ΗΥ-150 Προγραμματισμός
CS-150 Programming

Recursion

G. Papagiannakis

University of Crete

Recursively Defined functions

- For some problems, it's useful to have functions *call themselves*
- As often it is difficult to express the members of an object or numerical sequence explicitly.

e.g.: The **Fibonacci** sequence:

$\{f_n\} = 0,1,1,2,3,5,8,13,21,34,55,\ldots$

- There may, however, be some "local" connections that can give rise to a ***recursive definition*** –a formula that expresses higher terms in the sequence, in terms of lower terms.

e.g.: Recursive definition for $\{f_n\}$:

INITIALIZATION: $f_0 = 0, f_1 = 1$

RECURSION: $f_n = f_{n-1} + f_{n-2}$ for $n > 1$.

Recursive Definitions and Induction

- Recursive definition and inductive proofs are complement each other: a recursive definition usually gives rise to natural proofs involving the recursively defined sequence.
- This is follows from the format of a recursive definition as consisting of two parts:
- **Initialization** –analogous to induction **base cases**
- **Recursion** –analogous to **induction step**
- In both induction and recursion, the domino analogy is useful.

Recursion

- We must always make sure that the recursion *bottoms out*:
  - A recursive function must contain at least one non-recursive branch.
  - The recursive calls must eventually lead to a non-recursive branch.
- Recursion is one way to decompose a task into smaller subtasks. At least one of the subtasks is a smaller example of the same task.
- The smallest example of the same task has a non-recursive solution.
- Fibonacci numbers:

  `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...`

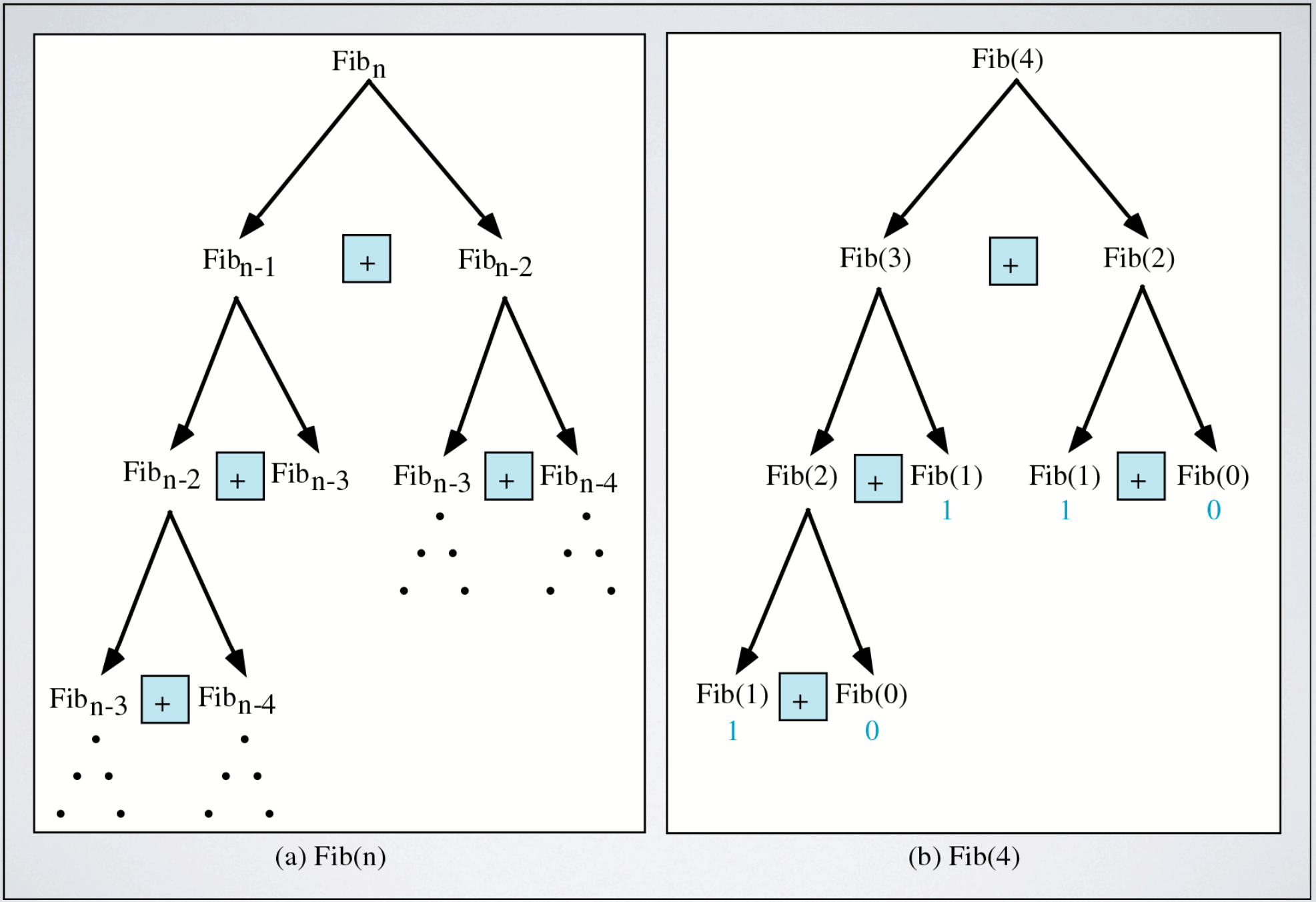
  where each number is the sum of the preceding two.
- Recursive definition:
  - `F(0) = 0;`
  - `F(1) = 1;`
  - `F(number) = F(number-1)+ F(number-2);`

