# Tutorium Funktionale Programmierung 2019

## Part 2 -Functional Decomposition, Parsing and Divide-and-Conquer

**VO - *Einführung***

Benedikt Dornauer, 14.10.2019

| precedence | operators | associativity |
|---|---|---|
| 9 | !!, . | left(!!), right(.) |
| 8 | ^, ^^, ** | right |
| 7 | *, /, `div` | left |
| 6 | +, - | left |
| 5 | :, ++ | right |
| 4 | ==, /=, <, <=, >, >= | none |
| 3 | && | right |
| 2 | \|\| | right |
| 1 | >>, >>= | left |
| 0 | $, $!, `seq` | right |

## Precedence

**Example:** 4 * 4 + 4

What is calculated first
4 * 4 or 4 + 4 ?

→ (4 * 4) + 4

## Associativity

**Example :** True && True && True

Which order should I take:
(True && True) && True
or True && (True && True) ?

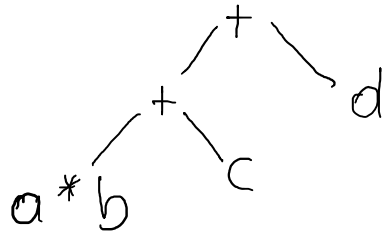
→ **True && (True && True)**

# Abstract Syntax Tree (AST) for *a * b + c + d*

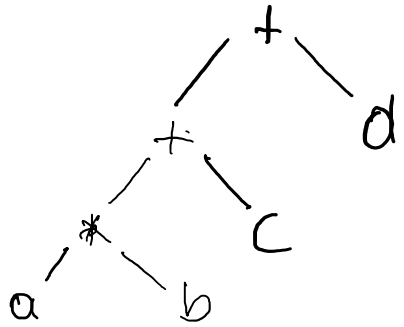
- a * b + c + d
- a * b + c + d
- a * b + c + d

  (see table ~> left)

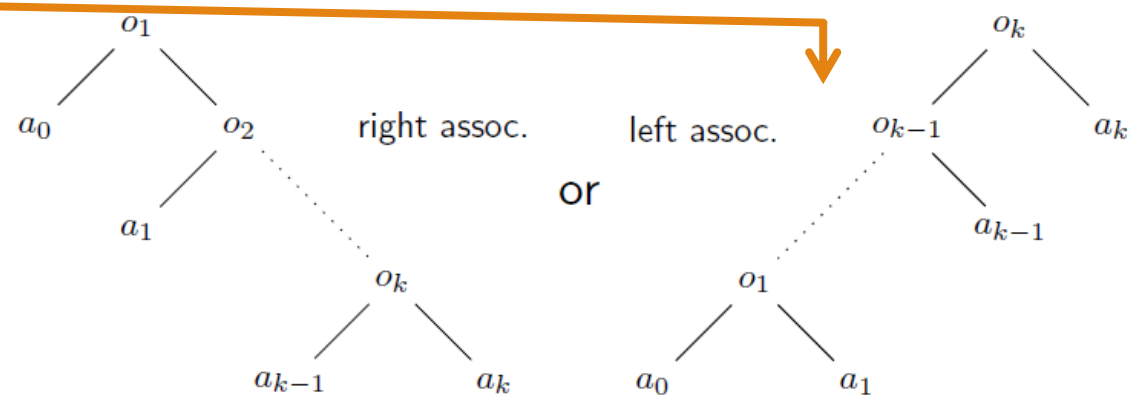
- (a * b + c) + d



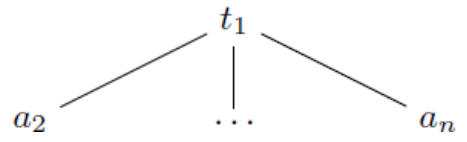

- (a * b + c) + d




- have: $t = t_1 \dots t_n$ where $t$ is non-atomic
- if some $t_i$ is an operator
  - split $t$ at those operators of $\{t_1, \dots, t_n\}$ with least precedence
    - obtain $t = s_0 o_1 s_1 o_2 \dots o_k s_k$ where the $o_i$ are the oper. with least prec.
    - throw error if there is no strict alternation of oper. with non-operators
  - if $k > 1$ check whether all $o_i$ have same associativity ($\neq$ none)
  - return




    where $a_0, \dots, a_k$ are the ASTs of $s_0, \dots, s_k$
- otherwise (function application)
  - check that $t_1$ is a name
  - compute ASTs $a_2, \dots, a_n$ for $t_2, \dots, t_n$
  - return



