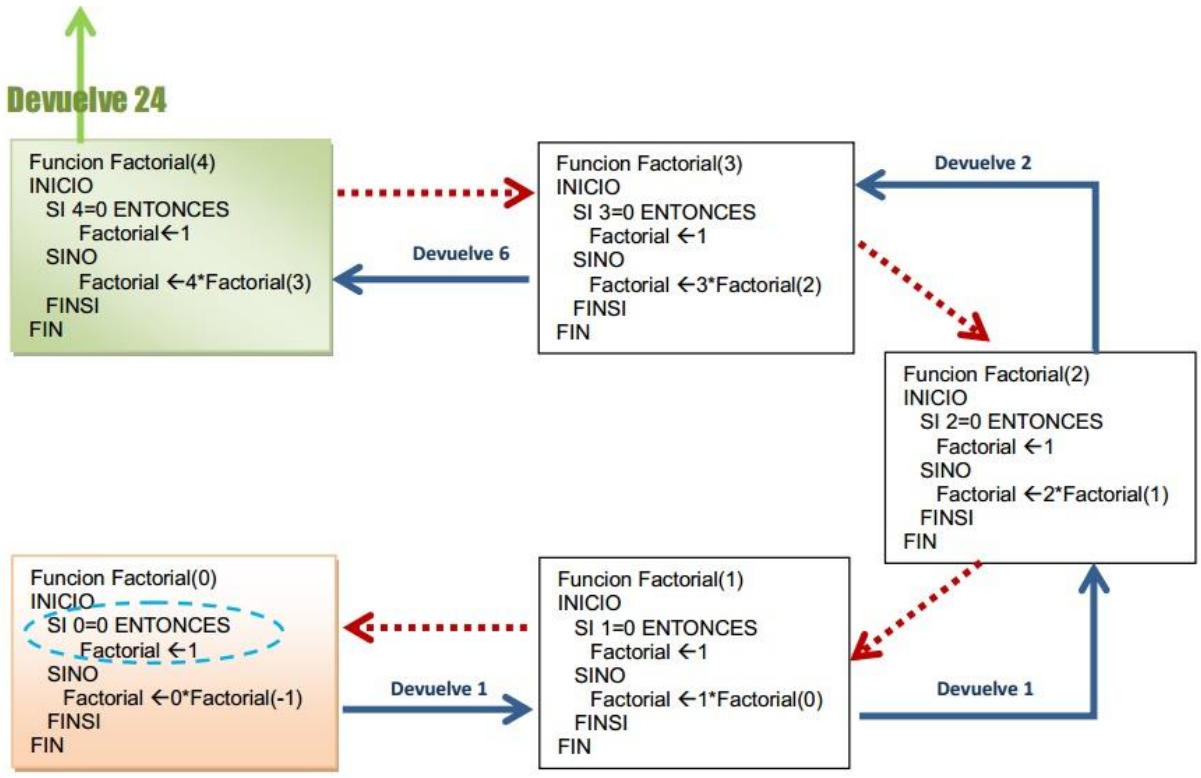
5! = 120

Esto puede expresarse también como:

Problema	***Descomposición del Problema***
*5! = 5 * 4!*	*Factorial de 5 (Original)*
*4! = 4 * 3!*	*Factorial de 4*
*3!= 3 * 2!*	*Factorial de 3*
*2! = 2 * 1!*	*Factorial de 2*
*1! = 1 * 0!*	*Factorial de 1*
0! = 1	*Solución Directa Factorial de 0 = 1*

Como puede observarse este problema se expresa en términos cada vez más simples hasta que se alcanza el caso base, donde la solución más simple se obtiene directamente (por definición se sabe que el factorial de 0 es 1). A partir del caso base pueden obtenerse las soluciones para los problemas más complejos.

```
FUNCION Factorial(E n: ENTERO):ENTERO
INICIO
    SI n=0 ENTONCES
       Factorial←1;
    SINO
       Factorial←n*Factorial(n-1);
    FINSI
FIN
```



Procedimientos y Funciones Recursivos

Un procedimiento *P* que contiene una sentencia de llamada a sí mismo, se dice que es un procedimiento recursivo. Un procedimiento recursivo debe cumplir con lo siguiente:

1. incluye un cierto criterio, llamado caso base, por el que el procedimiento no se llama a sí mismo.
2. cada vez que el procedimiento se llame a sí mismo (directa o indirectamente), debe estar más cerca del caso base.

