KTH Information and Communication Technology

ID1019
Johan Montelius

Programming II (ID1019) template

Name: ______________________________

Instructions

- All answers should be written in these pages, use the space allocated after each question to write down your answer.
- Answers should be written in English.
- You should hand in the whole exam.
- No additional pages should be handed in.

Grade

The exam consists of two parts. A basic part of five questions (1-5) and one part for higher grade. The first five questions are about basic functional programming: pattern matching, recursion, immutable data structures etc. These five questions should be be answered satisfyingly (8 out of 10 points) for the grade of E (7 points for FX).

The remaining four questions (6-9) are about: semantics, higher order functions, complexity, processes etc. The higher grades are based only on these questions but is only given (and only corrected) if the basic part has been answered satisfyingly.

- D: 1 points
- C: 2 points
- B: 3 points
- A: 4 points

Namn:________________________________ Persnr:________________________________

1 double even [2p]

Implement a function that takes a list of numbers and returns a, equivalent list but one where all even numbers have been doubled. You can use the function `rem(n.k)` which returns the reminder when dividing n with k.

2 a binary tree [2p]

Implement a function, `sum/1`, that takes a binary tree and returns the sum of all values in the tree. The tree is represented as follows:

```
@type tree :: {:node, integer(), tree(), tree()} |  nil
```

Namn:________________________________ Persnr:________________________________

3 mirror a tree [2p]

Implement a function that mirrors a binary tree. A tree is mirrored by mirroring, and changing place on, its two branches.

4 add/2 [2p]

Implement the function `add/2` that adds an integer to a heap. A heap is a tree with the largest element in its root and where the two branches also are heaps.

To keep the heap balanced you should swith the left and right branches that is, when you add an element to a branch you add it to the right branch but make the result the left branch of the heap.

- `@spec add(heap(), integer()) :: heap()`

Namn:________________________________ Persnr:________________________________

5 Fizz-Buzz [2p]

We should implement a function `fizzbuzz/1` that given a number $n \geq 0$ returns a list of the n first elements in the fizz-buzz series. Fizz-buzz is a series from 1 to n where you replace all numbers that are a multiple of 3 by `:fizz`, those multiple by 5 by `:buzz` and those a multiple of both 3 and 5 by `:fizzbuzz`. The first five elements is thus: `[1,2,:fizz,4,:buzz]`.

You should implement the function `fizzbuzz/4` that helps us do this. The first argument is the next element in the list, the second tells us if we are done and the third and fourth keeps track of if the number is a multiple of 3 or 5. You are only allowed to use addition, no other arithmetic operation. You should not make the function tail recursive.

```
def fizzbuzz(n) do fizzbuzz(1, n+1, 1, 1) end
```

Namn:________________________________ Persnr:________________________________

6 Formal semantics [1p]

Lambda calculus

Evaluate the following lambda expressions:

- $(\lambda x \rightarrow x + 5)4$
- $(\lambda x \rightarrow (\lambda y \rightarrow x + 2 * y)3)5$
- $(\lambda x \rightarrow (x)5)(\lambda z \rightarrow z + z)$

Operational semantics

Something about operational semantics.

Namn:______________________________ Persnr:______________________________

7 reduce a tree [1p]

reduce/3

Implement a function `reduce(tree, acc, op)` that takes a binary tree, with values in the nodes, an initial value and an operation, `op`, that takes two arguments; a reduced value and the value of a node. The function `reduce/3` should return a reduced value that is computed recursivly:

- the reduced value of an empty tree is the accumulated value
- the reduced value of a node is the reduced value of the right branch - where the accumulated value is the operand applied to the reduced value of the left branch and the value of the node.

The tree is represented as follows:

```
@type tree :: {:node, any(), tree(), tree()} |  nil
```

to_list/1

Implement a function `to_list/1`, that takes a tree and returns a list of all elements in the tree in infix order. The function should be implemented using `reduce/3`.

Namn:______________________________ Persnr:______________________________

8 a queue [1p]

Assume that we represent a queue with the help of two lists and have the below implementation of `enqueue/2` and `dequeue/1`. What is the amortized time complexity for <u>adding and then removing an element from a queue</u>?

```
def enqueue({:queue, head, tail}, elem) do
  {:queue, head, [elem|tail]}
end

def dequeue({:queue, [], []}) do :fail end
def dequeue({:queue, [elem|head], tail}) do
  {:ok, elem, {:queue, head, tail}
end
def dequeue({:queue, [], tail}) do
  dequeue({:queue, reverse(tail), []})
end
```

Namn:______________________________ Persnr:______________________________

9 parallel sum [1p]

Implement a finction `sum/1` that takes a binary tree with numbers in the leafs, and sums all numbers of the tree <u>in parallel</u>.