Recursive Example: Fibonacci numbers

```
//Calculate Fibonacci numbers using recursive function.
//A very inefficient way, but illustrates recursion well
int fib(int number)
{
  if (number == 0) return 0;
  if (number == 1) return 1;
  return (fib(number-1) + fib(number-2));
}

int main(){     // driver function
  int inp_number=0;
  cout << "Please enter an integer: ";
  cin >> inp_number;
  cout << "The Fibonacci number for "<< inp_number
       << " is "<< fib(inp_number)<<endl;
  return 0;
}
```

```
f(0) is 0
f(1) is 1
f(2) is 1
f(3) is 2
f(4) is 3
f(5) is 5
f(6) is 8
```


Fib$_n$
Fib$_{n-1}$
+
Fib$_{n-2}$
Fib$_{n-2}$
+
Fib$_{n-3}$
Fib$_{n-3}$
+
Fib$_{n-4}$
Fib$_{n-3}$
+
Fib$_{n-4}$
(a) Fib(n)
Fib(4)
Fib(3)
+
Fib(2)
Fib(2)
+
Fib(1)
1
Fib(1)
+
Fib(0)
0
Fib(1)
1
+
Fib(0)
0
(b) Fib(4)

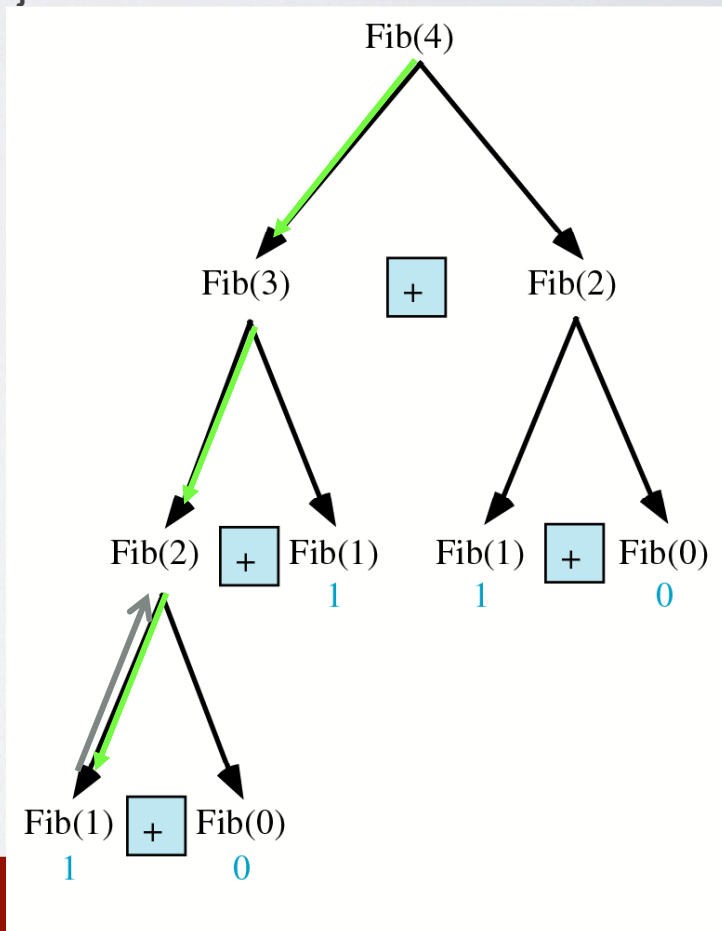
Trace a Fibonacci Number

- Assume the input number is 4, that is, num=4:

```
fib(4):
  4 == 0 ? No;    4 == 1?     No.
  fib(4) = fib(3) + fib(2)
  fib(3):
    3 == 0 ? No; 3 == 1? No.
    fib(3) = fib(2) + fib(1)
    fib(2):
      2 == 0? No; 2==1? No.
      fib(2) = fib(1)+fib(0)
       fib(1):
          1== 0 ? No; 1 == 1? Yes.
              fib(1) = 1;
      return fib(1);
```

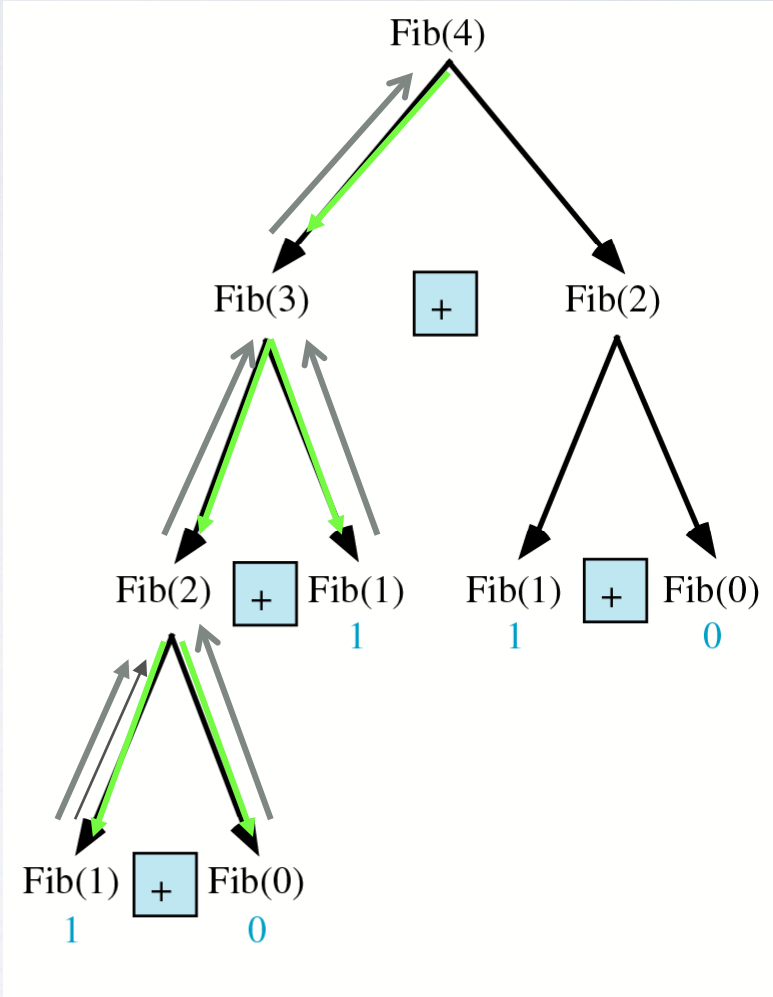
```
int fib(int num)
{
   if (num == 0) return 0;
   if (num == 1) return 1;
   return

   (fib(num-1)+fib(num-2));
}
```



