

Computer Programming 143 – Lecture 12

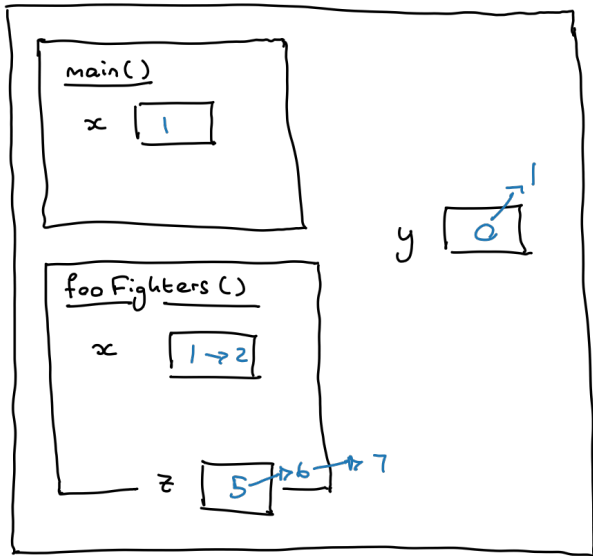
Functions IV

Faculty of Engineering



Variable scope

- Local variables :
 - Live inside functions
 - Used up to now
- Global variables :
 - Live everywhere
 - Use sparingly!
- Static variables :
 - Live inside functions, but across function calls
 - The weird one



5.14 Recursion I

Function calls so far

- Functions call one another in a disciplined and hierarchical manner
- Functions only call other functions – not themselves

Definition of recursion

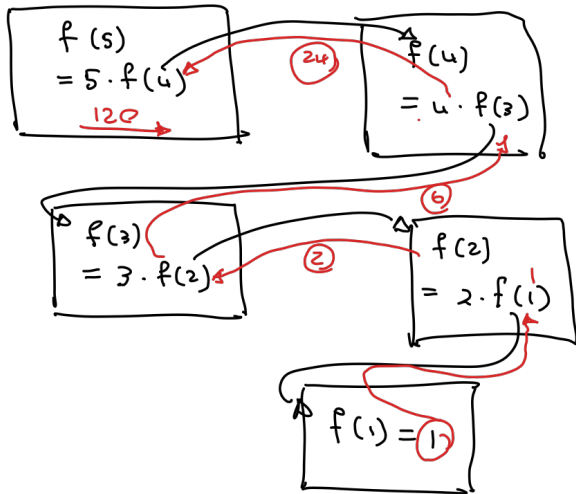
A **recursive function** is a function that calls itself, either directly or indirectly through another function

Recursion example in mathematics

What is $f(5)$?

$$f(n) = \begin{cases} 1 & \text{if } n = 1 \\ n \cdot f(n-1) & \text{if } n > 1 \end{cases}$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$



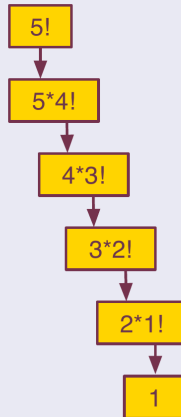
5.14 Recursion IV

Factorial 5!

$$\begin{aligned}5! &= 5 \times 4 \times 3 \times 2 \times 1 \\&= 5 \times (4 \times 3 \times 2 \times 1) \\&= 5 \times (4!)\end{aligned}$$

Similarly,

$$\begin{aligned}4! &= 4 \times (3!) \\3! &= 3 \times (2!) \\2! &= 2 \times (1!) \\1! &= 1\end{aligned}$$

