RECURSIVIDAD

Recursión: Funcion que se llama a si misma en su cuerpo de definición

f(x) = …. f()

Ej: Factorial(x)

fact(x)= { 1 si x = 1
x * fact(x-1) si x >1 definicion de la funcion por tramos

x pertenecen al conj de num. naturales

fact(4) = 4 * fact(4-1) = 4 * fact(3) seguimiento de la ejecucion
= 4 * (3 * fact(2))
= 4 * (3 * (2* fact(1))
= 4 * (3 *(2 *1))
= 4 * (3 *2)
= 4 * 6
= 24

recursividad simple:
f = … f …
recursividad multiple:
f = … f … f … (funcion de fibonacci)
recursividad indirecta:
f = … g …
g = …f …

Características de una función recursiva bien definida:

1) tener un tramo sin llamada recursiva -> brinde un valor concreto
2) el / los tramos recursivos tienen llamadas con su parametro/s (que van cambiando de llamada en llamada) de forma tal que se garantice que el tramo no recursiva SIEMPRE va a ser alcanzado

Secuencia fibonacci = 1,1,2,3,5,8,13,21…..

fibo(n) = { 1   n =1,2
fibo(n-1)+fib(n-2)   n>2

fib(5) = fibo(4) + fibo(3)
= fibo(3) + fib(2) + fibo(2) + fibo(1)
= fibo(2) + fibo(1) + 1 + 1+ 1
= 1 + 1 +1 +1 + 1

fibo(7) = fibo(6) + fibo(5)
= f(5)+f(4) + f(4) + f(3)
= f(4) + f(3) + f(3)+f(2) + f(3)+f(2) + f(2)+f(1)

g(n) = { 10 n = 100
1+ g(n+1) n<100

dominio = naturales entre 1..100
c[char] = [‘s’,’a’,’n’,’t’,’i’,’a’,’g’,’o’,’’]

concat(c)=cAux(c,0,””)

cAux(c[char],int p, string s) = { s c[p] == ‘’
cAux(c,p+1,s.c[p]) c[p] != ‘’

concatenacion de array de chars
char c[100] ;
concat(char [100]c , int p){
if (p=0) {return RESULT_ACUM}
else c[p]+concat(c,p-1)
}

String c = concat(arr,99);

n-> el valor a calcular
p-> posic donde estoy calculando
a1 -> fib anterior a p (fibo(p-1))
a2 -> fib ante-anterior p (fibo(p-2))

fAux(n,p,a1,a2) = { fAux(n, p+1, a1+a2, a1) n>p
a1+a2 n=p

supuesto= ya conozco f(p-1) f(p-2) -> resultado = f(p-1)+f(p-2)

fibo(n) = { 1 n < 3
fAux(n,3,1,1) n>=3

fibo(6) = fAux(6,3,1,1)
= fAux(6,4,1+1,1)
= fAux(6,5,2+1,2)
=fAux(6,6,3+2,3)
= 5+3 = 8