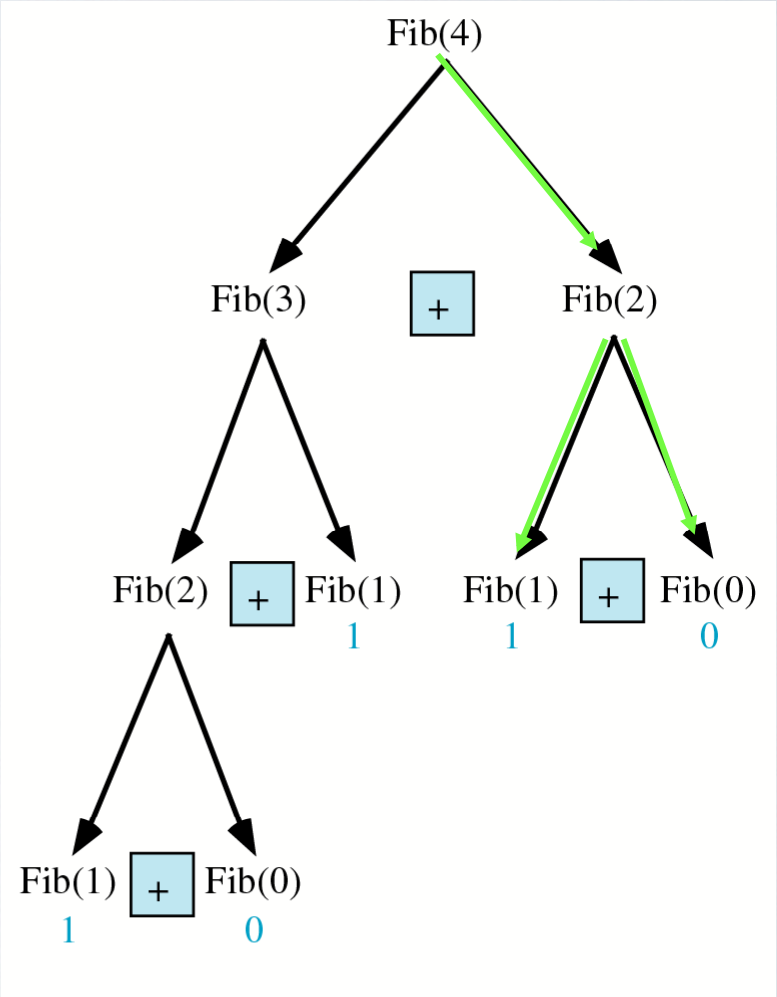
Trace a Fibonacci Number

```
fib(0):
   0 == 0 ?  Yes.
   fib(0) = 0;
   return fib(0);
fib(2) = 1 + 0 = 1;
return fib(2);
fib(3) = 1 + fib(1)
fib(1):
  1 == 0 ? No; 1 == 1? Yes
  fib(1) = 1;
  return fib(1);
fib(3) = 1 + 1 = 2;
return fib(3)
```



Trace a Fibonacci Number

```
fib(2):
2 == 0 ? No; 2 == 1?   No.
fib(2) = fib(1) + fib(0)
fib(1):
   1== 0 ? No; 1 == 1?  Yes.
       fib(1) = 1;
return fib(1);
 fib(0):
     0 == 0 ?   Yes.
       fib(0) = 0;
       return fib(0);
 fib(2) = 1 + 0 = 1;
 return fib(2);
fib(4) = fib(3) + fib(2)
       = 2 + 1 = 3;
return fib(4);
```



Fibonacci number w/o recursion

```
//Calculate Fibonacci numbers iteratively
//much more efficient than recursive solution


int fib(int n)
{
 int f[n+1];
 f[0] = 0; f[1] = 1;
  for (int i=2; i<= n; i++)
       f[i] = f[i-1] + f[i-2];
 return f[n];
}
```