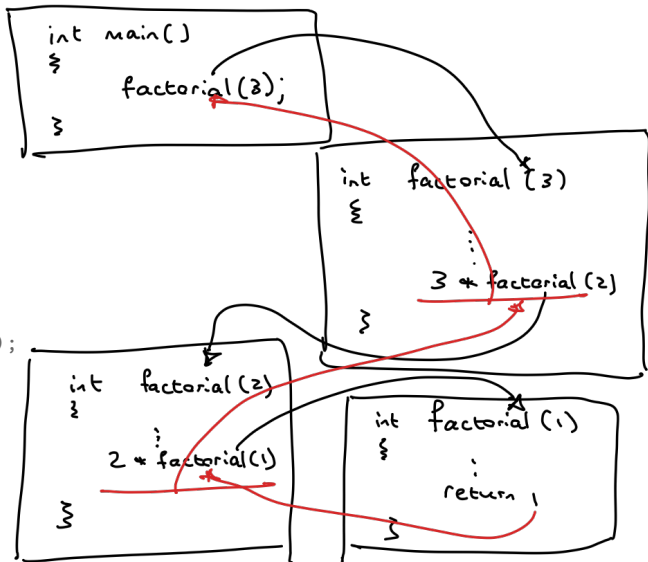


Factorial code

```
int factorial(int n)
{
    int result;

    if (n == 1)
        result = 1;
    else
        result = n*factorial(n - 1);

    return result;
}
```



5.15 Example: Fibonacci Series

Fibonacci series

The Fibonacci series, 0, 1, 1, 2, 3, 5, 8, 13, 21, ..., begins with 0 and 1 and has the property that each subsequent Fibonacci number is the sum of the previous two Fibonacci numbers. Any Fibonacci number can therefore be

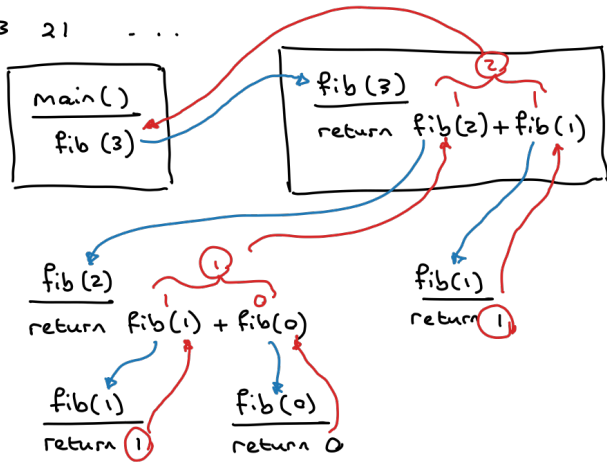
calculated as

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

Fibonacci

n	0	1	2	3	4	5	6	7	8	
fibonacci	0	1	1	2	3	5	8	13	21	...

```
def fib(n):  
    if n == 0 or n == 1:  
        return n  
    elif n > 1:  
        return fib(n-1) + fib(n-2)
```

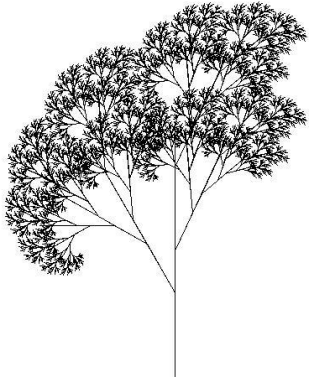


5.16 Recursion vs. Iteration I

Recursion vs. Iteration

- All problems that can be solved with recursion can also be solved with iteration
- Recursion is more processor and memory intensive
- Sometimes the recursion solutions are more elegant

Recursion in practice



The Consequences of Recursion

Question:

But what's so great about that?

Answer, Part 1:

Once recursion gets started, *it never has to stop*.

- ▶ Once the rules allow a sentence inside a sentence...
(Dave thinks that **he is cool**)
- ▶ ...then they also allow a sentence in a sentence in a sentence...
(Mary said that **Dave thinks that he is cool**)
- ▶ ...and a sentence in a sentence in a sentence in a sentence...
(John wonders if **Mary said that Dave thinks that he is cool**)
- ▶ ...and so on, and so on....

Syntax:
Recursion,
Conjunction, and
Constituency

Supplementary
Readings

Recursion

Conjunction

Constituency Tests

Auxiliary Verbs

Homework

- 1 Study Sections 5.14-5.16 in Deitel & Deitel
- 2 Do Self Review Exercises 5.1(k)-(q) in Deitel & Deitel
- 3 Do Exercises 5.34, 5.36 in Deitel & Deitel