Programming Paradigms
Written Exam

26.09.2014

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

Question	Marks	Achieved
1	25	
2	6	
3	8	
4	16	
5	10	
6	15	
7	10	
8	10	
Total	100	

Exercise 1 (25 marks)

a. (5 marks) Briefly describe the main differences, advantages and disadvantages between static typing and dynamic typing.

b. (5 marks) Duck typing in Ruby gives a lot of flexibility when accessing objects. For example, an array can be accessed as a stack (last in, first out) or as a queue (first in, first out).

   Assume that `arr = [1,2,3,4,5]` is a Ruby array object that (in addition to the standard array operation) also supports the following operations:

   - `push` appends an element to the array
   - `pop` removes the last element of the array
   - `enqueue` adds an element to the front of the array and shifts all other elements by one position
   - `dequeue` removes last element of the array

   What will array `arr` look like after the following sequence of operations?

```
arr[0] = 10
arr.pop
arr.enqueue 6
arr.push 8
arr.dequeue
```

c. (5 marks) Briefly describe the generate and test pattern in Prolog.

d. (5 marks) Briefly explain the concept of list comprehension in Haskell and give a short example.

e. (5 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 (6 marks) Write a function ticketPrice(noOfZones,age) in Ruby that, depending on the number of zones and the age of the person, computes the price of a ticket for public transport. The base price of a ticket is €2. Each additional zone after the first adds €1 to the price. Children up to (and including) the age of 12 and persons who are 60 years and older pay half price.

Exercise 3 (8 marks) Write a Ruby function to calculate the fuel consumption of a car (in liters) for road trips it has taken. The input parameters of the function are fuelConsumpt, measured in liters per 100 kilometers, and trips[], an array containing the distances of road trips (in kilometers) taken by that car. Compute the overall fuel consumption for the voyages stored in trips.

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) Write a Prolog program aless(X,Y) that compares two words X, Y and succeeds if X is alphabetically smaller than Y. For example, aless('Friday','Saturday') succeeds, whereas aless('Friday','Friday') fails. Hint: You can use the predicate name(X,LX) that translates an atom (word) X into a list of its character codes LX, e.g., name(alp,X) yields LX = [97,108,112].

Exercise 6 (15 marks) Write an Erlang process that acts as a time server, i.e., when it receives a request from another process, it sends back the current time of day. The server process receives a tuple containing the ID and the time zone of the process requesting the time. For example, a request may look like this {PID1, +3} or this {PID1, -6}.

Assume that you have a function time(), which returns the current time in UTC (Coordinated Universal Time) or GMT (Greenwich Mean Time). The result of a call of time() is a tuple containing the hours, minutes, and seconds.

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

loop () ->
```

Exercise 7 (10 marks) Write a function in Haskell that finds an element in a list. It returns `True` if the element is in the list, otherwise it returns `False`.

```
module Findelement where

findelement ::
```

Exercise 8 (10 marks) Pascal's triangle is used to compute Binomial coefficients:

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
   ...
```

You compute the number in each row by adding the two numbers above it, e.g. $6 = 3 + 3$. If there is only one number, then the missing number is considered to be 0. The first row always contains a single 1.

Write a function `pascal` in Haskell that computes the first `n` rows of the Pascal triangle. For example, calling `pascal 3` would return `[[1], [1,1], [1,2,1]]`. Assume that you have a function `nextRow`, which, given a row, computes the next row.

```
pascal :: Int -> [[Int]]
```

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. `arr = [6, 10, 2, 3, 4]`

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. List comprehension in Haskell is a compact way to specify complex and possibly infinite lists by specifying an output function, a variable, an input set and a predicate. Example: The list of even numbers between 1 and 100 can be defined as
`[x | x <- [1..100], mod x 2 == 0]`

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 ticketPrice(noOfZones,age)
   price = 2.0 + noOfZones - 1.0
   return price/2.0 if (age <= 12 || age >= 60)
   return price
end
```

Solution 3

```
def fuel(fuelConsumpt, trips=[])
  sum = 0.0
  trips.each{ |x| sum = sum + (x/100 * fuelConsumpt) }
  return sum
end
```

Solution 4

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

Solution 5

```
aless(X,Y) :-
    name(X,LX),
    name(Y,LY),
    alessx(LX,LY).

alessx([],[_,_]).
alessx([X|_], [Y|_]) :- X < Y.
alessx([H|TX],[H|TY]) :- alessx(TX,TY).
```

Solution 6

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

loop () ->
   receive
      {PID,TZ} ->
         {H,M,S} = time(),
         if
            H + TZ > 23 -> PID ! {H + TZ - 24,M,S};
            H + TZ < 0 -> PID ! {H + TZ + 24,M,S};
            true -> PID ! {H + TZ,M,S}
         end,
         loop()
   end.
```

Solution 7

```
module Findelement where

findelement ::  Eq a => a -> [a] -> Bool
findelement x [] = False
findelement x (h:t) =
   if x == h then
      True
   else
      findelement x t
```

Solution 8

```
pascal ::  Int -> [[Int]]
pascal n = loop n [1]
           where
           loop 1 xs = [xs]
           loop n xs | n > 1 = xs :  loop (n-1) (nextRow xs)
```