02157 Functional programming

Michael R. Hansen
DTU COMPUTE
December 5, 2017

Exercises for Dec. 8^{th}: Selected from old exam questions

Exercise 1 Extracted from exam May 30th, 2016

Consider the program skeleton:

```
let sumGt k xs = List.fold (...) ... xs;
```

Fill in the two missing pieces (represented by the dots ...), so that `sumGt` $k\,xs$ is the sum of those elements in xs which are greater than k. For example, `sumGt` $4\,[1;5;2;7;4;8] = 5+7+8 = 20$.

Exercise 2 Extracted from exam Dec. 20th, 2016

Consider the following F# declarations:

```
type 'a tree = | Lf
               | Br of 'a * 'a tree * 'a tree;;

let rec f(n,t) = match t with
                 | Lf            -> Lf
                 | Br(a, t1, t2) -> if n>0 then Br(a, f(n-1, t1), f(n-1, t2))
                                    else Lf;;

let rec g p = function
              | Br(a, t1, t2) when p a -> Br(a, g p t1, g p t2)
              | _                      -> Lf;;

let rec h k = function
              | Lf            -> Lf
              | Br(a, t1, t2) -> Br(k a, h k t1, h k t2);;
```

1. Give the types of `f`, `g` and `h`, and describe what each of these three functions compute. Your description for each function should focus on *what* it computes, rather than on individual computation steps.

Exercise 2 Extracted from exam May 30th, 2016

We shall now consider *containers* that can either have the form of a *tank*, that is characterized by it length, width and height, or the form of a *ball*, that is characterized by its radius. This is captured by the following declaration:

```
type Container = | Tank of int * int * int // (length, width, height)
                 | Ball of int              // radius
```

1. Declare two F# values of type `Container` for a tank and a ball, respectively.
2. A tank is called *well-formed* when its length, width and height are all positive and a ball is well-formed when its radius is positive. Declare a function `isWF` : `Container` → `bool` that can test whether a container is well-formed.
3. Declare a function `volume` c computing the volume of a container c. (Note that the volume of ball with radius r is $\frac{4}{3} \cdot \pi \cdot r^3$.)

A *cylinder* is characterized by its radius and height, where both must be positive integers.

4. Extend the declaration of the type `Container` so that it also captures cylinders, and extend the functions `isWF` and `volume` accordingly. (Note that the volume of cylinder with radius r and height h is $\pi \cdot r^2 \cdot h$.)

A *storage* consist of a collection of uniquely named containers, each having a certain *contents*, as modelled by the type declarations:

```
type Name     = string
type Contents = string
type Storage  = Map<Name, Contents*Container>
```

where the name and contents of containers are given as strings.

Note: You may choose to solve the below questions using a list-based model of a storage (`type Storage = (Name * (Contents*Container)) list`), but your solutions will, in that case, at most count 75%.

5. Declare a value of type `Storage`, containing a tank with name `"tank1"` and contents `"oil"` and a ball with name `"ball1"` and contents `"water"`.
6. Declare a function `find` : `Name` → `Storage` → `Contents` ∗ `int`, where `find` n stg should return the pair (cnt, vol) when cnt is the contents of a container with name n in storage stg, and vol is the volume of that container. A suitable exception must be raised when no container has name n in storage stg.

Exercise 3 Extracted from exam Dec. 17th, 2015

We consider the use of *appliances* (in Danish 'husholdningsapparater') like washing machines, dishwashers and coffee machines. A *usage* of an appliance a is a pair (a, t), where t is the time span (in hours) the appliance is used. A *usage list* is a list of the individual usages during a full day, that is, 24 hours. This is modelled by:

```
type Appliance = string
type Usage     = Appliance * int

let ad1 = ("washing machine", 2)
let ad2 = ("coffee machine", 1)
let ad3 = ("dishwasher", 2)
let ats = [ad1; ad2; ad3; ad1; ad2]
```

where `ats` is a value of type `Usage list` containing one usage of the dishwasher and two usages of the washing machine and the coffee machine.

1. Declare a function: `inv: Usage list -> bool`, that checks whether all time spans occurring in a usage list are positive.
2. Declare a function `durationOf: Appliance -> Usage list -> int`, where the value of `durationOf` *a ats* is the accumulated time span appliance a is used in the list *ats*. For example, `durationOf "washing machine" ats` should be 4.
3. A usage list *ats* is *well-formed* if it satisfies `inv` and the accumulated time span of any appliance in *ats* does not exceed 24. Declare a function that checks this well-formedness condition.
4. Declare a function `delete`(a, ats), where a is an appliance and *ats* is a usage list. The value of `delete`(a, ats) is the usage list obtained from *ats* by deletion of all usages of a. For example, deleting usage of the coffee machine from `ats` should give `[ad1; ad3; ad1]`. State the type of `delete`.

We now consider the *price* of using appliances. This is based on a *tariff* mapping an appliance to the price for one hour's usage of the appliance:

```
type Price  = int
type Tariff = Map<Appliance, Price>
```

5. Declare a function `isDefined` *ats trf*, where *ats* is a usage list and *trf* is a tariff. The value of `isDefined` *ats trf* is true if and only if there is an entry in *trf* for every appliance in *ats*. State the type of `isDefined`.
6. Declare a function `priceOf: Usage list -> Tariff -> Price`, where the value of `priceOf` *ats trf* is the total price of using the appliances in *ats*. The function should raise a meaningful exception when an appliance is not defined in *trf*.