
CS1371

Introduction to Computing for Engineers

Recursion

Background

- We will review dynamic memory allocation
- We will start with the familiar iterative approach
 - *This works when processing simple, linear collections*
 - *It will not work at all on the more complex structures we will meet later in the course*
- Consequently, while the structures are still simple, we will introduce the engineers' worst nightmare:

Recursion

The Trick to Recursive Programming

- Solving a problem involves first analyzing the given information
- The most elegant solutions emerge when the solution models the structure of the data
- Examples:
 - *Searching the WWW for images*
 - *Searching a file system for a specific file*
 - *Numerical problems with recursive definitions*

Solving a problem in terms of itself.

- **wget**
 - *Given a URL*
 - *Read in the file*
 - *Save it to disk*
 - *Scan through file*
 - **Find any JPG or GIF references then save them**
 - **Find any HTML references then call **wget****

Solving a problem in terms of itself

- **ShowMe(directory)**
 - *Read the directory*
 - *Process each entry*
 - *If an entry is a file then*
 - Print it on the screen
 - *Else // It's a directory!!!*
 - ShowMe(the Entry)

Solving a problem in terms of itself

n!

Is it:

n * (n-1) * (n-2) * ... 3 * 2 * 1

or

n * (n - 1)!

Application to Iterative Solutions

- If your data are in an array, the description of the structure might be:
 - *An array of items is a collection[1..n] of item*
- From which, we naturally produce an iterative solution:
 - *for[$l = 1 \dots n$] do-something-with(array(i))*

Recursion Defined

- A **procedure or function** which **calls itself**.
- Powerful mechanism for **repetition**.
- Makes algorithms more **compact** and simple.
- Module calls a “**clone**” of itself.
- Very useful –
 - *For numerical problems with recursive definitions.*
 - *For **dynamic data types***

Three Characteristics of Recursion

1. The function calls itself recursively
2. The function has some terminating condition
3. The function parameters move the solution “closer” to the terminating condition.

Know this!!!

Another Recursive Procedure

Problem: Count from N to 10.

```
function CountToTen(count)
    if count <= 10
        disp( count );
        CountToTen( count + 1 );
    end
```

Call the procedure
`CountToTen(7);`

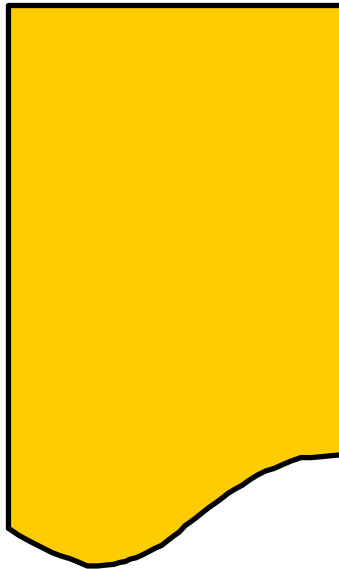
Tracing The Recursion

- To keep track of recursive execution, do what a computer does: maintain information on an **activation stack**.
- Each stack frame contains:
 - *Module identifier*
 - *Variable values*

ModuleID: Data values

```
function CountToTen(7)
    if 7 <= 10
        disp( 7 );
        CountToTen( 8 );
    end
```

CountToTen: **count=7**



```
function CountToTen(7)
    if 7 <= 10
        disp( 7 );
        CountToTen( 8 );
    end
```

CountToTen: count=7

7

```
function CountToTen(7)
    if 7 <= 10
        disp( 7 );
        CountToTen( 8 );
    end
```

CountToTen: count=7

7

```
function CountToTen(8)
    if 8 <= 10
        disp( 8 );
        CountToTen( 9 );
    end
```

CountToTen: count=8

CountToTen: count=7

7

```
function CountToTen(8)
    if 8 <= 10
        disp( 8 );
        CountToTen( 9 );
    end
```

CountToTen: count=8

CountToTen: count=7

7
8

```
function CountToTen(8)
    if 8 <= 10
        disp( 8 );
        CountToTen( 9 );
    end
```

CountToTen: count=8

CountToTen: count=7

7
8

```
function CountToTen(9)
    if 9 <= 10
        disp( 9 );
        CountToTen( 10 );
    end
```

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

7
8

```
function CountToTen(9)
    if 9 <= 10
        disp( 9 );
        CountToTen( 10 );
    end
```

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

7
8
9

```
function CountToTen(9)
    if 9 <= 10
        disp( 9 );
        CountToTen( 10 );
    end
```

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

7
8
9

```
function CountToTen(10)
    if 10 <= 10
        disp( 10 );
        CountToTen( 11 );
    end
```

CountToTen: **count=10**

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

7
8
9

```
function CountToTen(10)
    if 10 <= 10
        disp( 10 );
        CountToTen( 11 );
    end
```

