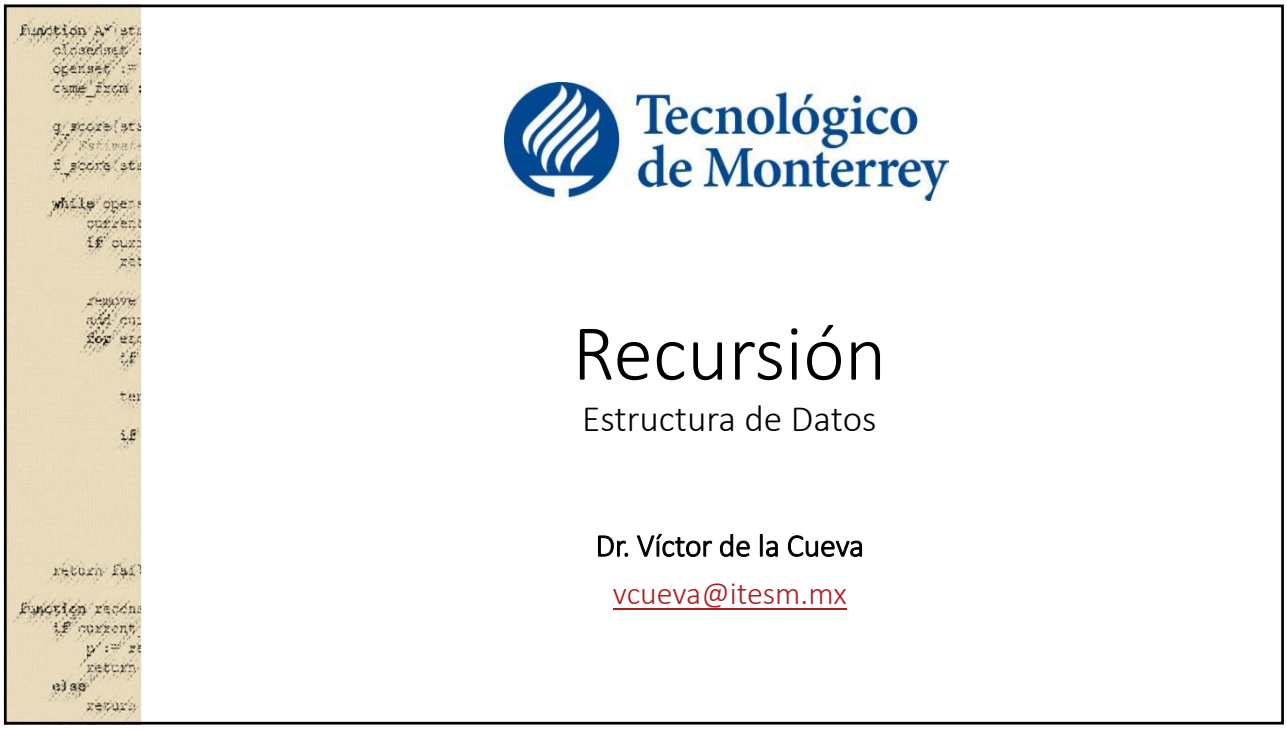


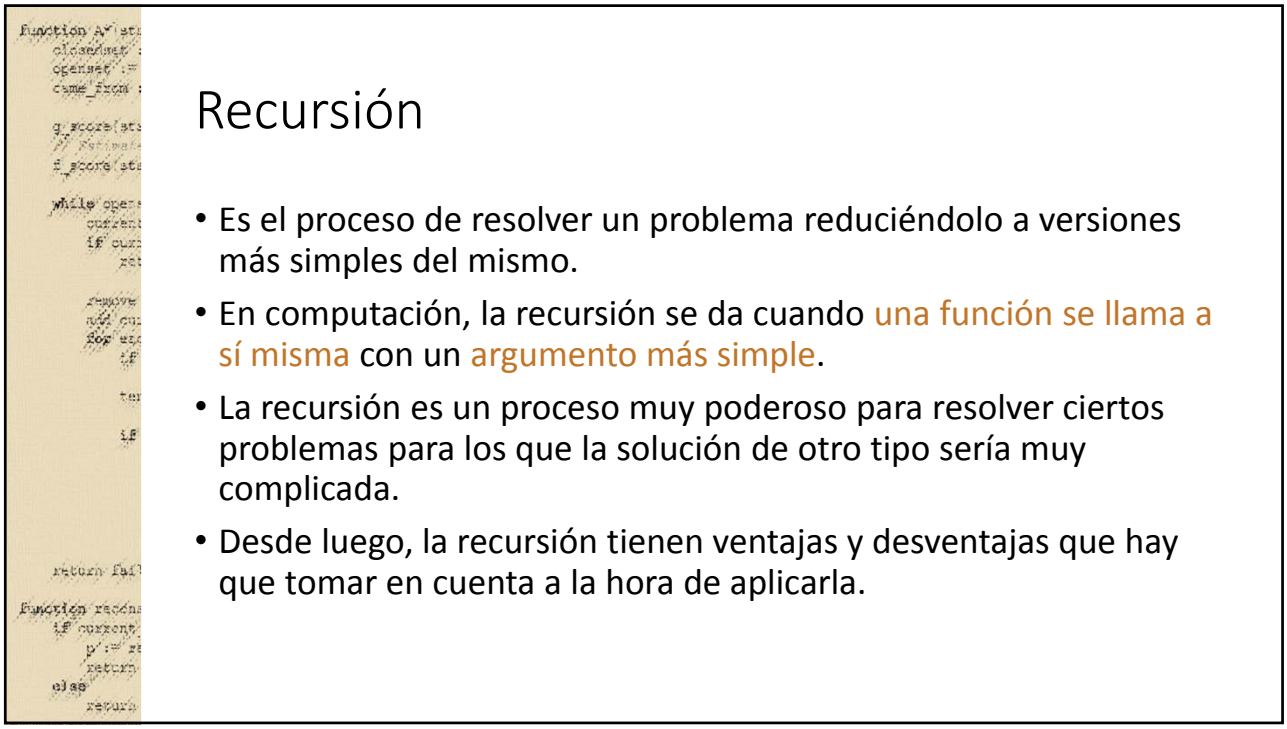
Tecnológico de Monterrey

# Recursión

Estructura de Datos

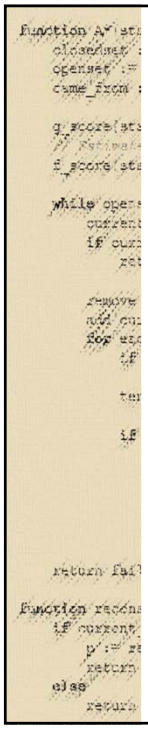
Dr. Víctor de la Cueva

vcueva@itesm.mx



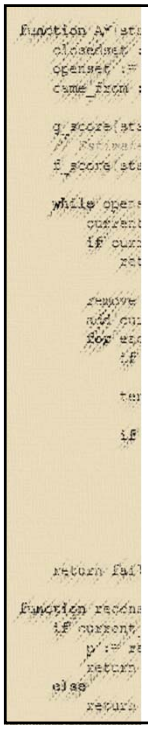
## Recursión

- Es el proceso de resolver un problema reduciéndolo a versiones más simples del mismo.
- En computación, la recursión se da cuando una función se llama a sí misma con un argumento más simple.
- La recursión es un proceso muy poderoso para resolver ciertos problemas para los que la solución de otro tipo sería muy complicada.
- Desde luego, la recursión tienen ventajas y desventajas que hay que tomar en cuenta a la hora de aplicarla.



# Definición recursiva de un problema

- Para garantizar que un problema se puede resolver con recursión basta plantear una definición recursiva del mismo.
- Una definición es recursiva cuando se hace en función del mismo concepto.
- Desde luego que, como se está definiendo algo en función de sí mismo, deben cumplirse ciertas condiciones para que esta redefinición no se haga infinita.
- Todas las definiciones recursivas inician (o terminan) en alguno (o algunos) caso base, cuya definición es independiente del concepto.

