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.
- Use a pen and not a pencil.
- Write neatly and clearly. The clarity of your explanations will affect your grade.
- At least 51 points are required to pass the exam.
- The duration of the exam is 2 hours.

Good luck!

Do not write in this space

<table>
<thead>
<tr>
<th>Exercise</th>
<th>Marks</th>
<th>Achieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Exercise 1 (20 marks)

a. (4 marks) Briefly describe the concept of tail recursion, and why it is desirable to write tail recursive functions.

b. (3 marks) Is the following Ruby expression syntactically correct?

```ruby
['2', 'plus', '3', 'is', '#{2+3}'].each { |x| puts x }
```

If no, explain what is wrong. If yes, what does the expression do?

c. (4 marks) Briefly describe the generate and test pattern in Prolog.

d. (4 marks) Consider the Haskell function `prod x y = x * y`. How is the function call `prod 2 4` evaluated? What is the name of this evaluation concept?

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

![Erlang Supervision Tree Diagram]

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 $A = [6, 5, 1, 7, 1, 1, 2]$ we obtain $A' = [0, 3, 1, 0, 0, 1, 1, 1]$. The element $A'[0]$ represents that the number 0 occurs zero times, $A'[1]$ represents that the number 1 occurs three times, etc. Scanning $A'$ allows to retrieve the sorted array $[1, 1, 1, 2, 5, 6, 7]$. 
Exercise 3 (8 marks) Write a Ruby class Animal with a property

- kind: a string that holds the type of the animal

and the following instance methods:

- eat: takes a parameter food and prints a message that the animal is eating food
- sleep and wake: these two methods do not have any arguments; instead, they will set an instance variable @state to the string "asleep" and "awake", respectively

Write a second Ruby class Person with the following characteristics:

- Inherits from Animal
- Automatically sets @kind to "person"
- Adds 3 new instance variables: age, gender, name
- Overrides the eat method so that a person cannot eat a "person"

Exercise 4 (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 5 (10 marks) The following Prolog knowledge base describes a small social network using the friend relation that represents a direct friendship between two persons.

friend(tom,tim).
friend(tom,alf).
friend(alf,ann).
friend(alf,joe).
friend(joe,sue).
friend(joe,tim).
friend(sue,ann).

Write a predicate friends_dist(X,Y,D) which tells whether X and Y are connected by friendship relations at a distance of D. For instance, friends_dist(tom,Y,2) succeeds and instantiates Y=ann and Y=joe.
**Exercise 6** (10 marks) Write a Haskell function `palindrom` that given a string in input returns the palindrom of this string. Here are a few examples:
```
palindrom "abc" returns "abccba"
```

**Exercise 7** (14 marks) Write a Haskell module that exports a tail-recursive function `noOfElem` that takes as input an element `x` and a list and returns the number of occurrences of `x` in the list. For instance, `NoOfElem 1 [1,2,3,1]` returns 2.

**Exercise 8** (10 marks) Write a server/receiver program in Erlang that implements a buffered announcement board. The server receives messages and stores them in a list. When `N` messages are collected, the messages are printed, the buffer is flushed, and the server continues running. Write also the command that is needed to send messages to the server (no need to implement a receiver) and the command to start the server from the command line.
Solution 1

a. A function is tail-recursive if there is no operation after the recursive call, that is, no operations are executed after the recursive call terminates. As a consequence, no data need to be stored on the stack that is needed when the recursive call terminates. Hence, different from non-tail-recursive function, for tail-recursive functions the stack does not grow with each recursive call.

b. The expression is syntactically correct. The result is: 2
   plus
   3
   is
   5

c. The “generate and test” pattern has the following form:
   foo :- g1, g2, ..., gn, t1, t2, ..., tm.

   • The predicates g1, g2, ..., gn generate lots of different potential solutions.
   • The predicates t1, t2, ..., tm test, whether something generated by g1, g2, ..., gn is a solution; if this is not the case, backtracking starts and g1, g2, ..., gn generate the next candidate.

d. The function is evaluated in two steps:

   • The first input parameter is applied, i.e., prod 2, yielding a partially evaluated function (\ y \rightarrow 2 * y)

   • The partially evaluated function is applied to the second argument, i.e., (\ y \rightarrow 2 * y) 4, yielding 8

   This is called curried functions.

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

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

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

    newarray
```

5
for i in 0...(newarray.length)
    newarray[i].times do
        finalarray.push(i)
    end
end

return finalarray
end

Solution 3

class Animal
    @kind
    @state

    def initialize(kind)
        @kind = kind
    end

    def eat(food)
        print "Animal eats: #{food}\n"
    end

    def sleep
        @state = "asleep"
    end

    def wake
        @state = "awake"
    end
end

class Person < Animal
    @age
    @gender
    @name

    def initialize(age,gender,name)
        super("person")
        @age = age
        @gender = gender
        @name = name
    end
def eat(food)
    print "Animal eats: #{food}\n" if food != "person"
end
end

Solution 4

- 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 5

friends_dist( X, Y, 1 ) :- friend( X, Y ).
friends_dist( X, Y, D ) :-
    friend( X, Z ),
    D1 is D - 1,
    friends_dist( Z, Y, D1 ).

7
Solution 6

module Palindrom (palindrom) where

palindrom :: [Char] -> [Char]
palindrom [] = []
palindrom (h:t) = [h] ++ palindrom t ++ [h]

Solution 7

module NoOfElem (noOfElem) where

noOfEl :: Eq a => a -> [a] -> Int -> Int
noOfEl x [] cnt = cnt
noOfEl x (h:t) cnt | x == h = noOfEl x t (cnt+1)
noOfEl x (h:t) cnt = noOfEl x t cnt

noOfElem :: Eq a => a -> [a] -> Int
noOfElem x lst = noOfElem x lst 0

Non tail-recursive version:

module NoOfElem (noOfElem) where

noOfElem :: Eq a => a -> [a] -> Int
noOfElem x [] = 0
noOfElem x (h:t) = (if h == x then 1 else 0) + noOfElem x t

Solution 8

-module(bufab).
-export([loop/3]).

loop(N,L,C) ->
    receive
        X ->
        if
            C+1 == N ->
                io:format("Board: ~p~n", [[X|L]]),
                loop(N,[],0);
            true ->
                loop(N,[X|L],C+1)
end
end.

% Starting server from command line
Server = spawn(bufab,loop,[N,[],0]).
Server ! message1.