Teknillinen korkeakoulu
Tietojenkäsittelyopin laboratorio
Seppo Törmä

23.10.2000

Tik-76.010 Ohjelmoinnin peruskurssi T1: Osatentti A

Please note: English translation of the examination questions is available.

Kirjoita jokaisen vastauspaperiin

- Tik-76.010 Ohjelmoinnin peruskurssi T1
- Osatentti A 23.10.2000
- nimi
- opiskelijanumero
- koulutusohjelma ja vuosikurssi
- vastauspaperin numero, esim. 'Paperi 1/3' tarkoittaa ensimmäistä paperia kolmen palautetun paperin nipussa.

Voit vastata kysymyksiin suomeksi, ruotsiksi tai englanniksi. Pyri selkeisiin ja melko lyhyisiin vastauksiin. Liian pitkistä ja sekavista vastauksista voi menettää pisteitä.

1. Vastaa seuraaviin kysymyksiin lyhyesti; kussakin kohdassa vastauksen pituus saa olla korkeintaan viisi riviä (1 piste/kohta, yhteensä 6 pistettä).

   a) Kirjoita lauseke `(let ((a 2)) (+ a a))` siten, että käytät let-muodon sijasta lambdaa.

   b) Mitä tarkoittaa korkeamman kertaluokan proseduuri (*higher-order procedure*)?

   c) Mikä ero on sidotulla muuttujalla (*bound variable*) ja vapaalla muuttujilla (*free variable*)?

   d) Piirrä lausekkeen `(append (list (cons 'a 'b) (list 'b 'c)) (list (cons 'd 'e)))` tuottaman rakenteen visuaalinen laatikko-osoitin-esitys (*box-and-pointer representation*).

   e) Miksi proseduuri `(define (foo n m) (if (= n 0) m (foo (- n 1) (* n m))))` synnyttää iteratiivisen laskentaprosessin (*iterative process*)?

   f) Mitä tarkoitetaan, jos sanotaan, että "proseduurin `(foo n)` tilankäytön kasvuluokka (*order of growth*) on $\Theta(n)$"?

2. Ympäristömalli.

   Kirjoita essee ympäristömallista. Voit käsitellä aihetta vapaamuotoisesti, mutta pyri vastaamaan ainakin seuraaviin kysymyksiin:

   - Mikä vaihtoehtoinen malli ympäristömallille kirjassa on esitetty ja mitä ongelmia siinä on?
   - Minkälaisia ympäristöt ovat rakenteeltaan?
   - Minkälaisia proseduurioliot ovat rakenteeltaan?
   - Mitkä neljä operaatiota ympäristömalli selittää?
   - Miten kukin näistä operaatioista ympäristömallin mukaan suoritetaan?

   (6 pistettä)

3. Kerro kussakin alla olevassa kohdassa, minkä arvon Scheme-tulkki palauttaa viimeisen lausekkeen evaluoimisen jälkeen? Huomaa, että kutakin kohtaa voidaan ajatella toisistaan erillisenä Scheme-tulkin käyttösessiona, eli määritelmät eivät näy kohdasta toiseen. (1 piste/kohta, yhteensä 6 pistettä)

a)
```
(let ((a 1) (b 10))
  (+ (let ((a b) (b (+ a 1)))
       (* a b))
     (- a b)))
```

b)
```
(define d 'b)
(define c 'd)
(define b d)
(define a c)
(list a 'b c d)
```

c)
```
(define (foo x)
  (if (not (pair? x))
      x
      ((cadr x) (car x) (caddr x))))
(foo (list (list 2 / 4) + (list 6 - 2)))
```

d)
```
(define a (list 3 2 1 0))
(set-cdr! a a)
a
```

e)
```
(define (g x)
  (lambda (m)
    (cond ((eq? m 'foo) x)
          ((eq? m 'bar) (* x 2)))))
(- ((g 10) 'bar) ((g 1) 'foo))
```

f)
```
(define (foo m) (map list (map cadr m)))
(foo '((1 2 3) (2 3 4) (3 4 5)))
```

Vastauksesta on käytävä selvästi ilmi tulkin antama vastaus. Jos tulkin antamaa vastausta ei kokonaan pysty kirjoittamaan paperille, esitä siinä tapauksessa kuinka vastaus alkaa.