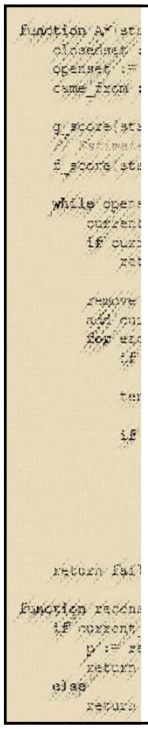


# Ej de definición recursiva: factorial

- El factorial de un número entero no negativo es el producto que se obtiene de todos los números positivos menores o iguales a él.
- El factorial se representa con el símbolo !
  5! = 5 * 4 * 3 * 2 * 1 = 120
- El factorial es una función que se puede definir en forma recursiva:

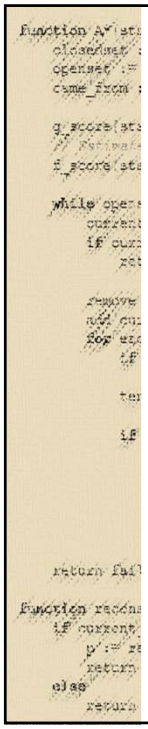
$$n! = \begin{cases} 1 & n = 0 \\ n * (n-1)! & n > 0 \end{cases}$$

5! = 5 * 4!, 4! = 4 * 3!, 3! = 3 * 2!, 2!= 2 * 1!, 1! = 1 * 0! Y 0! = 1
Observe que el 0! Ya no se define en función del factorial sino que es 1.



# Casos en la definición recursiva

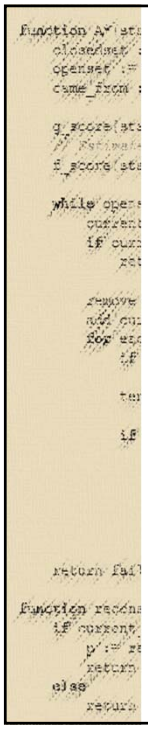
- Una definición recursiva debe contener de 2 elementos:
  - Caso base: un caso que no depende del concepto
    - En el ejemplo es el 0 ya que 0!=1
  - Caso recursivo: un caso que se define en función del mismo concepto pero para un valor más simple.
    - En el ejemplo es n! la cual es igual a n multiplicado por el factorial de un número más pequeño en una unidad (n-1)!
- Los casos base pueden ser más de 1, al igual que los casos recursivos.



# Funciones recursivas

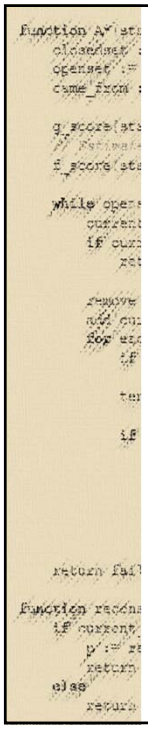
- Una vez que se tiene una definición recursiva de una función (problema) la codificación de la misma es inmediata en cualquier lenguaje de programación que soporte la recursión.

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



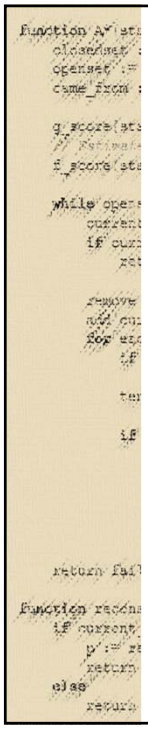
# Características de una función recursiva

- De la misma forma que las definiciones recursivas tienen 2 elementos, las funciones recursivas también tienen algunos:
  - Un condicional para saber si se trata de un caso base (condición de paro) o un caso recursivo, dependiendo del valor del argumento.
  - Si se trata de que un caso base, la función regresa un valor, sin llamar más a la función (se detienen las llamadas recursivas para empezar a calcular los resultados parciales).
  - Si es un caso recursivo, la función regresa un valor que se calcula con el argumento actual y una llamada recursiva a la misma función pero con un argumento más simple (llamada recursiva más simple).



