Programming Paradigms
Written Exam (6 CPs)

22.06.2017

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	15	
5	10	
6	15	
7	10	
8	10	
Total	100	

Exercise 1 (20 marks)

a. (4 marks) Briefly describe the main difference between a compiled language and an interpreted language.

b. (4 marks) What does the following Ruby-code print?

```
def like_map(array)
  result = []
  array.each do |element|
    result << (yield element)
  end
  result
end

x = like_map([1, 2, 3]) do |number|
  number * 2
end

print x
```

c. (4 marks) Consider the following Prolog program:

```
a(1).
a(z).
b(3).
b(z).
c(A,B) :- b(B), !, a(A).
d(A,B) :- b(B), a(A).
```

How many solutions are produced by each of the two queries `c(A,B)` and `d(A,B)`? Briefly explain your answer.

d. (4 marks) What does the following list comprehension written in Haskell produce?

```
[(x,y) | x <- [1..4], y <- (filter odd [1..4])]
```

e. (4 marks) What is wrong with the following case statement in Erlang? How could you fix the code?

```
case X of
   {_,_} -> doA;
   {_,3} -> doB;
   {2,_} -> doC;
   {2,3} -> doD
end.
```

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) Write a class `Vehicle` with a field `year` and a method `printStats` that prints the value of `year`. The value of `year` is given as an argument to the constructor of `Vehicle`. Next, write a class `Car` that extends `Vehicle` and has two fields `model` and `brand`. The constructor of car has three arguments, and it assigns values to `model` and `brand` and calls the constructor of the superclass `Vehicle` to initialize `year`. Furthermore, create a method `printStats` which prints `model` and `brand`, and calls the parent method in `Vehicle` to print `year`. Finally, assume a module `FourWheeled` and use it to create a mixin with the class `Car`.

Exercise 4 (15 marks) Write a Prolog program `duplicates(L,D)`, which takes a list `L` of integer values and returns in `D` a list of all elements in `L` that occur at least twice; `D` should contain each duplicate value only once. For example, `duplicates([1, 2, 5, 5, 2, 4, 2])` should return `D = [2, 5]` (or `D = [5, 2]`, the order does not matter) and not `D = [2, 5, 5, 2, 2]`). You should use an accumulator to collect the elements of the result list. (Hint: if you have difficulties with the accumulator, try to write a program without an accumulator, for which you can get up to 10 marks)

Exercise 5 (10 marks) Write a Prolog program `count(X,L,N)` that counts the number `N` of occurrences of element `X` in the list `L`.

Exercise 6 (15 marks) Write a Haskell module that exports a function `subseq` that takes as input a list `x` and a list `y` and returns `true` if `x` is a subsequence of `y` and false otherwise. For instance, `subseq [1,2,3] [3,4,1,2,3,5]` returns `true`, whereas `subseq [1,2,3] [1,2]` returns `false`.

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

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

```
unknown :: [a] -> Integer
unknown [] = 0
unknown (h:t) = 1 + unknown t
```

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

Exercise 8 (10 marks) Write a server/receiver program in Erlang which 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. The server is initialized with the parameter `N`. 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. Compiled languages are translated into a from that can be run directly on a computer's processor. Usually the whole program is translated before it is run.

   Interpreted languages are processed by a higher-level virtual machine. Usually a program is translated on the fly, i.e., a statement is translated and then immediately executed.

b. The result is `[2, 4, 6]`. The `like_map()` method takes and array and a code block as arguments. `like_map()` iterates over each element of the array, yields the code block, and appends the result to the result array. `like_map()` behaves like the `Array#map` method.

c. (a) `c(A,B)` has 2 solutions: `A = 1, B = 3; A = z, B = 3.`
   The `cut` operator (`!`) stops backtracking and prevents `b(B)` from being matched with the second `b`-fact.

   (b) `d(A,B)` has 4 solutions: `A = 1, B = 3; A = z, B = 3; A = 1, B = z; A = z, B = z (or A = B, B = z).`

d. `[(1,1),(1,3),(2,1),(2,3),(3,1),(3,3),(4,1),(4,3)]`

e. The cases are in the wrong order, i.e., the most general ones come first. Consequently, the later cases will never be reached. The code can be fixed by reversing the order of the cases.

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

```
class Vehicle
  def initialize(year)
    @year = year
  end

  def printStats
    print "Year constructed: #@year\n"
  end
end

class Car < Vehicle
  include FourWheeled
  def initialize(brand,model,year)
    super(year)
    @brand = brand
    @model = model
  end

  def printStats
    print "Brand: #@brand\n"
    print "Model: #@model\n"
    super
  end
end
```

Solution 4

```
duplicates( L, D ) :-
    duplicates_acc( L, [], D ).

duplicates_acc( [], A, A ).
duplicates_acc( [H|T], A1, A ) :-
    member(H, T),
    \+( member( H, A1 ) ), !,
    duplicates_acc( T, [H|A1], A ).
duplicates_acc( [_|T], A1, A ) :-
    duplicates_acc( T, A1, A ).
```

A solution without accumulator, and the result list may contain more than one occurence of each duplicate.

```
duplicates2( [], [] ).
duplicates2( [H|T], [H|D] ) :-
    member( H, T ), !,
    duplicates2( T, D ).
duplicates2( [_|T], D ) :-
    duplicates2( T, D ).
```

Solution 5

```
count(_,[],0).
count(A,[A|L],N):-
    !,
    count(A,L,N1),
    N is N1+1.
count(A,[_|L],N):-
    count(A,L,N).
```

Solution 6

```
module Subseq (
       subseq
) where

-- Verify whether x occurs at the beginning of y
sseq :: Eq a => [a] -> [a] -> Bool
sseq x [] = False
sseq [] y = True
sseq (x:xs) (y:ys) = if x == y then sseq xs ys else False

-- Verify whether x occurs somewhere in y
subseq :: Eq a => [a] -> [a] -> Bool
subseq x [] = False
subseq x y =
   if sseq x y then
       True
   else
      subseq x (tail y)
```

Solution 7

a. The program computes the length of an array.

b. Tail-recursive version: pass the length of the list seen so far as a second parameter; when the list is empty, the second parameter contains the length of the list.

```
module Tail (
  len
) where

len :: [a] -> Integer -> Integer
len [] x = x
len (h:t) x = len t (x+1)
```

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.
```