CountToTen: count=10

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

7
8
9
10

```
function CountToTen(10)
    if 10 <= 10
        disp( 10 );
        CountToTen( 11 );
    end
```

CountToTen: count=10

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

7
8
9
10

```
function CountToTen(11)
    if 11 <= 10
        disp( 11 );
        CountToTen( 11 );
    end
```

CountToTen: count=11

CountToTen: count=10

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

7
8
9
10

```
function CountToTen(10)
    if 10 <= 10
        disp( 10 );
        CountToTen( 11 );
    end
```

CountToTen: count=10

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

**7
8
9
10**

```
function CountToTen(9)
    if 9 <= 10
        disp( 9 );
        CountToTen( 10 );
    end
```

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

7
8
9
10

```
function CountToTen(8)
    if 8 <= 10
        disp( 8 );
        CountToTen( 9 );
    end
```

CountToTen: count=8

CountToTen: count=7

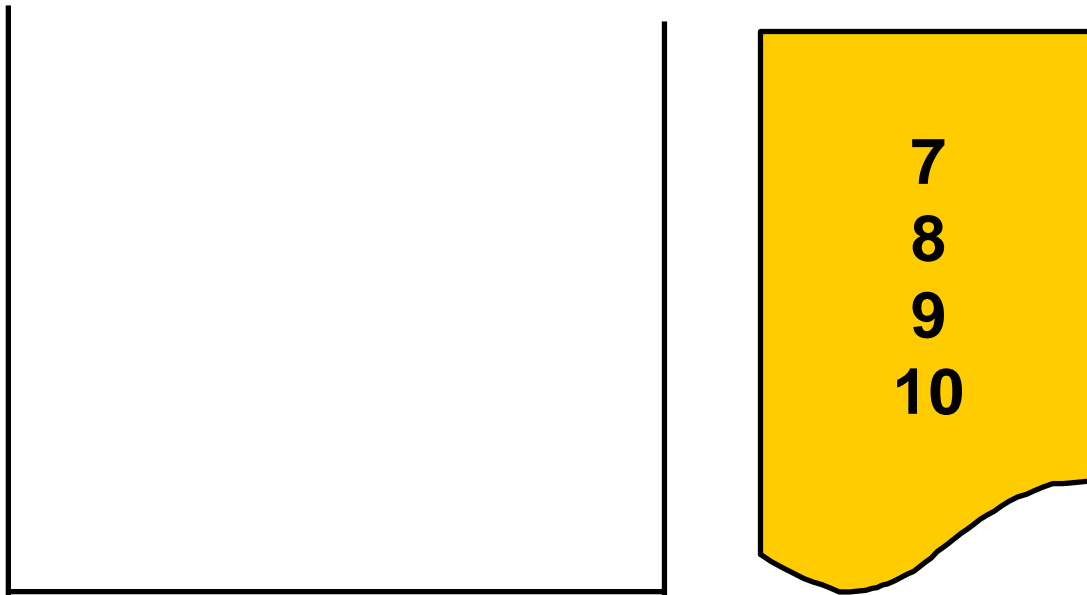
7
8
9
10

```
function CountToTen(7)
    if 7 <= 10
        disp( 7 );
        CountToTen( 8 );
    end
```

CountToTen: count=7

7
8
9
10

-
- Return to the script.



Reversing the Work and Recursion

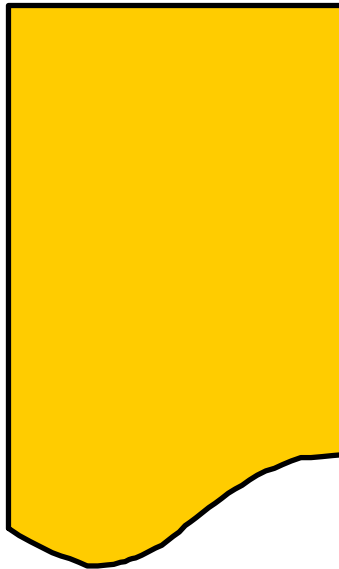
Problem: Count from 10 to N.

```
function CountToTen(count)
    if count <= 10
        CountToTen( count + 1 );
        disp( count );
    end
```

Now the work will happen as the frames pop off the stack!