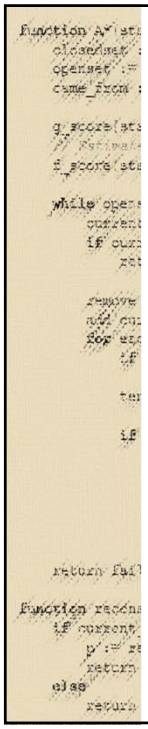
# Ejercicios para recursión

- Hacer una función recursiva que:
  - Calcule la multiplicación de dos números, utilizando solamente sumas
  - Calcule la potencia de dos números usando solamente multiplicaciones
    - Use la función de multiplicación que hizo en el punto anterior
  - Obtener el máximo número de un arreglo
  - Obtener el número de vocales en un string
  - Obtener los números de Fibonacci



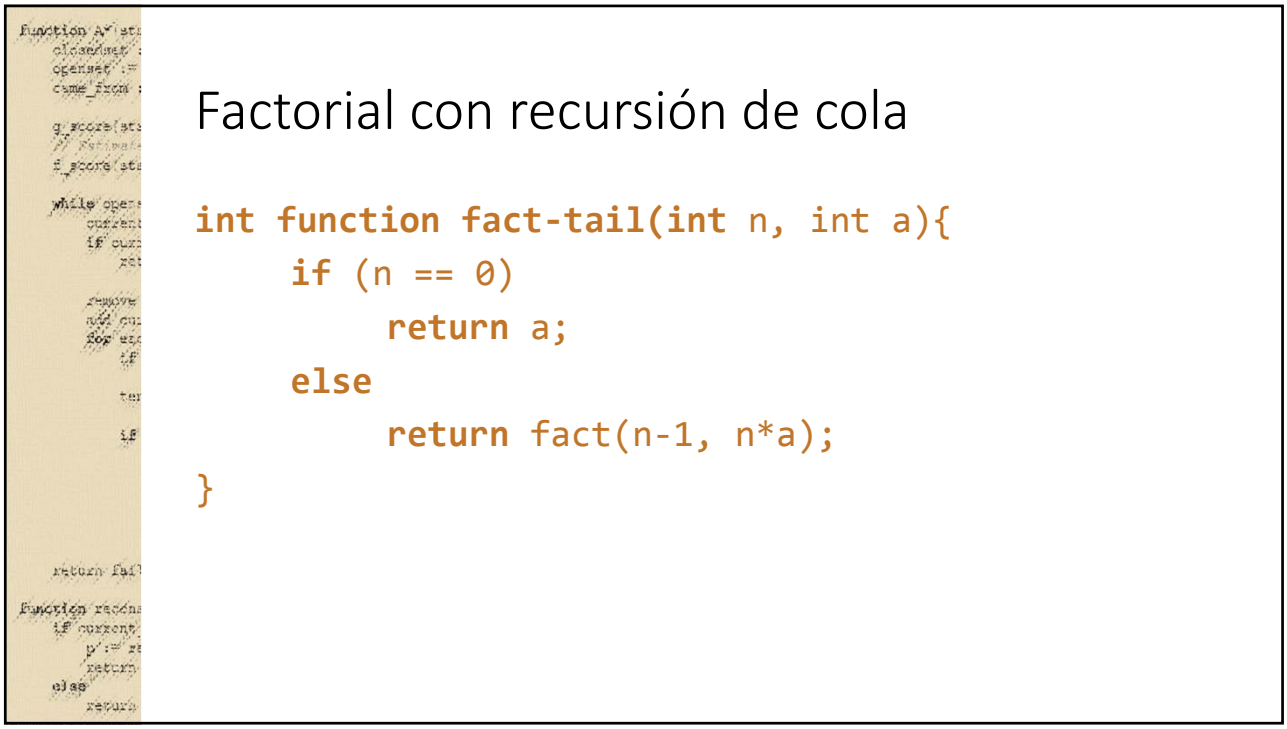
## Directa vs Indirecta

- Una función tiene recursión directa si se llama a sí misma.
- Una función que llama a otra función y eventualmente esta llama a la función original se dice que tienen recursión indirecta.
  - Si una función A llama a la función B y la función B llama a la función A, se dice que la función A tienen recursión indirecta o es indirectamente recursiva.
- La recursión indirecta puede ser de varios niveles de profundidad.
