Programiranje I: 1. Izpit

28. januar 2020

Čas reševanja je 150 minut. Veliko uspeha!

V nalogah si lahko pomagate z rešitvami prejšnjih nalog.

1. naloga

a) Definirajte funkcijo `option_sum: int option -> int option -> int option`, ki vrne vsoto argumentov, če je to mogoče, ali pa `None`, če je katerikoli od argumentov `None`.

```
# option_sum (Some 1) None;;
- : int option : None
```

b) Imamo funkcijo `strange_map f l r x`, kjer velja: `f: ('a -> 'b * 'c)`, `l: ('b -> 'd)`, `r: ('c -> 'e)` in `x : 'a`. Funkcija `x` s funkcijo `f` preslika v par in na komponentah ustrezno uporabi `l` in `r`. Definirajte ustrezno funkcijo.

c) Definirajte funkcijo `function_repeat: ('a -> int) -> 'a list -> 'a list`, ki sprejme funkcijo `f` in seznam ter vrne nov seznam, kjer se posamezen element `x` iz začetnega seznama ponovi `f x` krat. Nepozitivne ponvitve pomenijo, da elementa v končnem seznamu ni. Za vse točke naj bo funkcija repno rekurzivna, kar tudi argumentirajte.

```
# function_repeat (fun x -> x) [0;1;2;(-2)];;
- : int list = [1;2;2]
```

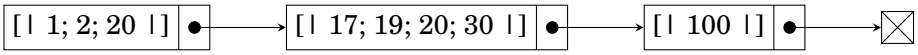
d) Definirajte funkcijo `iterate: ('a -> 'a) -> ('a -> bool) -> 'a -> 'a`, ki sprejme funkcijo za iteriranje, zaustavitveni pogoj in začetno vrednost. Funkcija naj funkcijo za iteriranje zaporedoma uporablja, dokler ne velja zaustavitveni pogoj. Funkcija naj vrne prvi rezultat, pri katerem zaustavitveni pogoj vrne `true`.

```
# iterate (fun x -> 0.5*.(x+.10.0/.x)) (fun x -> abs_float(x*.x-.10.0)<0.01) 10.0;;
- : float = 3.16245562280389
```

2. naloga

Napreden povezan seznam je podoben običajnemu seznamu v OCaml-u, le da imamo namesto ene vrednosti v posameznem vozlišču tabelo, ki lahko vsebuje več elementov (velikosti niso nujno enake). Enako kot običajen povezan seznam je sestavljen in dveh različnih gradnikov: praznega seznama in vozlišča s tabelo in preostankom seznama.

a) Definirajte polimorfen tip `'a improved_list` ter seznam `test : int improved_list`, ki predstavlja spodnji izboljšan seznam:



b) Definirajte funkcijo `len: 'a improved_list -> int`, ki vrne dolžino podanega seznama.

c) Definirajte funkcijo `index: 'a improved_list -> int -> 'a option`, ki vrne i-ti element, ali `None`, če le ta ne obstaja. Za vse točke mora biti funkcija repno rekurzivna.

```
# index test 5;;
- : int option = Some 20
```

d) Definirajte funkcijo `is_sorted: ('a -> 'a -> bool) -> 'a improved_list -> bool`, ki sprejme funkcijo primerjanja in vrne `true`, če je napreden seznam urejen glede na to funkcijo. Za vse točke mora biti funkcija repno rekurzivna in imeti linearno časovno zahtevnost.

```
# is_sorted (<=) test;;
- : bool = false
```

e) Definirajte funkcijo `update: 'a improved_list -> int -> 'a -> 'a improved_list`, ki vrne nov izboljšan seznam, kjer vrednost na indeksu drugega argumenta nadomesti z vrednostjo tretjega argumenta. Če je indeks nesmiselen, naj funkcije vrne kar originalni seznam. Pazite, da pri tem originalni seznam ostane nespremenjen. Za vse točke mora biti funkcija repno rekurzivna, kar v komentarju tudi argumentirajte.

```
# index (update test 5 (-3)) 5;;
- : int option = Some (-3)
# index test 5;;
- : int option = Some 20
```

3. naloga

a) Na mizo dolžine n si želimo postaviti dekoracijo iz m pravokotnih posod za rože, kjer je vsaka posoda dolžine l. Posode za rože postavljamo eno za drugo, med dvema zaporednima posodama pa mora biti vsaj 1 enota mize prazna. Sestavite funkcijo, ki sprejme n, m in l in vrne število različnih postavitev posod za rože na mizo, kjer moramo vedno porabiti vse posode in posod med seboj ne razlikujemo.

b) Dolžine korit so lahko različne. Sestavite funkcijo, ki sprejme število m in seznam celih števil, ki predstavlja dolžine posod za rože in vrne število različnih postavitev posod na mizo, kjer je vrstni red korit določen z vrstnim redom v podanem seznamu. (Ta podnaloga je lažja oblika prejšnje, če je rešitev popolnoma pravilno šteje tudi kot pravilna rešitev prejšnje podnaloge.)