Recursion

Peter Tiño
P.Tino@cs.bham.ac.uk, www.cs.bham.ac.uk/~pxt

School of Computer Science
University of Birmingham, UK
Recursive functions

Recursive function can call itself as a subroutine.

Example:
- check whether in a given binary tree (number labels in nodes) there is a node with label in an interval [MIN,MAX]
- need to search the tree

- we know what to do in a leaf node: just check if the label is [MIN,MAX]

- what about the internal nodes?

If in an internal node:
- check the label
- if left descendent is not NIL - continue search there
- if right descendent is not NIL - continue search there
Searching binary trees

**Information in each node:**
- numerical label
- reference to the right descendent
- reference to the left descendent

Let’s outline a function **SearchTree**
- it takes an input parameter - reference to the current node

```
SearchTree(refCurrentNode)
```

- check if refCurrentNode.label is in [MIN,MAX]

- **if** refCurrentNode.refLeftDescendent \(\neq\) NIL  
  **then** SearchTree(refCurrentNode.label.refLeftDescendent)

- **if** refCurrentNode.refRightDescendent \(\neq\) NIL  
  **then** SearchTree(refCurrentNode.refRightDescendent)

Recursion – p.3/7
How to **initialize the search** process?

SearchTree(refRoot)

How do we know the search will **finish**?

In the **leaf nodes** we just check the label, no further recursive calls

Draw a binary tree with numerical labels in the nodes and try to imitate the working of SearchTree!
Another simple example with natural recursive structure is \textbf{factorial}

\[ n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n - 1) \cdot n \]
\[ = (n - 1)! \cdot n \]

By definition: \(0! = 1\).

\[ 4! = 4 \cdot 3! \]
\[ = 4 \cdot 3 \cdot 2! \]
\[ = 4 \cdot 3 \cdot 2 \cdot 1! \]
\[ = 4 \cdot 3 \cdot 2 \cdot 1 \cdot 0! \]
factorial(n)

**input parameter:** n

**base case:**
If n == 0, then fact = 1

**now the trick:**
Else fact = n * factorial(n-1)

return fact

- It is important to make sure the recursion "bottoms up" and terminates (Base Case)!

Emulate evaluations and function calls initiated by factorial(4).
**Fibonacci numbers** is a sequence defined by:

**Initialization:** \( x_0 = 0, x_1 = 1 \)

**for all** \( n > 1 \): \( x_n = x_{n-1} + x_{n-2} \)

What is the base case?
How would you write the main recursion?

\[
\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)
\]

Think carefully about potential problems/drawbacks of this approach...