Recursive example: factorial calculation

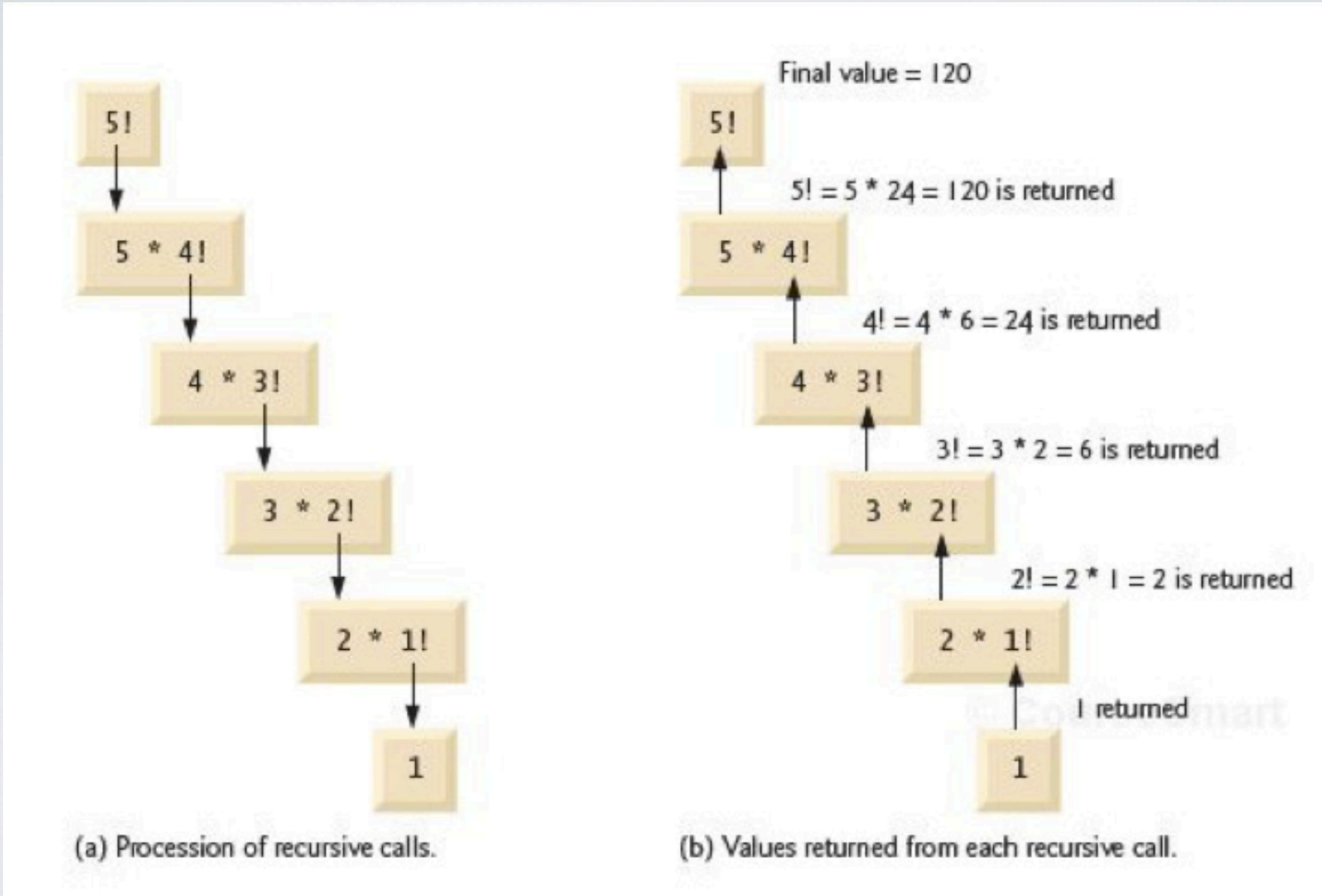
- n! = n*(n-1)*(n-2)*….1
- Iterative solution: (e.g. 5!= 5*4*3*2*1)

```
factorial = 1;
for ( int counter = number; counter >= 1; counter-- )
   factorial *= counter;
```

- Recursive solution: (e.g. 5!=5*4! i.e. n!=n*(n-1)!)

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$
$$5! = 5 \cdot (4 \cdot 3 \cdot 2 \cdot 1)$$
$$5! = 5 \cdot (4!)$$

Recursive example: factorial calculation II



Recursive example: factorial calculation III

```
#include <iostream>
#include <iomanip>
using namespace std;

unsigned long factorial( unsigned long ); // function prototype

int main()
{
   // calculate the factorials of 0 through 10
   for ( int counter = 0; counter <= 10; counter++ )
      cout << setw( 2 ) << counter << "! = " << factorial( counter )
         << endl;
} // end main

// recursive definition of function factorial
unsigned long factorial( unsigned long number )
{
   if ( number <= 1 ) // test for base case
      return 1; // base cases: 0! = 1 and 1! = 1
   else // recursion step
      return number * factorial( number - 1 );
} // end function factorial
```

Thank you!