Komparativna analiza programskih jezika

Test 2

Upute:

Rješenja šaljete na mail abolic@pmf.unsa.ba. U naslov maila obavezno staviti **"KAPJ test 2: ime, prezime, broj indeksa"**. Sve zadatke spasiti u **jedan .rkt fajl.** Provjerite da li je mail poslan na ispravnu email adresu, u suprotnom vam se ispit ne priznaje.

Dozvoljeno je korištenje **isključivo** oficijelne Racket dokumentacije: https://docs.racket-lang.org/

1. **(7 bodova)** Tip za binarno stablo koje sadrži cijele brojeve definisan je sa dvije strukture:

```
(struct node (left num right) #:transparent)
(struct empty () #:transparent)
```

   a) **(2 boda)** Implementirati funkciju ***forall*** koja prima dva parametra: funkciju *f* i binarno stablo *s.* Funkcija se treba evaluirati na *#f* ako i samo ako postoji broj *e* u stablu *s* za koji se *f(e)* evaulira na *#f.* Ako je *s* prazno stablo, funkcija se evaulira na *#t.*
   b) **(2 boda)** Koristeći funkciju ***forall*** implementirati funkciju ***all_in_range*** koja prima dva broja *a, b* i stablo *s*, te se evaulira na *#t* ako svi brojevi u stablu *s* pripadaju intervalu *[a, b].*
   c) **(3 boda)** Koristeći funkciju ***forall*** implementirati funkciju ***num_nodes*** koja prima stablo *s* te vraća broj čvorova u tom stablu.

2. **(6 bodova)** Stream definišemo kao thunk koji kada se pozove vraća par gdje je *cdr* tog para ponovo stream, a *car* od para je neka vrijednost koju generišemo.
   a) **(1 boda)** Implementirati stream koji generiše brojeve oblika $(-1)^k \cdot (4k+1)$ za sve $k = 1,2,3, \ldots$.
   b) **(2 boda)** Implementirati funkciju ***enumerate*** koja prima stream *s* proizvoljnog tipa elemenata i vraća novi stream *v*. Ako je stream *s* na *i*-toj poziciji proizveo element *e*, stream *v* na *i*-toj poziciji treba proizvesti par *(i e)*. Funkciju testirati sa streamom iz dijela pod a). Indeksiranje počinje od $i = 0$.
   c) **(3 boda)** Implementirati funkciju ***greater_than_previous*** koja prima stream *s,* a vraća stream *v* koji sadrži sve elemente iz *s* koji su veći od svih prošlih elemenata u streamu *s.* Funkciju testirati sa streamom iz dijela pod a).

3. **(7 bodova)** Niz *Tribonaccijevih brojeva* je definisan sa $a_0 = 0, a_1 = 0, a_2 = 1$ i $a_i = a_{i-1} + a_{i-2} + a_{i-3}$ za sve $i \geq 3$. Implementirati funkciju ***calculate_Tribonacci*** koja prima prirodan broj n te vraća n-*ti* član niza a_n. U implementaciji koristiti princip memoizacije. Zatim, implementirati funkciju ***first_N_Tribonacci*** koja prima prirodan broj n i prirodan broj k i koja kreira i vraća thunk pomoću *delay* metode koji sadrži listu od prvih n članova niza koji su veći od k . Thunk koji funkcija vrati treba da se može evaluirati na običnu listu koristeći *force* metodu.

4. **(5 bodova)** Implementirati macro ***partial*** koji prima izraz ***call*** kojem nedostaje zadnji argument. Poziv macro-a partial vraća funkciju koja prima jedan parametar *x* i koja vraća rezultat evaluacije *call* čiji su parametri prošireni parametrom *x* na kraju. Na primjer, poziv ***(define add_ten (partial (+ 5 5))*** definiše funkciju *add-ten* jednaku ***(lambda (x) (+ 5 5 x))***, ili na primjer poziv ***(define eq_ten (partial (equal? (+ 5 5))))*** definiše funkciju *eq-ten* jednaku ***(lambda (x) (equal? (+ 5 5) x))***.

   Hint: moguće je spajati dva *quote* izraza pomoću *append* funkcije, npr. poziv *(append (quote (+ 2 3)) (quote (4 5))* daje kao rezultat *(quote (+ 2 3 4 5)).*