Un procedimiento recursivo con estas dos propiedades se dice que está bien definido.

De igual modo, una función es recursiva si su definición contiene una invocación a sí misma. Una función recursiva debe cumplir con lo siguiente:

1. debe haber ciertos argumentos, llamados valores base, para los que la función no se refiera a sí misma.
2. cada vez que la función se refiera a sí misma, el argumento de la función debe acercarse más al valor base.

Una función recursiva con estas dos propiedades se dice que está bien definida.

EJERCICIOS RESUELTOS

Los siguientes ejemplos ilustran procedimientos y funciones recursivos.

1. El procedimiento *Mostrar_Numeros* visualiza, en orden creciente, valores enteros entre 1 y cantidad. Por ejemplo, si se invoca a *Mostrar_Numeros* con el parámetro cantidad igual a 5, el algoritmo se comporta como sigue:

```
PROCEDIMIENTO Mostrar_Numeros(E cantidad: entero)
INICIO
    SI cantidad=1 ENTONCES
       ESCRIBIR cantidad
    SINO
       Mostrar_Numeros(cantidad-1)
       ESCRIBIR cantidad
    FINSI
FIN
```

1. Se invoca al procedimiento con el parámetro *cantidad* igual a 5, se evalúa la condición (*cantidad=1*) y como ésta no se cumple, se invoca a *Mostrar_Numeros* con parámetro *cantidad* igual a 4. Tenga en cuenta que la acción *ESCRIBIR cantidad* (con valor 5 en este caso) no se realizará hasta que finalice la ejecución del nuevo llamado al procedimiento.

Parámetro cantidad	Condición (cantidad=1)	Acción
5	FALSO	Invocar Mostrar_Numeros(4) **Escribir cantidad = 5**

2. *Mostrar_Numeros*, ahora con parámetro *cantidad* igual a 4, comprueba la condición (*cantidad=1*). Al evaluar dicha condición se determina que es FALSA, y por lo tanto se realiza el llamado al procedimiento *Mostrar_Numeros* con parámetro *cantidad* igual a 3. Esto se repite en tanto no se cumpla que *cantidad* sea igual a 1. Las acciones *ESCRIBIR cantidad* con valores 4, 3 y 2 no se llevan a cabo hasta que finalice la ejecución de las invocaciones realizadas.

Parámetro cantidad	Condición (cantidad=1)	Acción
4	FALSO	Invocar Mostrar_Numeros(3) **Escribir cantidad = 4**

Parámetro cantidad	Condición (cantidad=1)	Acción
3	FALSO	Invocar Mostrar_Numeros(2) **Escribir cantidad = 3**

Parámetro cantidad	Condición (cantidad=1)	Acción
2	FALSO	Invocar Mostrar_Numeros(1) **Escribir cantidad = 2**

3. Finalmente, cuando se invoca a *Mostrar_Numeros* con parámetro *cantidad* igual a 1, se verifica la condición (*cantidad=1*) que finaliza la recursión (no se realizan más llamados al procedimiento) y se ejecuta la sentencia *ESCRIBIR cantidad* (con valor 1). Al terminar la ejecución de este llamado, se realiza la instrucción *ESCRIBIR cantidad* del procedimiento *Mostrar_Numeros* que tiene parámetro *cantidad* igual a 2, al finalizar éste se ejecutan las instrucciones ESCRIBIR de todos los llamados anteriores.

Parámetro cantidad	Condición (cantidad=1)	Acción
1	VERDADERO	**Escribir cantidad = 1**

Nótese que el procedimiento ejemplificado contiene las 2 partes indicadas para los algoritmos recursivos, una que realiza el llamado al mismo procedimiento y otra que finaliza la recursión (condición *cantidad=1*).