  - Si A llama a B, B llama a C, C llama a D y D llama a A, se dice que A es indirectamente recursiva y necesitó 4 niveles.
- La recursión indirecta requiere una mayor análisis para garantizar que nuestra implementación no se cicla.



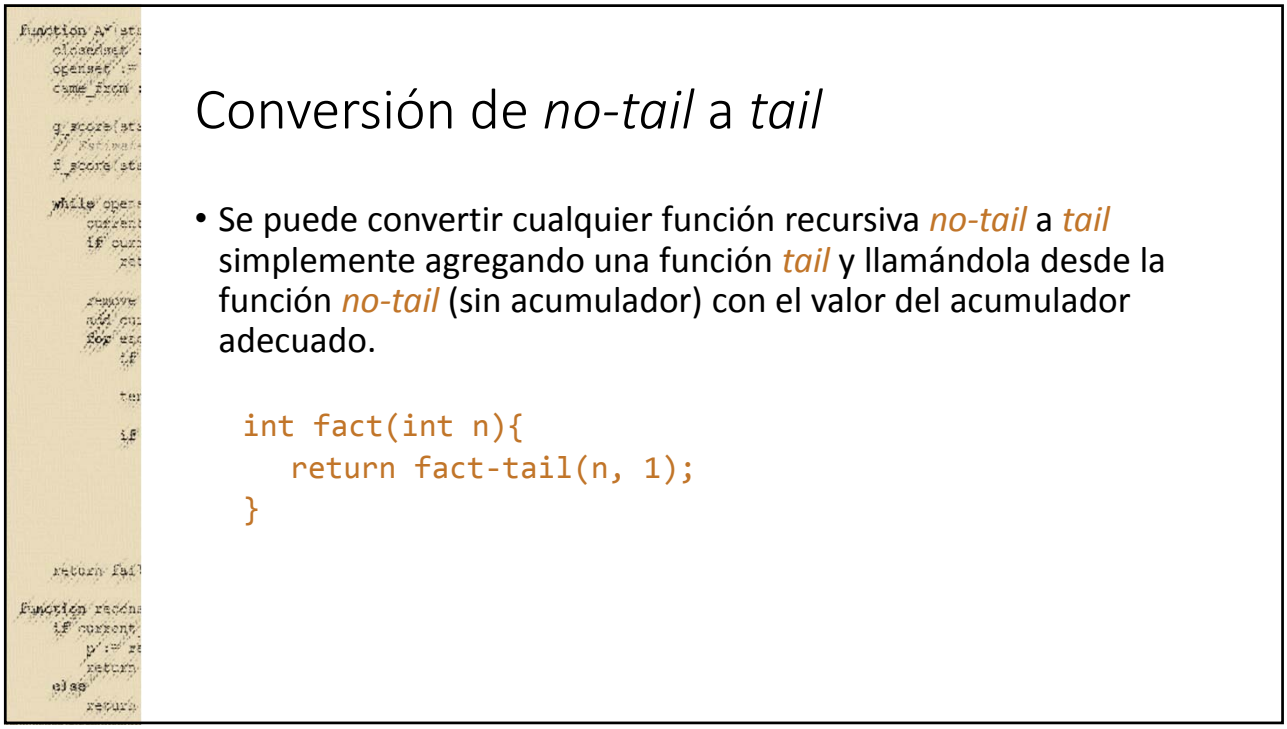
## Recursión de cola (*tail recursion*)

- Si en una función la última instrucción ejecutada es una llamada recursiva (o la de regresar un valor), se dice que tienen recursión de cola (*tail recursion*).
- La recursión de cola requiere de un acumulador que se pasa como parámetro y es el que se regresa cuando se cumple el criterio de paro.
- La recursión de cola es muy eficiente y permite que algunos compiladores transformen un código recursivo a un ciclo (iterativo) en forma automática.
  - Cuando se llama por primera ve a la función el acumulador debe tener un valor adecuado para hacer la operación (e.g. 0 si son sumas).