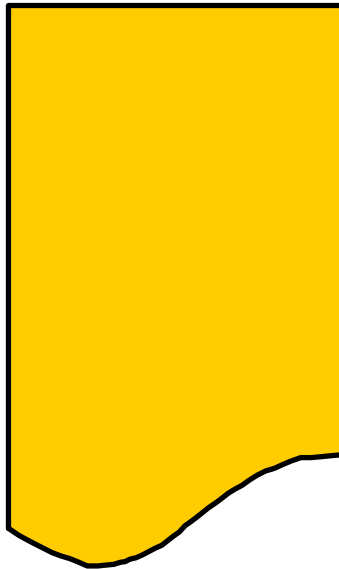
```
function CountToTen(7)
    if 7 <= 10
        CountToTen( 8 );
        disp( 7 );
    end
```

CountToTen: count=7



```
function CountToTen(7)
    if 7 <= 10
        CountToTen( 8 );
        disp( 7 );
    end
```

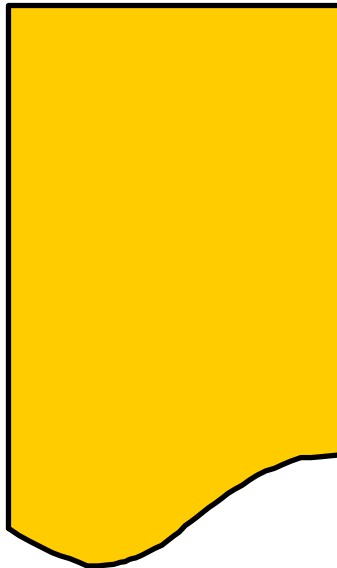
CountToTen: count=7



```
function CountToTen(8)
    if 8 <= 10
        CountToTen( 9 );
        disp( 8 );
    end
```

CountToTen: count=8

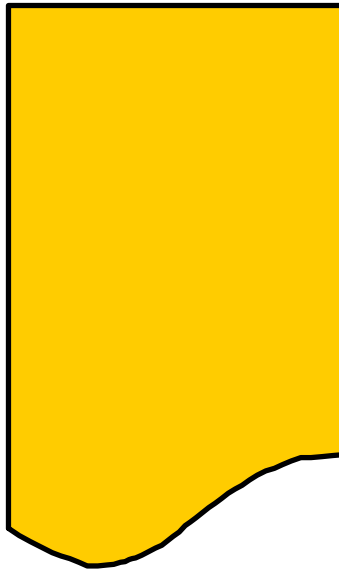
CountToTen: count=7



```
function CountToTen(count)
    if 8 <= 10
        CountToTen( 9 );
        disp( 8 );
    end
```

CountToTen: count=8

CountToTen: count=7

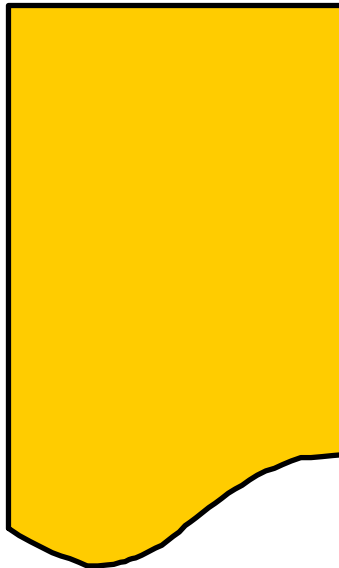


```
function CountToTen(9)
    if 9 <= 10
        CountToTen( 10 );
        disp( 9 );
    end
```

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

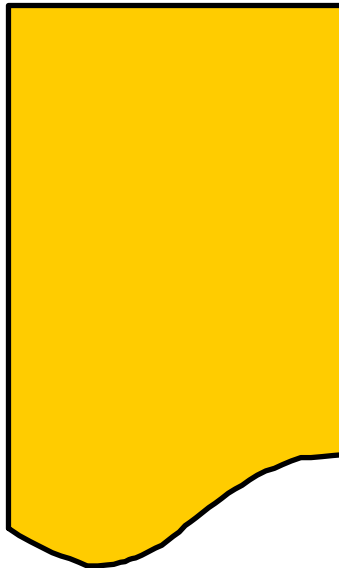


```
function CountToTen(9)
    if 9 <= 10
        CountToTen( 10 );
        disp( 9 );
    end
```

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7



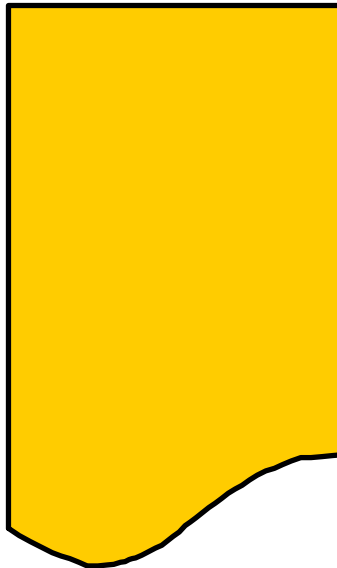
```
function CountToTen(10)
    if 10 <= 10
        CountToTen( 11 );
        disp( 10 );
    end
```

