Clase #13

Martes 5 de Noviembre de 2013

*Recursividad. Iteraciones vs. Recursividad. Ventajas y desventajas.*

Recursividad

* + 1. Un objeto es recursivo cuando una parte de el, esta formada por el objeto mismo. Es decir estaríamos en presencia de objetos en una parte del objeto esta formada por sigo mismo.

Un identificador es o bien una letra o guión bajo o bien un identificador seguido por una letra, número o guión bajo.

|  |  |
| --- | --- |
| Identificador | - Una letra o guión bajo |
|  | - Un identificador seguido por una letra, número o guión bajo |

Función para calcular un factorial:

fact(4) = 4 \* 3 \* 2 \* 1 = 4 \* fact(3)

fact(3) = 3 \* 2 \* 1 = 3 \* fact(2)

fact(2) = 2 \* 1 = 2

fact(n) = n \* fact(n -1)

fact(0) = 1

int fact(int n){

if(n==0){//caso base

return 1;

}else{//caso recursivo

return n\*fact(n-1);

}

Pruebas de escritorio

|  |  |
| --- | --- |
| n | fact(n) |
| 4 | 4 \* fact(3) **6 = 24** |
| 3 | 3 \* fact(2)  **2 = 6** |
| 2 | 2 \* fact(1)  **1 = 2** |
| 1 | 1 \* fact(0)  **1 = 1** |
| 0 | 1 - NULL |

|  |  |
| --- | --- |
| n | fact(n) |
| -1 | -1 \* fact(-2) **6 = 24** |
| -2 | -2 \* fact(-3)  **2 = 6** |
| -3 | -3 \* fact(-4)  **1 = 2** |

Recursividad “infinita” → Desbordamiento de Pila = Stack overflow [iniciar en modo depuración]

Toda función recursiva deberá tener 2 partes

1- Caso base

2- Caso recursivo

Técnica “Divide y vencerás”

Si un problema es demasiado grande, es mejor dividirlo en problemas más pequeños para intentar resolver. Luego juntar todas las soluciones para llegar a la solución del problema principal.

Potencias:

24 = 2 \* 2 \* 2 \* 2 → 2 \* 23

23 = 2 \* 2 \* 2 → 2 \* 22

22 = 2 \* 2

A B = A \* A B - 1

A 0 = 1

int potencia(int a, int b){

if (b==0){//caso base

return 1;

}else{//caso recursivo

return a\*potencia(a,b-1);

}

}

Teorema:

Todo problema que admita una solución iterativa, admite también una solución recursiva y viceversa. Es decir si algo puede ser resuelto mediante un ciclo, también se puede resolver mediante recursividad.

Imprimir del 1 al 100 de forma recursiva

void imprirmir (int n){

if(n>0){

imprimir(n-1);

printf(“%4d”,n);

}

}

imprimir(100)

Pruebas de escritorio

|  |  |
| --- | --- |
| n | Imprime |
| 100 | **100** |
| 99 | **99** |
| 2 | **2** |
| 1 | **1** |
| 0 |  |

Sucesión de Fibonacci:

1, 1, 2, 3, 5, 8, 13, 21....

fib(n) = fib(n – 1) + fib(n – 2)

fib(1) = 1

fib(2) = 1

int fib(int n){

if(n==1||n==2){

return 1;

}else{

return fib(n-1)+fib(n-2);

}

}

*Ver árbol de recursividad Fibonacci*

Torres de Hanoi

#include <stdio.h>

void mover (int n, int origen, int destino, int aux);

int main(){

int discos;

printf(“Cantidad de discos?”);

scanf(“%d”, &discos);

mover(discos,1,2,3);

return 0;

}

void mover (int n, int origen, int destino, int aux){

if(n>0){

mover(n-1,origen,aux,destino);

printf(“Muevo un disco de %d a %d”,origen,destino);

mover(n-1,aux,destino,origen);

}

}

Formula

MOV = 2 Discos - 1

*Ver torre de Hanoi*

void imprimirlistar(tpuntero cabeza){

if(cabeza!=NULL){

printf(“%4d”,cabeza->valor);

imprimirlistar(cabeza->sig);

}

*}*