Recursión

Carlos E. Alvarez[1].

[1]Dep. de Matemáticas aplicadas y Ciencias de la Computación, Universidad del Rosario

2019-II

Recursión

Ejemplo iterativo

```
1 DEFINE_NEW_INSTRUCTION face-north AS
2 BEGIN
3   WHILE not-facing-north DO
4   BEGIN
5     turnleft
6   END
7 END;
```

Recursión

Ejemplo recursivo

```
1 DEFINE_NEW_INSTRUCTION face-north AS
2 BEGIN
3   IF not-facing-north THEN
4   BEGIN
5     turnleft;
6     face-north
7   END
8 END;
```

Recursión

Paradigma de la recursión

```
1 type recursive_function(params){
2   if(testForBaseCase){
3     Compute solution without recursion;
4   }else{
5     Do the reduction;
6     recursive_function();
7   }
8 }
```

Recursión: Función factorial

Factorial (iterativo)

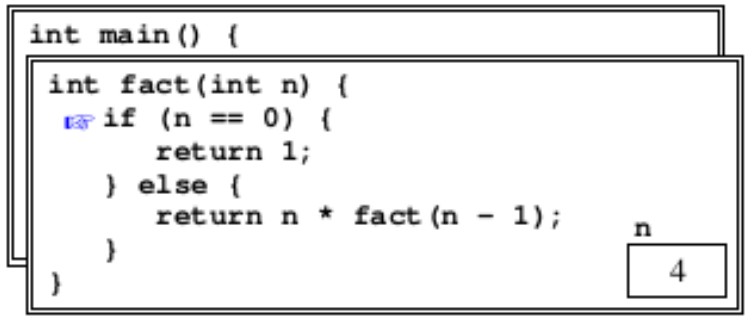
```
1 int factorial(int n){
2   int result = 1;
3   for(int i = 1; i <= n; i++){
4     result *= i;
5   }
6   return result;
7 }
```

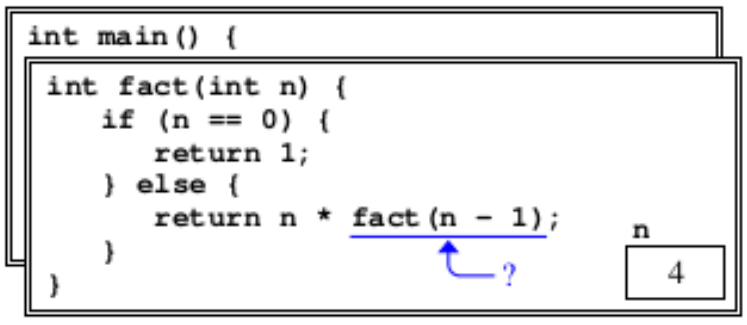
Recursión: Función factorial

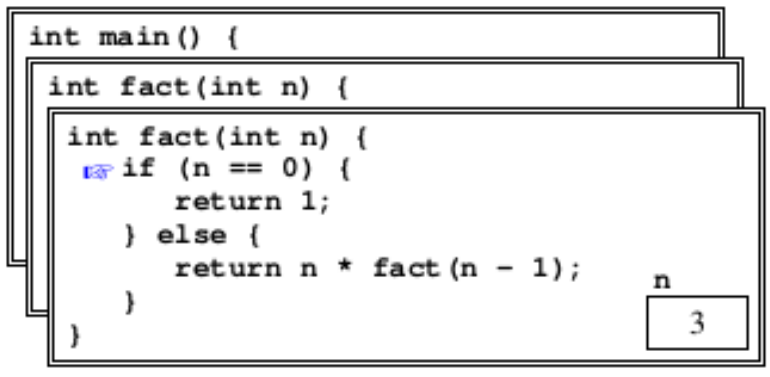
Factorial (recursivo)

```
1 int factorial(int n){
2   if(n == 0){
3     return 1;
4   }else{
5     return n * factorial(n-1);
6   }
7 }
```

