Michael R. Hansen
DTU Compute, 22-11-2023

# Problem 1 from May 2018 (approx 48 minutes )

Consider the following F# declaration:

```
let rec f xs ys = match (xs,ys) with
                  | (x::xs1, y::ys1) -> x::y::f xs1 ys1
                  | _                -> [];;
```

1. Give an evaluation (using $\leadsto$) for `f [1;6;0;8] [0; 7; 3; 3]` thereby determining the value of this expression.
2. Give the (most general) type for `f`, and describe what `f` computes. Your description should focus on *what* it computes, rather than on individual computation steps.
3. The declaration of `f` is *not* tail recursive. Give a brief explanation of why this is the case and provide a declaration of a tail-recursive variant of `f` that is based on an accumulating parameter. Your tail-recursive declaration must be based on an explicit recursion.
4. Provide a declaration of a continuation-based, tail-recursive variant of `f`.

# Problem 2.1 from May 2017 (approx 20 minutes )

Consider the following F# declarations:

```
let rec f  = function
             | 0          -> [0]
             | i when i>0 -> i::g(i-1)
             | _          -> failwith "Negative argument"
and g = function
        | 0 -> []
        | n -> f(n-1);;

let h s k  = seq { for a in s do
                        yield k a };;
```

1. Give the values of `f 5` and `h (seq [1;2;3;4]) (fun i -> i+10)`. Furthermore, give the (most general) types for `f` and `h`, and describe what each of these two functions computes. Your description for each function should focus on *what* it computes, rather than on individual computation steps.

# Problem 3 from May 16 (approx 48 minutes)

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 float * float * float // (length, width, height)
    | Ball of float                 // 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` $\rightarrow$ `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 float numbers.

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` $\rightarrow$ `Storage` $\rightarrow$ `Contents` $*$ `float`, 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 $stg$.

# Problem 4 from May 16 (approx. 48 minutes)

Consider the following F# declarations of a type `T<'a>` for binary trees having values of type `'a` in nodes, three functions `f`, `h` and `g`, and a binary tree `t`:

```
type T<'a> = L | N of T<'a> * 'a * T<'a>

let rec f g t1 t2 =
   match (t1,t2) with
   | (L,L) -> L
   | (N(ta1,va,ta2), N(tb1,vb,tb2))
           -> N(f g ta1 tb1, g(va,vb), f g ta2 tb2);;

let rec h t = match t with
              | L            -> L
              | N(t1, v, t2) -> N(h t2, v, h t1);;

let rec g = function
            | (_,L)                    -> None
            | (p, N(t1,a,t2)) when p a -> Some(t1,t2)
            | (p, N(t1,a,t2))          -> match g(p,t1) with
                                          | None -> g(p,t2)
                                          | res  -> res;;

let t = N(N(L, 1, N(N(L, 2, L), 1, L)), 3, L);;
```

1. Give the type of `t`. Furthermore, provide three values of type `T<bool list>`.

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

3. Declare a function `count` $a\,t$ that can count the number of occurrences of $a$ in the binary tree $t$. For example, the number of occurrences of 1 in the tree `t` is 2.

4. Declare a function `replace`, so that `replace` $a\,b\,t$ is the tree obtained from $t$ by replacement of every occurrence of $a$ by $b$. For example, `replace 1 0 t` gives the tree `N(N(L, 0, N(N(L, 2, L), 0, L)), 3, L)`.