KTH Information and
Communication Technology

ID1019
Johan Montelius

Programming II (ID1019)
2019-06-05

Instructions

- All answers should be written in these pages, use the space allocated after each question to write down your answer (not on the back side).
- 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. The first five questions are about basic functional programming: pattern matching, recursion, immutable data structures etc. The first part is the basic requirement to pass the course:

- FX: 7 points
- E: 8 points

The second part, 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 (8 out of 10 point).

- D: one question correctly answered
- C: two questions correctly answered
- B: three questions correctly answered
- A: all questions correctly answered

Namn:_____ Persnr:_____

1 drop every n'th [2p]

Implement a function, `drop/2`, that takes a list and a number $n > 0$ and returns a list where every n'th element has been removed.

Example: `drop([:a,:b,:c,:d,:e,:f,:g,:h,:i,:j], 3)` should give the answer `[:a,:b,:d,:e,:g,:h,:j]`.

2 rotate a list [2p]

Implement a function `rotate/2` that takes a list, of length l, and a number n, where $0 \leq n \leq l$, and returns a list where the elements have been rotated n steps.

You can use two library functions `append/2` and `reverse/1` (not `++`). Your solution can only call these functions ones each during an evaluation.

Example: `rotate([:a,:b,:c,:d,:e], 2)` returns `[:c,:d,:e,:a,:b]`.

Namn:_____ Persnr:_____

3 n'th leaf [2p]

Implement a function `nth/2` that finds the value of the n'th leaf in a binary tree traversed depth first left to right. The function shall take a number $n > 0$ and a tree and return either:

- `{:found, val}` if the n'th leaf is found and has the value `val` or
- `{:cont, k}` if only $n - k$ leafs were found i.e. you would need k more leafs to find the n'th leaf.

You should not transform the tree to a list and then find the n'th leaf. You should find the leaf by traversing the tree and stop as soon as you have found the n'th leaf.

Trees are represented as follows, note that there is no empty tree and that only the leaves have values:

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

Example:

```
nth(3, {:node, {:node, {:leaf, :a}
                       {:leaf, :b}}
               {:leaf, :c}})
```

should return `{:found, :c}`

Namn:_____ Persnr:_____

4 HP35 and reversed Polish notation [2p]

The worlds best calculator is of course the HP35 that uses reversed polish notation. You will not press `2 + 3 =` but `2 3 +` and imediately receives the result `5`. Each time a number was netered it was added to the stack. If you entered a binary operator the two uppermost elements on the stack was replaced by the result of the operation. If you entered `3 4 + 2 -` the answer was `5` and `3 4 + 2 1 + -` of course gave `4` as the result.

Implement a function `hp35/1` that takes a sequence of instructions and returns the result. The instructions consist of either numbers of operators and could of course be of arbitrary length. You don not have to handle illegal sequences, we assume all sequences represent valid expressions.

```
@type op() :: :add | :sub
@type instr() :: integer() | op()
@type seq() :: [instr()]

@spec hp35(seq()) :: integer()
```

Example `hp35([3, 4, :add, 2, :sub])` should return `5` since $(3+4)-2=5$.

Namn:_____ Persnr:_____

5 Pascal's triangel [2p]

Implement a function `pascal/1` that takes a number n (> 0) and returns the n'th row in Pascals triangle. Below you see how the triangle is constructed: the first row is `[1]` and the fifth row is `[1, 4, 6, 4, 1]`. The function should of course be able to generate any row not just the ones shown.

Think recursively.

```
              [1]
            [1 , 1]
          [1,  2,  1]
        [1,  3 , 3,  1]
      [1,  4,  6,  4,  1]
    [1, ...              1]
```

Example: `pascal(5)` returns `[1,4,6,4,1]`.

Namn:_____ Persnr:_____

6 Formal semantics [P/F]

order of evaluation

Elixir is a functional language where the arguments in a function call are evaluated before the function. This is what we call *eager evaluation*. Other functional languages use so called *lazy evaluation* where the arguments are evaluated only if needed.