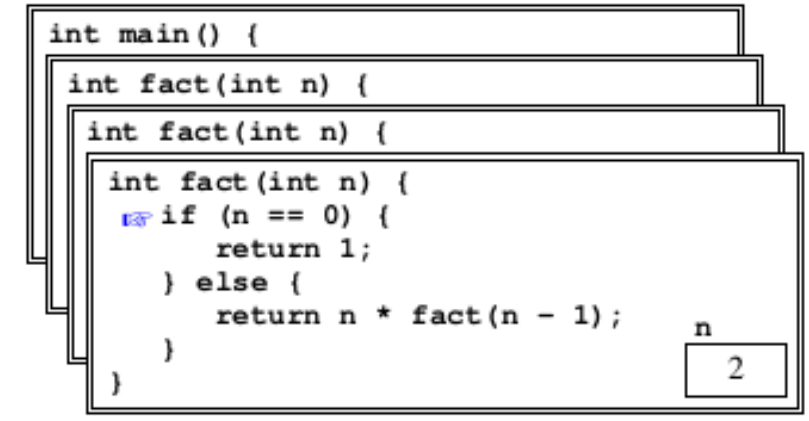
Recursión: Función factorial

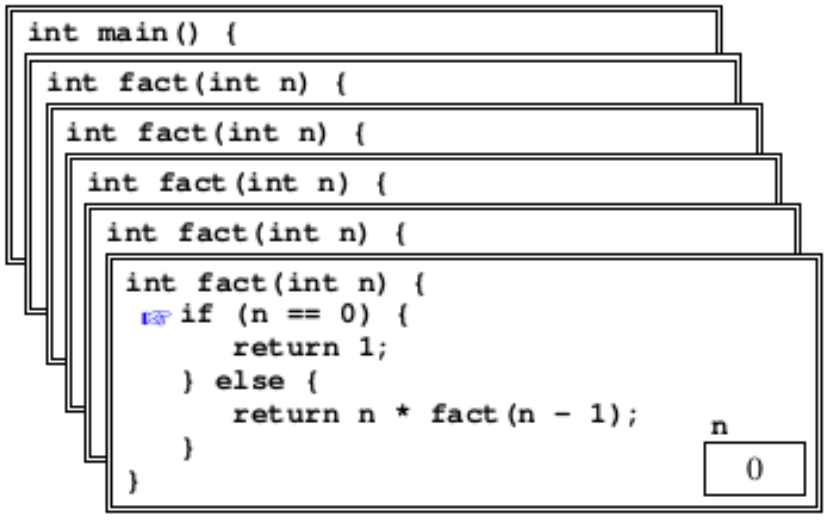




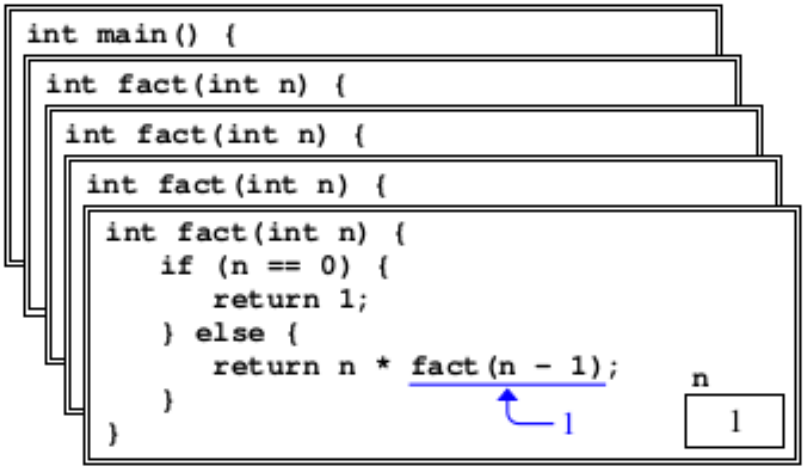


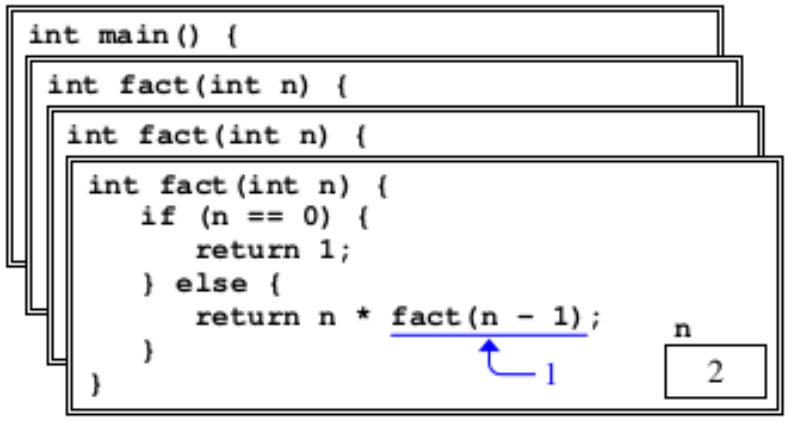
Recursión: Función factorial



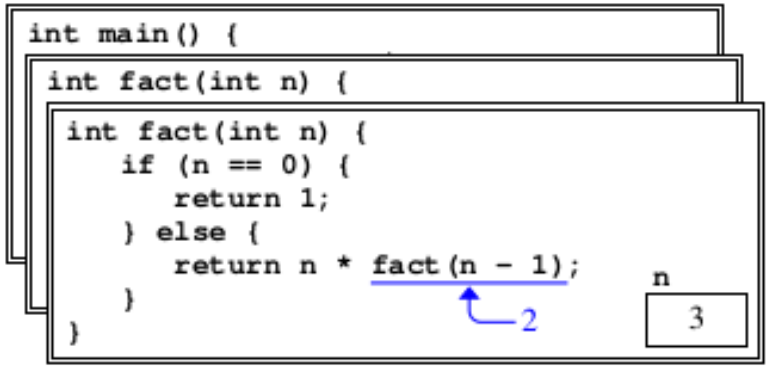