2. La función *Potencia* calcula el resultado de elevar un número entero a al exponente entero b, mediante productos sucesivos. Por ejemplo, si se invoca a *Potencia* con los parámetros *a* igual a 3 y *b* igual a 4, el algoritmo se comporta como sigue:

```
FUNCIÓN Potencia(E a:entero, E b:entero): entero
INICIO
    SI b=1 ENTONCES
       Potencia←a
    SINO
       Potencia←a * Potencia(a,b-1)
    FINSI
FIN
```

1. Al ejecutarse la función *Potencia* con los parámetros *a=3* y *b=4* se evalúa la condición (*b=1*) indicada en el algoritmo. Dicha condición resulta FALSA, por lo que se realiza *Potencia←a*Potencia(a,b-1)*, sentencia que invoca a *Potencia* con los parámetros *a=3* y *b=3*. Tenga en cuenta el valor final de la función no se obtiene hasta que todos los llamados recursivos finalicen su ejecución.

Parámetro *a*	Parámetro *b*	Condición (b=1)	Acción
3	4	FALSO	Potencia<-3 * Potencia(3,3) **Potencia = 81**

2. *Potencia*, con parámetros *a=3* y *b=3*, comprueba la condición (*b=1*) del algoritmo. Al evaluar dicha condición se determina que es FALSA, y por lo tanto se realiza el llamado a la función *Potencia* con parámetros *a=3* y *b=2*. Esto se repite en tanto no se cumpla que el parámetro *b* sea igual a 1. El valor de la función para cada llamado no se calcula hasta que finalice la ejecución de las invocaciones realizadas.

Parámetro *a*	Parámetro *b*	Condición (b=1)	Acción
3	3	FALSO	Potencia<-3 * Potencia(3,2) **Potencia = 27**

Parámetro *a*	Parámetro *b*	Condición (b=1)	Acción
3	2	FALSO	Potencia<-3 * Potencia(3,1) **Potencia = 9**

3. Finalmente, cuando se llama a *Potencia* con parámetros *a=3* y *b=1*, se verifica la condición (*b=1*) que finaliza la recursión (no se realizan más llamados a la función) y se ejecuta la sentencia *Potencia←a*, que asigna a la función el valor 3. Al terminar la ejecución de este llamado, se calcula el valor de la función *Potencia* con los parámetros *a=3* y *b=2* cuyo resultado es 9. Al finalizar ésta se calcula el valor de la función para todos los llamados anteriores.

Parámetro *a*	Parámetro *b*	Condición (b=1)	Acción
3	1	VERDADERO	**Potencia=3**

Nótese que el función ejemplificada contiene las 2 partes indicadas para los algoritmos recursivos, una que realiza el llamado a la misma función y otra que finaliza la recursión (condición *b=1*).

EJERCICIOS A RESOLVER

1. Realice la prueba de escritorio del módulo *calculo* del siguiente programa. Luego, codifique y compruebe los resultados obtenidos.

```
#include <iostream>
#include <stdlib.h>

using namespace std;
```

```
int calculo(int n);
main()
{
      int m;
      cout << "Ingrese dato: ";
      cin >> m;
      cout << "Resultado: " << calculo(m)<< endl;
      system("pause");
}

int calculo(int n)
{
  if (n==0)
    return 1;
  else
    return n*calculo(n-1);
}
```

2. Diseñe un procedimiento/función recursivo que calcule el producto de dos números enteros (utilice el concepto de sumas sucesivas).
3. Diseñe un procedimiento/función recursivo que calcule el cociente de la división entera de dos números enteros (utilice el concepto de restas sucesivas).
4. Compare los siguientes módulos recursivos y determine
   a) ¿cuál es el objetivo de estos módulos?
   b) ¿cuál es el propósito de la variable *p* del módulo *enigma*?

```
int incognita(int a, int b)
{
 if (b==0)
   return 1;
 else
   return a * incognita(a,b-1);
}
```

```
void enigma(int m, int n, int &p)
{
  if (n==0)
     p=1;
else
  { enigma(m,n-1,p);
     p=p*m;
  }
}
```