# Exercise 2.1: Parsing and AST (Abstract syntax tree)

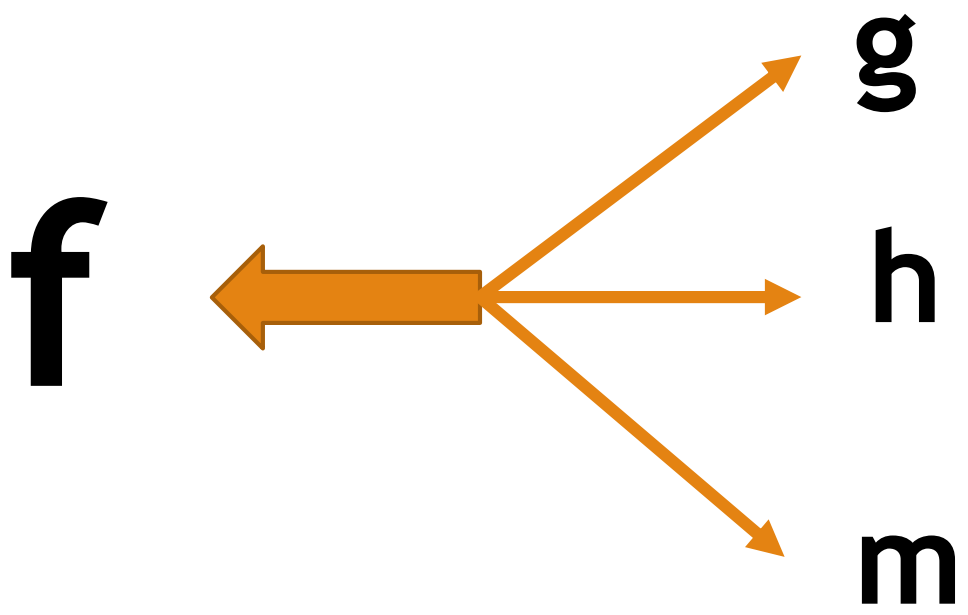
Draw the AST (**abstract syntax tree**) of the following expression

***funcTest True True || (False && False) || False***

Note: True

# Functional Decomposition

"Large or complex functionalities are more easily understood when **broken down into pieces** using functional decomposition."




**Hint**: write separate functions for each task

Citation: URL: https://www.tutorialspoint.com/software_testing_dictionary/functional_decomposition.htm (09.10.2019)

# Exercise 2.2: Functional Decomposition

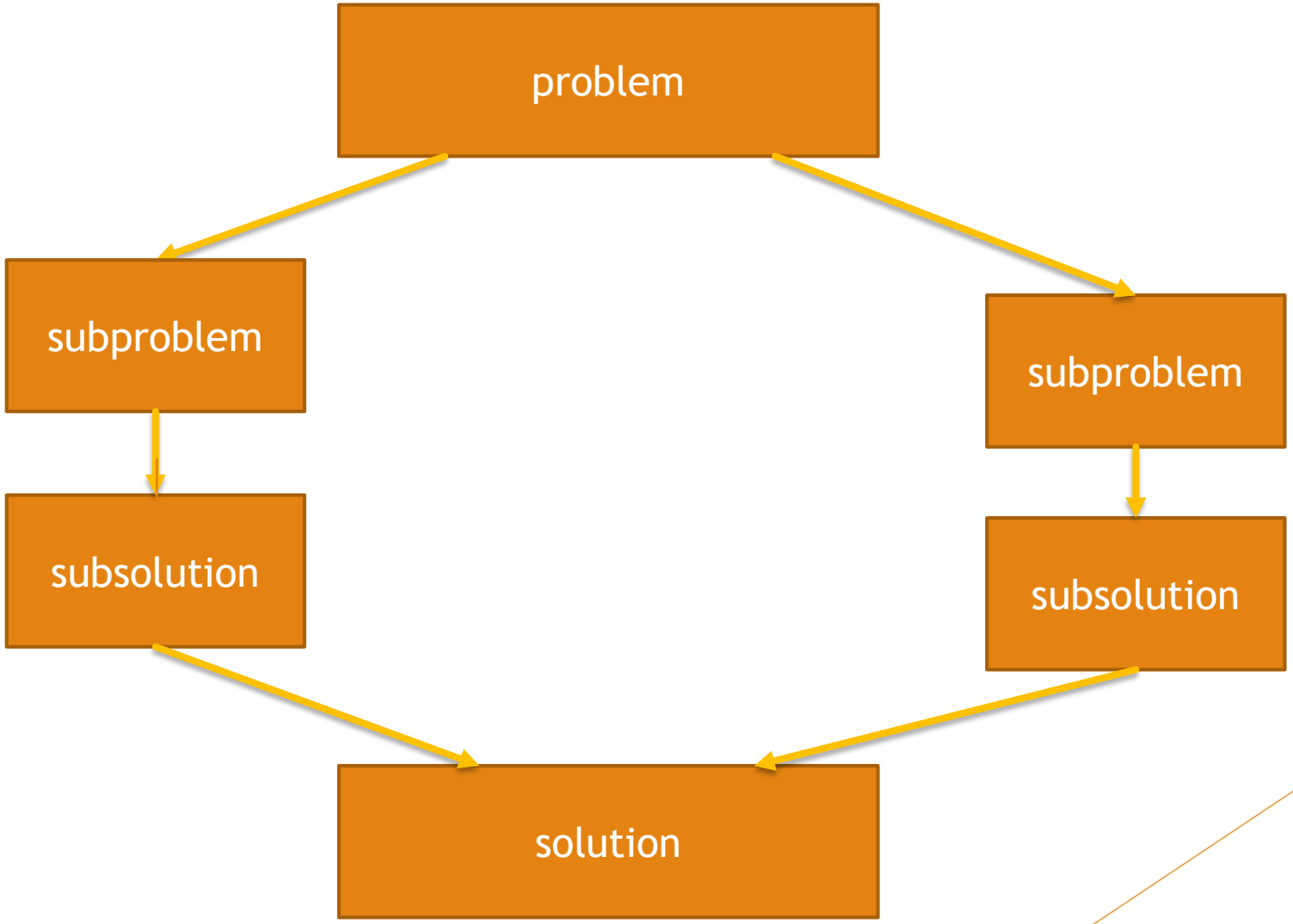
In beach volleyball two teams A and B play again each other. Team A has *a* points and team B has *b* points. A set point can be achieved by one team if it has more than 20 points and the point-difference between a and b is greater or equal 2.