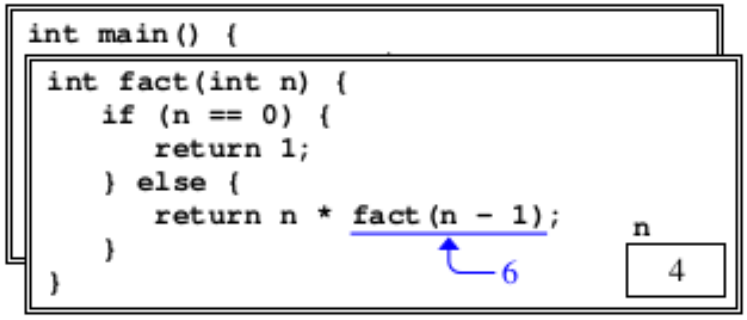
Recursión: Función factorial





Recursión: Función factorial





Recursión: Fibonacci

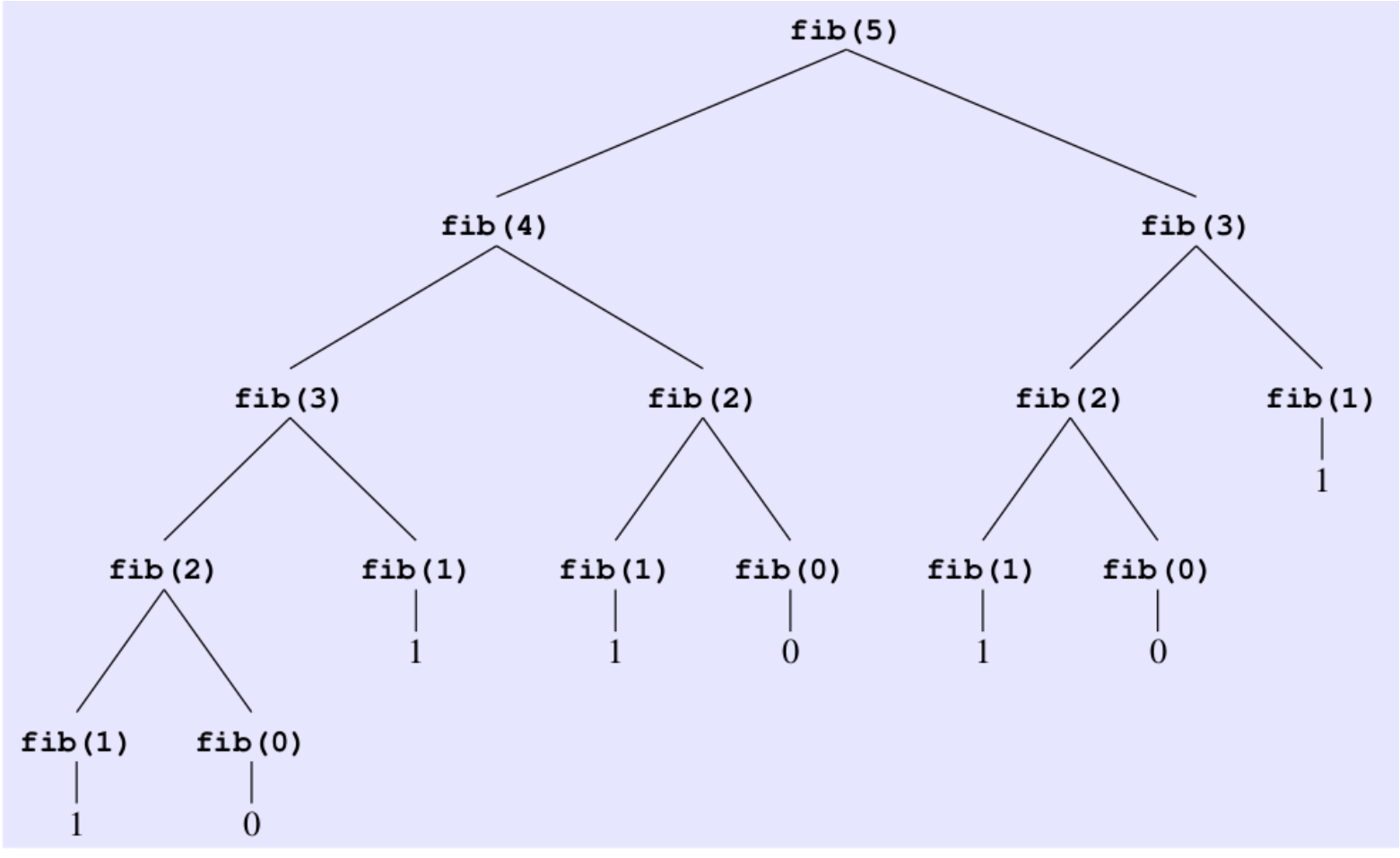
Mes	# parejas	fert/infert
Feb	1	1i+0f
Mar	2	1i+1f
Abr	3	1i+2f
May	5	2i+3f
Jun	8	3i+5f
Jul	13	5i+8f
⋮	⋮	⋮