If we stick to the functional subset of Elixir (without processes nor side-effects) then this might not mean anything in practice since the result we get is the same.

We can take the definition of `test/2` below as an example; the call `test(3+4, fib(17))` will result in `7` independent of which evaluation order we choose. Is it, in this case, irrelevant if choose eager or lazy evaluation? Motivate your answer.

Give an example where the result (or lack thereof) is different depending on if we use eager or lazy evaluation. Motivate your answer.

```
def test(one, two) do
   if  one > 10 do
      two
   else
      one
   end
 end
```

Namn:_____ Persnr:_____

7 Transform a tree [P/F]

Implement a function `trans/2` that takes a tree and a function, and returns an isomorphic tree where each leaf value has been transformed using the given function.

Then implement a function `remit/2` that takes a tree and a number n as arguments and returns, by using `trans/2`, a tree where each leaf value x has been replaced by the remainder when doing integer division by n (obtained by calling `rem(x,n)`).

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

@spec trans(tree(), (any()  -> any()))
```

Namn:_____ Persnr:_____

8 a one way ticket [P/F]

Going by train in Europe is popular right now so why not write a program that finds the shortest distance between two cities. We assume that we have a map that describes all lines between cities in Europe. Given a city we can by using the map find a list of all immediate neighbours and the distances to these cities.

The skeleton below (and next page) is a program that works even if the map contains cycles. It is also quite efficient since it is using dynamic programming. Complete the program.

```
@type city() :: atom()
@type dist() :: integer() | :inf

@spec shortest(city(), city(), map()) :: dist()

def shortest(from, to, map) do

      ....           =  Map.new([{to, 0}])

      ....           =  check (from, to, ....       ,....        )

  dist
end

@spec check(city(), city(), map(), map()) ::  {:fond, dist(), map()}

def check(from, to, ....       , ....         ) do

  case      ....                    do

    nil ->

      shortest(from, to, ....        , ....         )

    distance ->

      ....

  end
end
```

Namn:_____ Persnr:_____

```
@spec shortest(city(), city(), map(), map()) :: {:found, dist(), map()}

def shortest(from, to, ....     , ....        ) do

  ....                 = Map.put(....     , ....      , :inf)

  ....                 = Map.get(....     , from)

  ....                 =  select(neighbours, to, updated, map)

  ....

  {:found, dist, updated}
end

@spec select([{:city, city(), integer()}], city(), map(), map()) ::
         {:found, dist(), map()}

def select([], _,  ....    , ....      ) do  ....                end

def select([{:city, next, d1} | rest], to,   .....  , .....       )    do

        ....        = check(next, to,  ....     , ....          )

  dist = add(d1,d2)

        ....        = select(rest, to,  ....    , ....           )

  if sele  < dist do
    {:found, sele, updated}
  else
    {:found, dist, updated}
  end
end

@spec add(dist(), dist()) :: dist()

def add(  ....  , ....       ) do  ....        end
def add(  ....  , ....       ) do  ....        end
def add(  ....  , ....       ) do  ....        end
```

Namn:_____ Persnr:_____

9 HP35 revisited [P/F]

Since the HP35 is such a nice calculator you should implement a process that behaves as one. The process should be able to receive a sequence of messages that are either numbers or operations. You only have to implement addition but we wan the result returned to us when we perform an addition.

Implement the process and then provide a test function that shows how the process can be used to add to numbers.

Write your answer on the next page.

Namn:_____ Persnr:_____

Answer to question 9.

Namn:_____ Persnr:_____

Appendix

Map

- `put(map, key, value)` put(map(), key(), value()) :: map() Puts the given value under key in map.
- `get(map, key, default \\nil)` get(map(), key(), value()) :: value() Gets the value for a specific key in map. If key is present in map with value value, then value is returned. Otherwise, default is returned (which is `nil` unless specified otherwise).
- `new(enumerable)` new(Enumerable.t()) :: map() Creates a map from an enumerable. Duplicated keys are removed; the latest one prevails.