UNIVERSITY
OF SKÖVDE

School of Humanities and Informatics

WRITTEN EXAMINATION

Course Advanced Programming

Sub-course

Course code IT732A

Credits for written examination 5,5

Date 20191028

Examination time 3h

Examination responsible Gunnar Mathiason

Teachers concerned Elio Ventocilla

Aid at the exam/appendices

Other

Instructions

- ☐ Take a new sheet of paper for each teacher.
- ☒ Take a new sheet of paper when starting a new question.
- ☒ Write only on one side of the paper.
- ☒ Write your name and personal ID No. on all pages you hand in.
- ☒ Use page numbering.
- ☒ Don´t use a red pen.
- ☒ Mark answered questions with a cross on the cover sheet.

Grade points

Examination results should be made public within 18 working days

Good luck!

Total number of pages 3

Advanced Programming - IT732A HT19
Exam

University of Skövde

October 28, 2019

Rules

- All questions are to be answered within the context of functional programming.
- You are expected to answer in a thorough, yet concise manner. That is, elaborate on your answers without dwelling on aspects which are not strongly related to the question at hand.
- Code examples are to be written in Scala code. Small syntax mistakes will be overlooked.
- Write in an intelligible manner. If the hand writing needs to be decoded, no points will be awarded.
- **The exam is strictly individual.**
- The exam is composed of 5 questions, each with a value of 20 pts., adding to a total of 100 pts. A minimum of 50 pts. is required to pass.

Question 1.
Answer the following questions:

a) What is referential transparency? (10 points).

b) In your opinion, which limitations and advantages do you see in functional paradigm, compared to the imperative paradigm? (10 points).

Question 2.
Given the following function

```
def foo(n: Int): List[Int] = {
  if (n <= 0) Nil
  else foo(n-1) match {
    case Nil => n :: Nil
    case h :: t =>
      val x = h - n
      if (x > 0 && !t.contains(x)) x :: foo(n-1)
      else (h+n) :: foo(n-1)
  }
}
```

Do the following:

a) Decompose the call `foo(6)` by hand using the substitution model and show the result (5 points).

b) Convert it to a tail-recursive version (15 points).

Note: the method `contains` checks whether a given element exists within the list. It returns `true` if the element exists, `false` otherwise. The exclamation mark (!) is a negation, i.e., "not contains".

Question 3.

Write a generic function `counts` (i.e., one which uses universal polymorphism) which takes a list of elements *ls* of type `A`, and a function *f* of type `A => Boolean`, and returns a tuple where the first element contains the number for times *f* evaluated to `true` for all elements in *ls*, and the second the number of times it evaluated to `false`.

For example:

```
counts[Int](List(1, 2, 3, 4, 5), x => x < 3) // = (2, 3)
counts[Char]("ab cde f g".toList, x => x == ' ') // = (3, 7)
```

Your tasks:

a) Definition of the function `count` (15 points).

b) Make use of pattern matching within the function (5 points).

Question 4.

Consider the definition of an algebraic data type binary tree. The nodes of the tree are ordered objects (in terms of operations $<$, $>$ and $==$) and the leaves are empty (i.e., they do not contain any object). See Figure 1.

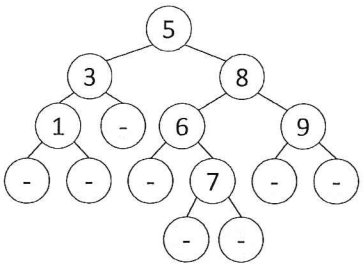

Figure 1: Binary tree with ordered numbers.

The tree is defined in such a way that for any node n (e.g., node with number 8 in Figure 1), the nodes to its `left` contain numbers that are smaller (e.g., nodes 6 and 7), whereas the nodes to its `right` contain numbers which are greater (e.g., node 9).
Your tasks:

a) Define a functional data structure for such binary tree. (10 points)

b) Define a method (i.e., a function as part of the defined data structure) `add` which takes a value of type Int, and returns a new tree with the value added. Note that adding a number that already exists in the tree should not make any "changes" to the tree, i.e., returns the same tree (7 points).

c) Give an example of the creation of the tree shown in Figure 1, using the add function (3 points).

Question 5.

In a Pascal triangle (see Table 1), a value n in row r and column c, is given by the following relation:

$$n^{r,c} = n^{r-1,c} + n^{r-1,c-1}$$

For example, value $n^{5,2}$ (row 5 and column 2) is given by $n^{5-1,2} + n^{5-1,2-1}$, which translates to $n^{4,2} + n^{4,1} = 3 + 1 = 4$. Your tasks:

a) Write a function `pascal` that takes a row and a column, and returns the corresponding value (8 points).

b) Write a function `pascalColumn` which takes a column value, and returns a `Stream[Int]` with all values for that given column (12 points).

1	1						
2	1	1					
3	1	2	1				
4	1	3	3	1			
5	1	4	6	4	1		
6	1	5	10	10	5	1	
7	1	6	15	20	15	6	1
	1	*2*	*3*	*4*	*5*	*6*	*7*

Table 1: Pascal triangle

Example calls:

```
pascal(7, 3)     // = 15
pascalColumn(3) // = Stream(1, ?)
pascalColumn(3).take(5).toList // = List(1, 3, 6, 10, 15)
```