CountToTen: count=10

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7



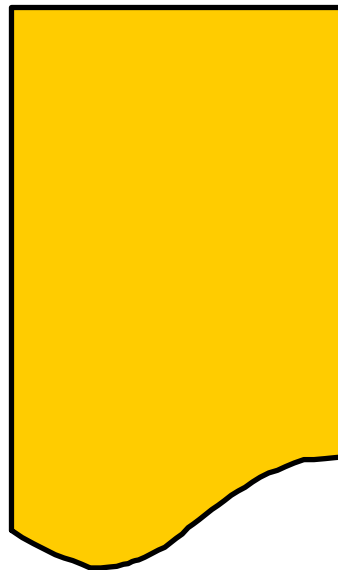
```
function CountToTen(10)
    if 10 <= 10
        CountToTen( 11 );
        disp( 10 );
    end
```

CountToTen: count=10

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7



```
function CountToTen(11)
    if 11 <= 10
        CountToTen( 12 );
        disp( 11 );
    end
```

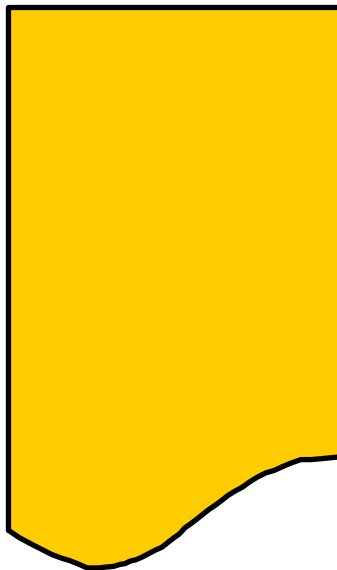
CountToTen: count=11

CountToTen: count=10

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7



```
function CountToTen(10)
    if 10 <= 10
        CountToTen( 11 );
        disp( 10 );
    end
```

CountToTen: count=10

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

10

```
function CountToTen(10)
    if 10 <= 10
        CountToTen( 11 );
        disp( 10 );
    end
```

CountToTen: count=10

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

10

```
function CountToTen(9)
    if 9 <= 10
        CountToTen( 10 );
        disp( 9 );
    end
```

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

10
9

```
function CountToTen(9)
    if 9 <= 10
        CountToTen( 10 );
        disp( 9 );
    end
```

CountToTen: count=9

CountToTen: count=8

CountToTen: count=7

10
9

```
function CountToTen(8)
    if 8 <= 10
        CountToTen( 9 );
        disp( 8 );
    end
```

CountToTen: count=8

CountToTen: count=7

10
9
8

```
function CountToTen(8)
    if 8 <= 10
        CountToTen( 9 );
        disp( 9 );
    end
```

CountToTen: count=8

CountToTen: count=7

10
9
8

```
function CountToTen(7)
    if 7 <= 10
        CountToTen( 8 );
        disp( 7 );
    end
```

CountToTen: count=7

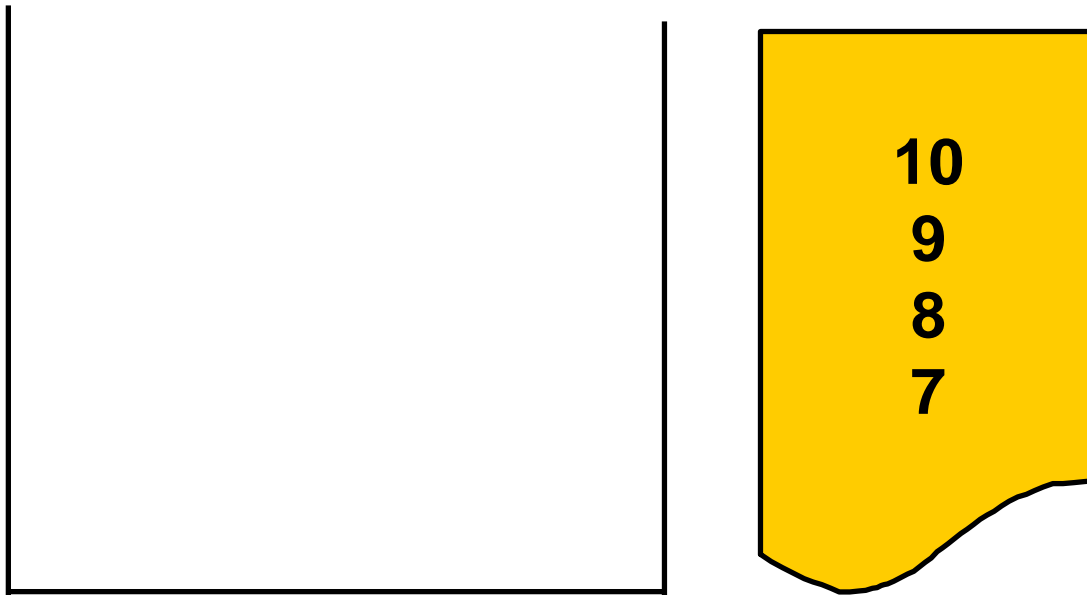
10
9
8
7

```
function CountToTen(7)
    if 7 <= 10
        CountToTen( 8 );
        disp( 7 );
    end
```

CountToTen: count=7

10
9
8
7

-
- Return to the script.



