# Programming Paradigms
# Written Exam (6 CPs)

31.01.2018

| First name | | Last name | |
|---|---|---|---|
| Student number | | Signature | |

## Instructions for Students

- Write your name and student number on the exam sheet and on every solution sheet you hand in and also sign them.
- This is a closed book exam: the only resources allowed are blank paper and pens (do not use pencils).
- Write neatly and clearly. The clarity of your explanations will affect your grade.
- The duration of the exam is 2 hours.

Good luck!

---

## Do not write in this space

| Exercise | Marks | Achieved |
|---|---|---|
| 1 | 20 | |
| 2 | 12 | |
| 3 | 8 | |
| 4 | 10 | |
| 5 | 16 | |
| 6 | 14 | |
| 7 | 10 | |
| 8 | 10 | |
| Total | 100 | |

**Exercise 1** (20 marks)

a. (4 marks) Briefly describe the main differences, advantages and disadvantages of static typing and dynamic typing.

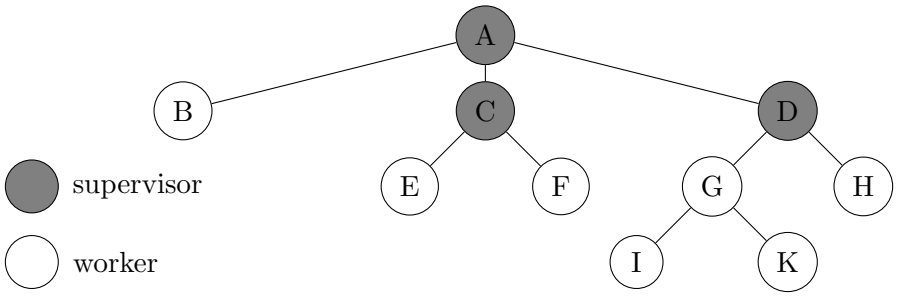
b. (4 marks) Briefly describe the concept of mixins in Ruby.

c. (4 marks) What is the following Prolog program doing?

```
foo :-
    repeat,
    read(X),
    write(X),
    nl,
    X = 'quit',
    !.
```

d. (4 marks) What is the following Haskell list comprehension producing?
`[(4-x,y) | (x,y) <- [(1,2), (2,3), (3,1)]]`

e. (4 marks) The following figure shows a hierarchy of linked Erlang processes, also called supervision tree. What happens if process $I$ crashes?



**Exercise 2** (12 marks) Write a Ruby function that implements `counting sort` for sorting an array of numbers, which works as follows: Given an array $A$ of positive integers $n_1, ..., n_k \in [0, k]$, create an array $A'$ of size $k + 1$. Each element of $A$ is associated with an index in $A'$. The algorithm counts then the number of occurrences of each element in $A$ and stores it in the corresponding cell in $A'$. Finally, with a scan of $A'$ the elements can be retrieved in ascending (or descending) order.

For example, for the input array `A = [6,5,1,7,8,1,2]` we obtain `A' = [0,2,1,0,0,1,1,1,1]`. The element $A'[0]$ represents that the number 0 occurs zero times, $A'[1]$ represents that the number 1 occurs two times, etc. Scanning $A'$ allows to retrieve the sorted array `[1,1,2,5,6,7,8]`.

**Exercise 3** (8 marks) The following Ruby function computes $\lceil \log_2 n \rceil$ recursively:

```
def log( n )
  return 0 if n == 1
  if n.even?
    n = n / 2
    return log( n ) + 1
  else
    n += 1
    return log( n )
  end
end
```

Rewrite this function into a tail-recursive function.

**Exercise 4** (10 marks) Write a Prolog program `pythagoras(A,B,C)` using the "generate and test" pattern, which for a given value $C$ computes all possible integer values of $A$ and $B$ for which the theorem of Pythagoras holds, i.e., $A^2 + B^2 = C^2$. For instance, `pythagoras(A,B,5)` returns `A=3, B=4` and `A=4, B=3`, whereas `pythagoras(A,B,6)` fails.

Hint: You can use the predicate `between(L,U,X)`, which generates all integers between `L` and `U`, e.g., `between(0,2,X)` generates

```
X = 0 ;
X = 1 ;
X = 2.
```

**Exercise 5** (16 marks) Write a program in Prolog that goes through a list of numbers and selects numbers (starting from the beginning of the list) whose sum is smaller than a given capacity. So as long as there is still enough capacity left, the program keeps selecting numbers (skipping numbers that are too large). The program should return the result in a list. For example, given the list `[2,5,3,8,1,12]` and the capacity 14, the program should return the list `[2,5,3,1]` as this sums up to 11, which is less than 14 (when 8 and 12 are reached, they are too large to be included). The order of the items in the result does not matter.