5. Modifique el programa del ejercicio 1 sustituyendo la función *calculo* por un procedimiento. ¿qué cambios realizó para adaptar el programa?
6. Diseñe un procedimiento/función recursivo que calcule la cantidad de dígitos de un número entero. Por ejemplo, dado el valor 1364, el módulo obtendrá el valor 4.
7. Diseñe los procedimientos/funciones recursivos que calculen las siguientes expresiones:

   a) $\sum_{k=1}^{numero} 2 \times k - 1$ b) $\prod_{i=0}^{total} 2 \times a^i$

8. Diseñe un procedimiento/función recursivo que calcule el MCD (Máximo Común Divisor) de 2 números enteros positivos. Para ello utilice el método de Euclides. Dicho método contempla los siguientes pasos:
   a) Dividir el número mayor (**X**) por el menor (**Y**). Si el resto de la división es cero, el número **Y** es el máximo común divisor.
   b) Si la división no es exacta, se divide el número menor (**Y**) por el resto de la división anterior.
   c) Se siguen los pasos anteriores hasta obtener un resto cero. El último divisor es el **MCD** buscado.
9. La siguiente secuencia de números es conocida como Serie de Fibonacci:

   1, 1, 2, 3, 5, 8, 13, 21, … , m, n, (m+n), …

   Cada término de la serie, a excepción del primer y segundo término (cuyo valor es 1), se calcula sumando el valor de los 2 términos precedentes. Por ejemplo, el 6º término de la serie (cuyo valor es 8) se obtiene de sumar los términos 4º (cuyo valor es 3) y 5º (cuyo valor es 5). Diseñe un procedimiento/función recursiva que calcule un término cualquiera de la serie.
10. Inspirándose en la serie de Fibonacci defina una serie personalizada (indique cómo se formará) y diseñe un procedimiento/función recursivo que calcule un término cualquiera de la serie propuesta.

11. El cuadrado de un número *N* puede calcularse como la suma de los ***N*** primeros números impares. Diseñe un procedimiento/función recursivo que calcule el cuadrado de un entero positivo *N* basándose en esta propiedad. Tenga en cuenta que el N-ésimo número impar se puede obtener como 2 * *N* – 1.

N	N-ésimo impar	N^2
1	2 * **1** – 1 = *1*	1
2	2 * **2** – 1 = *3*	3+1=4
3	2 * **3** – 1 = *5*	5+3+1=9
4	2 * **4** – 1 = *7*	7+5+3+1=16
5	2 * **5** – 1 = *9*	9+7+5+3+1=25

12. Diseñe un procedimiento/función recursivo que, dado un número entero, determine cuál es el dígito de mayor valor. Por ejemplo, dado el valor 4376, el módulo obtendrá el valor 7.
13. Diseñe un procedimiento/función recursivo que determine si un número es primo o no. Recuerde que un número *N* es primo si es divisible (resto igual a cero) ÚNICAMENTE por la unidad (1) y el mismo número (*N*).
14. Realice la prueba de escritorio del siguiente módulo recursivo (n=2018) y determine su propósito.

```
FUNCIÓN incognita(E n:ENTERO):ENTERO
INICIO
   SI (n=0) ENTONCES
     incognita←0
   SINO
     incognita←n mod 10 + incognita(n div 10)
   FIN_SI
FIN
```

15. Realice la prueba de escritorio del siguiente módulo recursivo (x=13, y=3), (x=25, y=5), y determine su propósito. Además indique cómo se realiza el llamado de este procedimiento desde el programa principal.

```
PROCEDIMIENTO incognita(E x: ENTERO, E y: ENTERO, E/S a: ENTERO, E/S b: ENTERO)
INICIO
   SI(x < y) ENTONCES
       a←x
       b←0
   SINO
      incognita(x-y, y, a, b)
      b←b+1
   FIN_SI
FIN
```