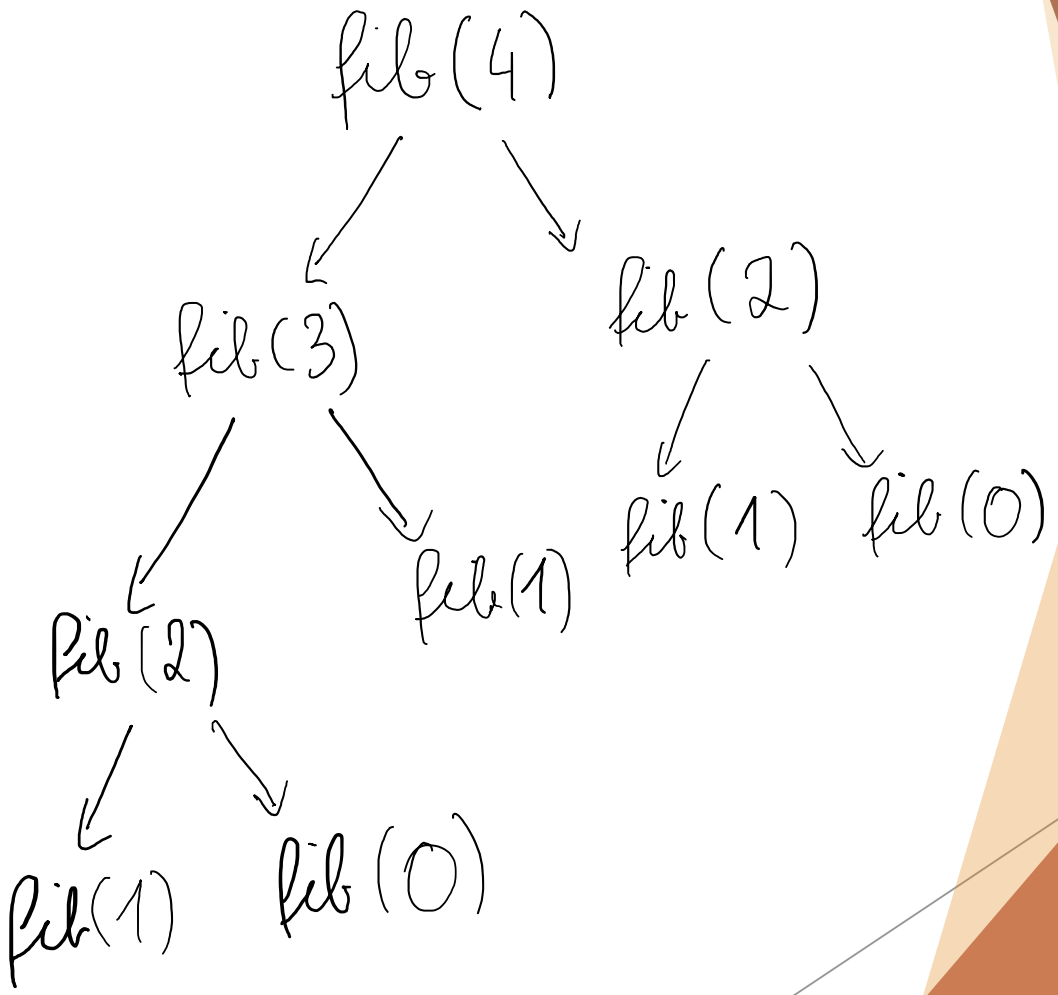
- Express a function which returns *True* if the game is over, otherwise *False*.
- Use functional decomposition!

**Example:**

```
*Main> checkerSetPoint 21 15
True
*Main> checkerSetPoint 21 20
False
```

# Divide-and-Conquer ("Teile-und-herrsche")






# Exercise 2.3: Divide-and-Conquer

Implement a function which returns the $n^{th}$ number of the Fibonacci-Sequence e.g. *fib 6 = 8*.

**Hint:**

Fibonacci Sequence [0,1,1,2,3,5,8 13, …]: The next number is found by adding up the two numbers before it.

# Alternative Implementations

```
--using only if then else
fib2 :: Integer -> Integer
fib2 x =
 if x == 0
 then 0
 else (if x == 1 then 1 else fib2 (x-1) + fib2 (x-2))
```

# Questions? Need help? Feedback? etc.

- benedikt.dornauer@student.uibk.ac.at