**Exercise 6** (14 marks) Write a function `isbalanced` in Haskell that checks whether a string containing open and closed parentheses is balanced. A string of parentheses is balanced when every open one has a corresponding closed one and at any point there are not more closed ones than open ones. For example, the strings `""`, `"(())"` and `"(())()"` are balanced, whereas the strings `"()(()"` and `"())"` are not balanced.

**Exercise 7** (10 marks) Look at the following recursive Haskell program:

```
mystery :: [a] -> Integer
mystery [] = 1
mystery (h:t) = 2 * (mystery t)
```

a. (4 marks) Briefly describe what the program does.

b. (6 marks) Transform the program into a tail-recursive one.

**Exercise 8** (10 marks) Write a function `loop` for an Erlang process that receives messages consisting of a single parameter and does the following:

- if the parameter is a number, it outputs to the console whether the number is positive, negative, or zero;
- if the parameter is "bye", the process terminates;
- otherwise, an error message is printed, e.g., "Unexpected message".

Show also how to start the process. (Hint: you can use a function `is_number(N)`, which is true if `N` is a number, and false otherwise)

**Solution 1**

a. Static typing:

- types and their constraints are checked before executing the program
- pro: less error-prone
- con: sometimes too restrictive

Dynamic typing:

- type checking is done during program execution
- pro: more flexible
- con: harder to debug

b. A mixin in Ruby is a combination of modules and classes. More specifically, a module can be included in a class definition. By doing so, all methods of the module are added and available to the class. Mixins have some similarity to the concept of multiple inheritance.

c. This is a simple echo program. It reads from the standard input and shows the input on the standard output. When the user input is "quit", the program terminates.

d. `[(3,2), (2,3), (1,1)]`

e. This causes process $G$ to terminate, which in turn terminates process $K$. Process $D$ traps the exit signal and restarts processes $G$, $I$, and $K$.

**Solution 2**

```
def counting_sort(myarray)
  newarray = Array.new(myarray.max+1,0)
  finalarray = Array.new

  myarray.each do |x|
    newarray[x] = newarray[x] + 1
  end

  for i in 0...(newarray.length)
    newarray[i].times do
      finalarray.push(i)
    end
  end

  return finalarray
end
```

**Solution 3**

```
def log_tail_recursive( n )
  return log_tr( n, 0 )
end

def log_tr( n, r )
  return r if n == 1
  if n.even?
    n = n / 2
    return log_tr( n, r+1 )
  else
    n += 1
    return log_tr( n, r )
  end
end
```

**Solution 4**

```
pythagoras( A, B, C ) :-
    between( 1, C, A ),
    between( 1, C, B ),
    X is C * C,
    Y is A * A + B * B,
    X = Y.
```

**Solution 5**

- Solution with accumulator

```
fit( L, C, R ) :- fit_acc( L, C, [], R ).

fit_acc( [], C, L, L ).
fit_acc( [H|T], C, L, R ) :-
    H > C,
    fit( T, C, L, R ).
fit_acc( [H|T], C, L, R ) :-
    H <= C,
    N is C - H,
    fit_acc( T, N, [H|L], R ).
```

- Solution without accumulator

```
fit( [], _, [] ).
fit( [H|T], C, R ) :-
    H > C,
    fit( T, C, R ).
fit( [H|T], C, [H|R] ) :-
    H =< C,
    C1 is C - H,
    fit( T, C1, R ).
```

**Solution 6**

```
module IsBalanced (isbalanced) where

isbalanced :: [Char] -> Bool
isbalanced s = isbalan s 0

isbalan :: [Char] -> Int -> Bool
isbalan [] 0 = True
isbalan [] nonzero = False
isbalan (h:t) x =
   if x < 0 then
      False
   else
      if h == '(' then
         isbalan t (x+1)
      else
         isbalan t (x-1)
```

**Solution 7**

a. The program computes the exponential function $2^l$, where $l$ is the length of the array that is passed as parameter.

b. Tail-recursive version: pass the length of the array seen so far as a second parameter; when the list is empty, the second parameter contains the length of the list. Call the function as `expfun [...]  1`

```
expfun :: [a] -> Integer -> Integer
expfun [] x = x
expfun (h:t) x = expfun t (x * 2)
```

**Solution 8**

```
-module(sign).
-export([loop/0]).

loop() ->
  receive
    N when is_number(N) ->
      if
        N < 0 -> io:format("Number is negative~n");
        N > 0 -> io:format("Number is positive~n");
        N == 0 -> io:format("Number is zero~n")
       end,
      loop();
    "bye" ->
      io:format("Bye~n");
    _ ->
      io:format("Unexpected message~n"),
      loop()
  end.

P = spawn( fun sign:loop/0 ).
```