Programiranje I: 1. izpit

27. junij 2019

Čas reševanja je 150 minut. Veliko uspeha!

1. naloga

Define a type of complex numbers represented in cartesian coordinates as follows:

```
type complex = { re : float ; im : float }
```

a) Write a function

```
complex_add : complex -> complex -> complex
```

that adds two complex numbers.

b) Write a function

```
complex_conjugate : complex -> complex
```

that computes the conjugate of a complex number.

c) Write a function

```
get_range : 'a list -> int -> int -> 'a list
```

such that `get_range xs i k` extracts the sublist between indices `i` and `k` (exclusive) from `xs`.

d) Write a function

```
list_apply_either : ('a -> bool) -> ('a -> 'b) -> ('a -> 'b) -> 'a list -> 'b list
```

that takes a boolean predicate `pred`, two functions `f` and `g`, and a list `xs`, and applies either `f` or `g` to each element `x` of the list, depending on whether `x` satisfies `pred`.

e) We want to define a function that evaluates a polynomial at a point. A polynomial is represented as a list of coefficients in increasing power. For example $x^3 - 2x + 3$ is represented as `p = [3; -2; 0; 1]`. Define a function

```
eval_poly : int list -> int -> int
```

that takes a representation of a polynomial and a point, and evaluates it. For example, `eval_poly p 0 = 3`, and `eval_poly p 3 = 24`. This function should be tail-recursive.

You may use the following auxiliary function:

```
let rec power x n = if n <= 0 then 1 else x * (power x (n - 1))
```

2. naloga

The city council has asked us for help managing the community garden. The garden is divided in plots that can be allotted to a tenant. Each tenant can either choose to farm his plot, or to subdivide the plot and create one or more new sub-plots.

We define a type of plots as follows:

```
type tenant = string

type plot = Farmed of tenant
          | Sublet of tenant * (plot * plot list)
          | Vacant
```

a) Define the following example as a value `p_example` of type `plot`:
Tenant Hannah sub-divided her plot into three subplots:

- tenant Ian farms his plot
- one plot is vacant
- tenant Chris farms their plot

b) Write a function

```
farmed_opt : plot -> tenant option
```

that reveals the name of a tenant that directly farms a plot (without subletting it).

c) Write a function

```
depth : plot -> int
```

that computes the maximum number of successive subdivisions of a plot. For the above example, the result should be one.

d) Write a function

```
plot_unused : plot -> bool
```

that checks if there are any actual farmers on a plot, or if it is only sublet and left vacant.

e) Write a function

```
tenants : plot -> tenant list
```

that lists all the tenants involved with a plot.

f) Write a function

```
farmers : plot -> tenant list
```

that lists all the tenants that farm a plot.

3. naloga

Nalogo lahko rešujete v Pythonu ali OCamlu.

Given are different kinds of containers that we want to load on our boat. Each kind of container has a specific weight, for example 1 ton, 3 tons, 4 tons, 7 tons, 10 tons. The boat has a given capacity, say 5 tons, or 40 tons, or maybe 300 tons. We want to compute the number of different ways we can load the boat to exactly fill out its capacity.

We are free to choose containers of each kind as often as we like, and we do not care about the order in which the boat is loaded. Let us therefore assume that the containers are picked in a fixed order, starting with the smallest, and once we move on to loading larger containers, we do not move back to smaller ones.

For example, for a capacity of 5 tons, there are three solutions: we can fill the boat with 5 containers of weight 1, or we can fill it with 2 containers of weight 1 and 1 container of weight 3, or with 1 container of weight 1 and 1 container of weight 4.

Fix a global variable containing the list of container weights, for example

```
containers = [1, 3, 4, 7, 10]
```

Write a function that takes as input the capacity of a boat, and computes the number of different ways to fill the boat with the given containers.

For bonus points, write a function that computes the solutions instead of only counting them.