Recursión: Fibonacci

$$f(n) = \begin{cases} n & , n = 0, 1 \\ f(n-1) + f(n-2) & , n > 1 \end{cases}$$

```
1 int fibonacci(int n){
2   if(n < 2){
3     return n;
4   }else{
5     return fibonacci(n-1) + fibonacci(n-2);
6   }
7 }
```

Recursión: Fibonacci



Roberts, Eric. (2013). *Programming Abstractions in C++*. Pearson.

Recursión: Secuencias aditivas

$$f(n) = \begin{cases} f_0 & , n = 0 \\ f_1 & , n = 1 \\ f(n-1) + f(n-2) & , n > 1 \end{cases}$$

```
1 int additiveSeq(int f0, int f1, int n){
2   if(n == 0) return f0;
3   else if(n == 1) return f1;
4   else{
5     return additiveSeq(f1, f0+f1, n-1);
6   }
7 }
```

Recursión: Secuencias aditivas

Versión mas eficiente de la sec. de Fibonacci:

```
1 int fibonacci(int n){
2   return additiveSeq(0, 1, n);
3 }
```

Recursión: Secuencias aditivas

Ejercicio:

1. Programe y compare los tiempos de ejecución de las dos versiones de `fibonacci`

Recursión: Palíndromos

Una cadena que se lee igual de izquierda a derecha que de derecha a izquierda.

Implementación A

```
1 bool isPalindrome(string str){
2   int len = str.length();
3   if(len <= 1){
4     return true;
5   }else{
6     return (str[0]==str[len-1])
7         && isPalindrome(str.substr(1,len-2));
8   }
9 }
```

Recursión: Palíndromos

Implementación B

```
1 bool isSubstringPalindrome(string str, int p1, int p2){
2   if(p1 >= p2){
3     return true;
4   }else{
5     return (str[p1]==str[p2])
6         && isSunstringPalindrome(str, p1+1, p2-1);
7   }
8 }
```

Implementación B

```
1 bool isPalindrome(string str){
2   return isSubstringPalindrome(str, 0, str.length()-1);
3 }
```

Recursión: Suma del subconjunto

Solución mala con tiempo $O(N!)$

```
1  bool subsetSumExists(set<int>& mySet, int target){
2    set<int>::iterator it;
3    if(!mySet.empty()){
4      int add = sum(mySet);
5
6      bool condition = (add-target == 0);
7      for(it = mySet.begin(); it != mySet.end(); ++it){
8        set<int> redSet(mySet.begin(), mySet.end());
9        redSet.erase(*it);
10       condition = condition
11                   || subsetSumExists(redSet, target);
12     }
13     return condition;
14   }
15 }
```

Recursión: Suma del subconjunto

Mejor solución $O(2^N)$

```
1  bool subsetSumExists(set<int> mySet, int target){
2    if(mySet.empty()){
3      return target == 0;
4    }else{
5      set<int>::iterator first = mySet.begin();
6
7      set<int> redSet(mySet.begin(), mySet.end());
8      redSet.erase(*first);
9
10     return subsetSumExists(redSet, target)
11            || subsetSumExists(redSet, target-*first);
12   }
13 }
```