## Factorial con recursión de cola

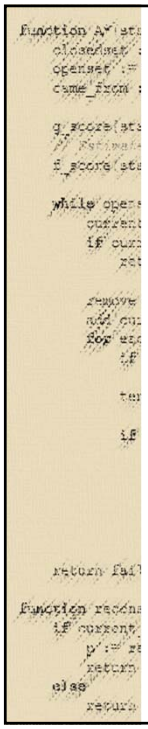
```
int function fact-tail(int n, int a){
     if (n == 0)
          return a;
     else
          return fact(n-1, n*a);
}
```



## Conversión de *no-tail* a *tail*

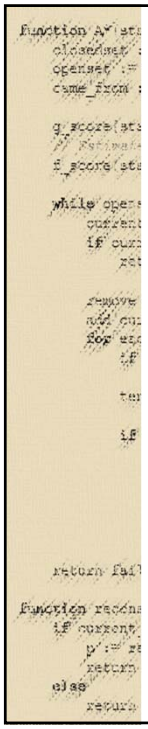
- Se puede convertir cualquier función recursiva *no-tail* a *tail* simplemente agregando una función *tail* y llamándola desde la función *no-tail* (sin acumulador) con el valor del acumulador adecuado.

```
int fact(int n){
   return fact-tail(n, 1);
}
```



# Uso de una pila en recursión

- El corazón de la recursión es una estructura de datos especial llamada PILA (la cual estudiaremos a detalle más adelante).
- Una pila es una "caja especial" para guardar cosas en la cual, las cosas se meten a la caja por la parte de arriba y se sacan también por la parte de arriba.
- Esto hace que la última cosa que metimos a la caja sea la primera que tenemos que sacar de ella (e.g. pila de platos, pila de camisas, etc).
- Cuando se llama a una función, siempre se tiene que guardar la dirección de memoria a la que se tiene que regresar cuando la función se termina de ejecutar.
- La mejor forma para guardar estas dirección es usar una pila.

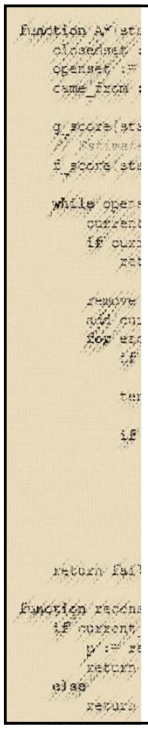


# Proyecto 2: Recursión

Haga 4 funciones (sólo funciones sin clase) recursivas:

- Dado un string, regresar el string "limpio", lo que significa que si hay caracteres adyacentes iguales, sólo deja uno de ellos.
  - Por ejemplo, "aabbbbccccccdd", debe regresar "abc".
- Contar el número de veces que un substring dado aparece en un string.
  - E.g. "ab" en el string "abbaaabba", debe regresar un 2.
- Dado un entero no negativo, regresar la suma de sus dígitos.
  - E.g. "12345" debe regresar 15.
- Dado un string formado por paréntesis anidados, regresar TRUE si están anidados correctamente o FALSE en caso contrario.
  - E.g. "(())" regresa TRUE, "((())" regresa FALSE.

Fuente: http://codingbat.com/java/Recursion-1



# Referencias

[1] Elliot B. Koffman. Objects, abstraction, data structures, and design: using C++. John Wiley & Sons Inc. (2005).

[2] Silvia Guardati. Estructura de datos orientada a objetos: Algoritmos con C++. Pearson (2007).