Questions?

Recursive Function Example: Factorial

Problem: calculate $n!$ (*n factorial*)

$$n! = 1 \quad \text{if } n = 0$$

$$n! = 1 * 2 * 3 * \dots * n \quad \text{if } n > 0$$

Recursively:

$$\text{if } n = 0, \text{ then } n! = 1$$

$$\text{if } n > 0, \text{ then } n! = n * (n-1)!$$

Solving Recursive Problems

- See recursive solutions as two sections:

- *Current*

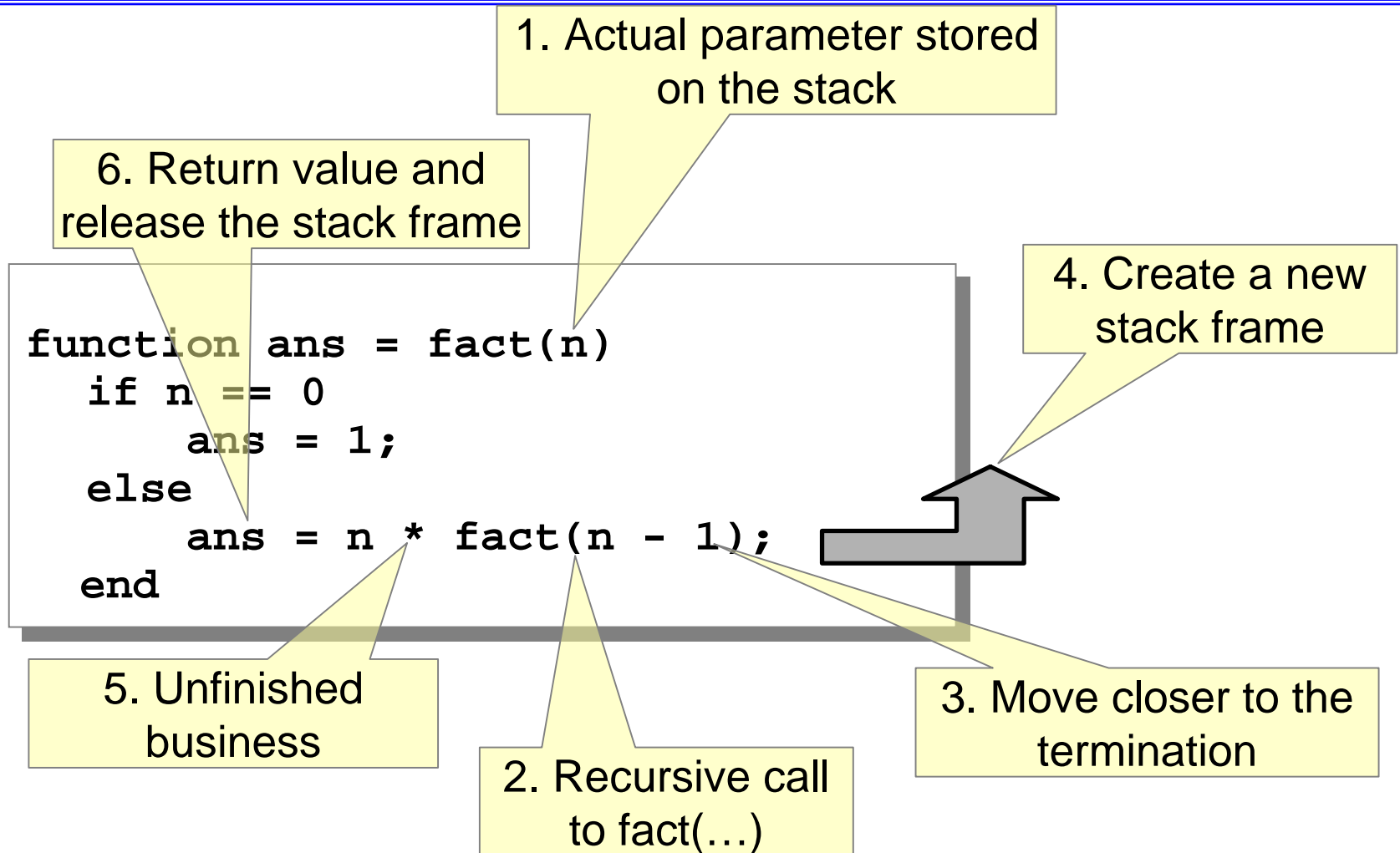
- *Rest*

- $N! = N * (N-1)!$

- $7! = 7 * 6!$

- $7! = 7 * (6 * 5 * 4 * 3 * 2 * 1 * 1)$

Tracing Details



Activation Stack for Factorial

Call the function: `answer = fact(5);`

main program:	Unfinished: <code>answer = fact(5)</code>
---------------	---

Activation Stack for Factorial

fact. 1st: N=5,

Unfinished: 5*fact(4)

main program:

Unfinished: answer = fact (5)

Activation Stack for Factorial

fact. 2nd: N=4,

Unfinished: $4 * \text{fact}(3)$

fact. 1st: N=5,

Unfinished: $5 * \text{fact}(4)$

main program:

Unfinished: answer = fact (5)

Activation Stack for Factorial

fact. 3rd: N=3,	Unfinished: 3*fact(2)
------------------------	------------------------------

fact. 2nd: N=4,	Unfinished: 4*fact(3)
------------------------	------------------------------

fact. 1st: N=5,	Unfinished: 5*fact(4)
------------------------	------------------------------

main program:	Unfinished: answer = fact (5)
----------------------	--------------------------------------

Activation Stack for Factorial

fact. 4th: N=2,	Unfinished: 2*fact(1)
------------------------	------------------------------

fact. 3rd: N=3,	Unfinished: 3*fact(2)
------------------------	------------------------------

fact. 2nd: N=4,	Unfinished: 4*fact(3)
------------------------	------------------------------

fact. 1st: N=5,	Unfinished: 5*fact(4)
------------------------	------------------------------

main program:	Unfinished: answer = fact (5)
----------------------	--------------------------------------

Activation Stack for Factorial

fact. 5th: N=1,	Unfinished: 1*fact(0)
------------------------	------------------------------

fact. 4th: N=2,	Unfinished: 2*fact(1)
------------------------	------------------------------

fact. 3rd: N=3,	Unfinished: 3*fact(2)
------------------------	------------------------------

fact. 2nd: N=4,	Unfinished: 4*fact(3)
------------------------	------------------------------

fact. 1st: N=5,	Unfinished: 5*fact(4)
------------------------	------------------------------

main program:	Unfinished: answer = fact (5)
----------------------	--------------------------------------

Activation Stack for Factorial

fact. 6th: N=0,	Finished: returns 1
fact. 5th: N=1,	Unfinished: 1*fact(0)
fact. 4th: N=2,	Unfinished: 2*fact(1)
fact. 3rd: N=3,	Unfinished: 3*fact(2)
fact. 2nd: N=4,	Unfinished: 4*fact(3)
fact. 1st: N=5,	Unfinished: 5*fact(4)
main program:	Unfinished: answer = fact (5)

Activation Stack for Factorial

fact. 5th: N=1,	Finished: returns 1*1
------------------------	------------------------------

fact. 4th: N=2,	Unfinished: 2*fact(1)
------------------------	------------------------------

fact. 3rd: N=3,	Unfinished: 3*fact(2)
------------------------	------------------------------

fact. 2nd: N=4,	Unfinished: 4*fact(3)
------------------------	------------------------------

fact. 1st: N=5,	Unfinished: 5*fact(4)
------------------------	------------------------------

main program:	Unfinished: answer = fact (5)
----------------------	--------------------------------------

Activation Stack for Factorial

fact. 4th: N=2,

Finished: returns 2*1

fact. 3rd: N=3,

Unfinished: 3*fact(2)

fact. 2nd: N=4,

Unfinished: 4*fact(3)

fact. 1st: N=5,

Unfinished: 5*fact(4)

main program:

Unfinished: answer = fact (5)

Activation Stack for Factorial

fact. 3rd: N=3,

Finished: returns $3*2$

fact. 2nd: N=4,

Unfinished: $4*fact(3)$

fact. 1st: N=5,

Unfinished: $5*fact(4)$

main program:

Unfinished: answer = fact (5)

Activation Stack for Factorial

fact. 2nd: N=4,

Finished: returns 4*6

fact. 1st: N=5,

Unfinished: 5*fact(4)

main program:

Unfinished: answer = fact (5)

Activation Stack for Factorial

Fact. 1st: N=5,

Finished: returns 5*24

main program:

Unfinished: answer = fact (5)

Activation Stack for Factorial

main program: Finished: answer = 120

Exponentiation

- **base**^{exponent}
 - e.g. **5³**
 - Could be written as a function
 - **Power(base, exp)**

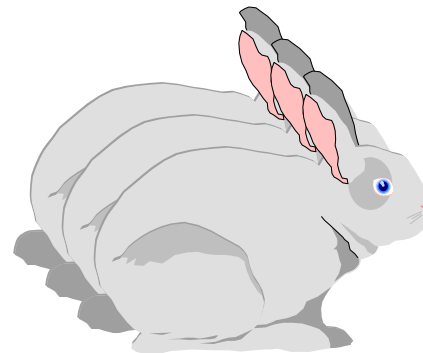
Can we write it recursively?

- $b^e = b * b^{(e-1)}$
- What's the limiting case?
- When $e = 0$ we have b^0 which always equals?
 - 1

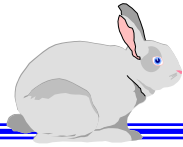
Another Recursive Function

```
function ans = power(base, exp) {  
    if( exp == 0 )  
        ans = 1.0;  
    else  
        ans = base * power( base, exp - 1 );  
    end
```

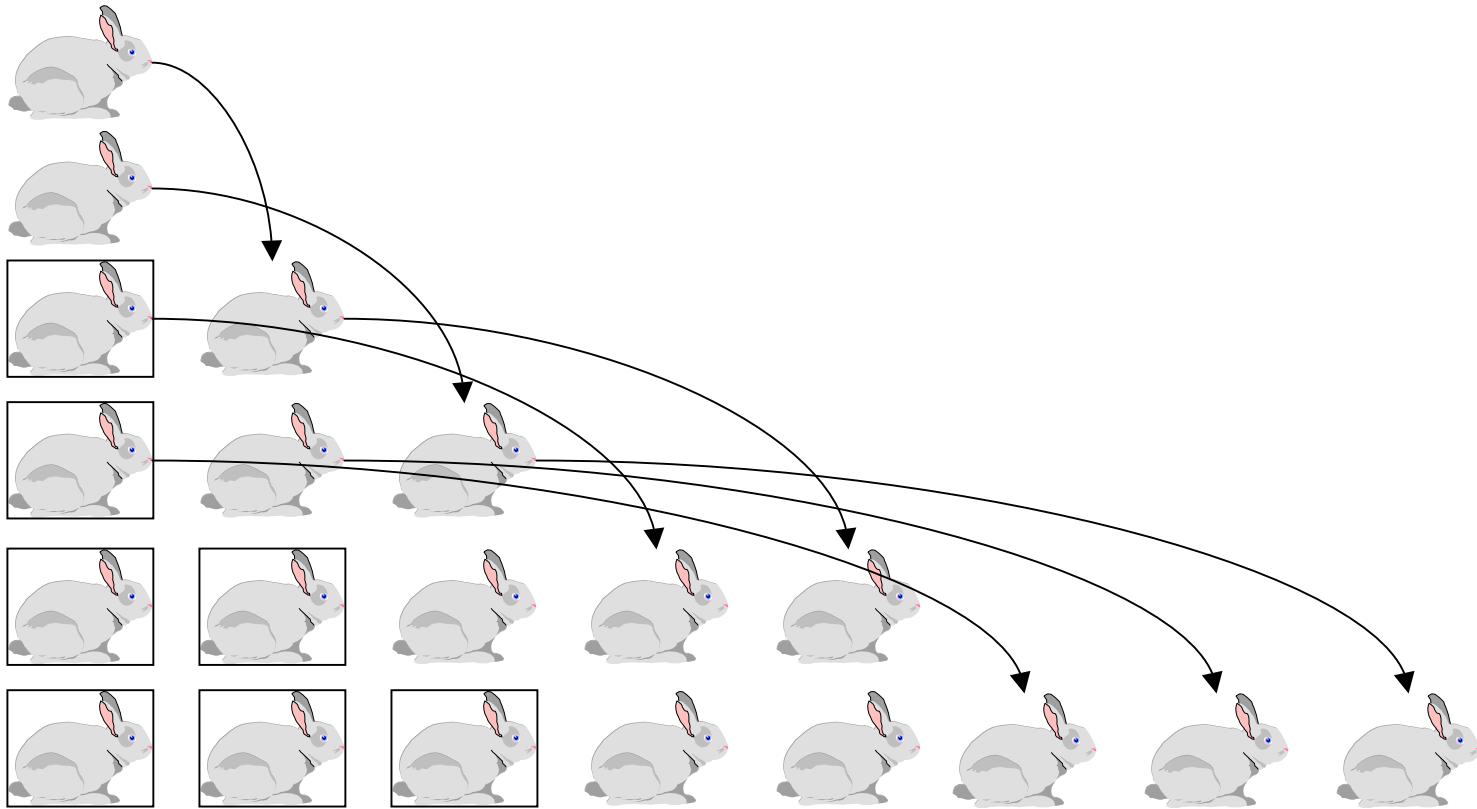
Bunnies?



- **The Time: 13th Century**
- **The Place: Italy**
- **The Man: Fibonacci**
- **The Problem: We start with a pair of newborn rabbits. At the end of the 2nd month, and each month thereafter the female gives birth to a new pair of rabbits: one male and one female. The babies mature at the same rate as the parents and begin to produce offspring on the same schedule. So how many rabbits do we have at the end of one year?**



= 1 pair bunnies (m/f)



A More Complex Recursive Function

- Fibonacci Number Sequence
- if $n = 1$, then $\text{Fib}(n) = 1$
- if $n = 2$, then $\text{Fib}(n) = 1$
- if $n > 2$, then $\text{Fib}(n) = \text{Fib}(n-2) + \text{Fib}(n-1)$
- Numbers in the series:
 - 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Fibonacci Sequence Function

```
function ans = Fib(n)
    if (n == 1) || (n == 2)
        ans = 1;
    else
        ans = Fib( n - 1 ) + Fib( n - 2 );
    end
```

Tracing with Multiple Recursive Calls

Main Algorithm: **answer = Fib(5)**

Tracing with Multiple Recursive Calls

Fib(5):	Fib returns Fib(3) + Fib(4)
----------------	---

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(3):	Fib returns Fib(1) + Fib(2)
----------------	---

Fib(5):	Fib returns Fib(3) + Fib(4)
----------------	---

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(1):	Fib returns 1
----------------	----------------------

Fib(3):	Fib returns Fib(1) + Fib(2)
----------------	---

Fib(5):	Fib returns Fib(3) + Fib(4)
----------------	---

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(3):	Fib returns 1 + Fib(2)
----------------	-------------------------------

Fib(5):	Fib returns Fib(3) + Fib(4)
----------------	------------------------------------

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(2):	Fib returns 1
----------------	----------------------

Fib(3):	Fib returns 1 + Fib(2)
----------------	-------------------------------

Fib(5):	Fib returns Fib(3) + Fib(4)
----------------	------------------------------------

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(3):	Fib returns 1 + 1
----------------	--------------------------

Fib(5):	Fib returns Fib(3) + Fib(4)
----------------	---

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(5):	Fib returns 2 + Fib(4)
----------------	-------------------------------

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(4):	Fib returns Fib(2) + Fib(3)
----------------	---

Fib(5):	Fib returns 2 + Fib(4)
----------------	--------------------------------------

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(2):	Fib returns 1
----------------	----------------------

Fib(4):	Fib returns Fib(2) + Fib(3)
----------------	---

Fib(5):	Fib returns 2 + Fib(4)
----------------	--------------------------------------

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(4):	Fib returns 1 + Fib(3)
----------------	-------------------------------

Fib(5):	Fib returns 2 + Fib(4)
----------------	-------------------------------

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(3):	Fib returns Fib(1) + Fib(2)
----------------	------------------------------------

Fib(4):	Fib returns 1 + Fib(3)
----------------	-------------------------------

Fib(5):	Fib returns 2 + Fib(4)
----------------	-------------------------------

Main Algorithm:	answer = Fib(5)
------------------------	-----------------

Tracing with Multiple Recursive Calls

Fib(1):	Fib returns 1
----------------	----------------------

Fib(3):	Fib returns Fib(1) + Fib(2)
----------------	---

Fib(4):	Fib returns 1 + Fib(3)
----------------	--------------------------------------

Fib(5):	Fib returns 2 + Fib(4)
----------------	--------------------------------------

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(3):	Fib returns 1 + Fib(2)
----------------	-------------------------------

Fib(4):	Fib returns 1 + Fib(3)
----------------	-------------------------------

Fib(5):	Fib returns 2 + Fib(4)
----------------	-------------------------------

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(2):	Fib returns 1
----------------	----------------------

Fib(3):	Fib returns 1 + Fib(2)
----------------	-------------------------------

Fib(4):	Fib returns 1 + Fib(3)
----------------	-------------------------------

Fib(5):	Fib returns 2 + Fib(4)
----------------	-------------------------------

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(3):	Fib returns 1 + 1
----------------	--------------------------

Fib(4):	Fib returns 1 + Fib(3)
----------------	--------------------------------------

Fib(5):	Fib returns 2 + Fib(4)
----------------	--------------------------------------

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(4):	Fib returns 1 + 2
----------------	--------------------------

Fib(5):	Fib returns 2 + Fib(4)
----------------	-------------------------------

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Fib(5):	Fib returns 2 + 3
----------------	--------------------------

Main Algorithm:	answer = Fib(5)
------------------------	------------------------

Tracing with Multiple Recursive Calls

Main Algorithm: answer = 5

Questions?