Alleviivaa tulkin antama vastaus, jos haluat lisätä vastaukseen selitystä. Selittäminen ei ole tarpeellista, vaan oikea tulkin vastaus riittää täysiin pisteisiin.

4. a) Kirjoita proseduuri `deep-remove-if`, joka poistaa (mahdollisesti sisäkkäisistä listoista koostuvasta) listarakenteesta kaikki annetun testin täyttävät lehtialkiot. Proseduuria on tarkoitus käyttää seuraavien esimerkkien mukaisesti:

```
(deep-remove-if '(1 2 (3 2 4) 5 6) even?)
(1 (3) 5)
(deep-remove-if '(a (b (c d)) a b (c d))
                (lambda (x) (or (eq? x 'b) (eq? x 'c))))
(a ((d)) a (d))
```

Proseduuri `deep-remove-if` ottaa siis kaksi argumenttia:

- *listarakenne*, joka on mahdollisesti sisäkkäisistä listoista koostuva lista; kaikki rakenteeseen sisältyvät listat ovat oikeita listoja (*proper lists*), eli päättyvät aina pariin, jonka cdr on nil
- *testi* on proseduuri, joka ottaa yhden argumentin ja palauttaa totuusarvon false, mikäli listarakenteen alkio ei täytä testiä; muuten testi palauttaa jonkin falsesta poikkeavan arvon.

Tee proseduuri funktionalisesti, eli älä käytä mutatoijia (set!, set-car!,...).

Vihje: Jos tehtävä tuntuu vaikealta, aloita tekemällä yksitasoisille listoille toimiva proseduuri ja mieti, mitkä muutokset siihen täytyy tehdä, jotta se toimisi myös sisäkkäisten listojen kanssa.

(3 pistettä)

b) Seuraavan proseduurin tarkoituksena on laskea listarakenteessa olevien parien lukumäärä.

```
(define (count-pairs s)
   (if (not (pair? s))
       0
       (+ (count-pairs (car s))
          (count-pairs (cdr s))
          1)))
```

Tämä proseduuri **ei** laske parien lukumäärää kaikissa tapauksissa oikein. Oletetaan nyt, että tulkissa on tehty määritelmä:

```
(define a (list 1 2 3))
```

Tämä määrittelee listarakenteen, jossa on 3 paria. Tässä tilanteessa ylläesitetty määritelmä toimii aiotulla tavalla:

```
(count-pairs a)
-> 3
```

Kirjoita sellainen Scheme-lauseke, joka ei luo uusia pareja (ei siis käytä cons- tai list-proseduureja), mutta joka muuttaa a:n arvona olevaa listarakennetta siten, että `(count-pairs a)` palauttaa arvon 5.

Piirrä lisäksi laatikko-osoitin-kaavio (*box-and-pointer diagram*) syntyvästä listarakenteesta.

(3 pistettä)

5. Tässä tehtävässä toteutetaan joukkotietotyyppi, jossa joukko esitetään oliokeskeisesti määriteltynä binääripuuna. Joukon alkiot ovat numeroita ja ainoat operaatiot, joita tarvitaan, on alkion lisääminen joukkoon sekä sen testaaminen, kuuluuko alkio joukkoon.

   Binääripuun kaikki solmut ovat samanlaisia olioita, joita voidaan muodostaa `make-node` –konstruktorilla. Solmulla on kolme tilamuuttujaa:

   - `entry`: solmun sisältö
   - `left-branch`: solmun vasen alipuu
   - `right-branch`: solmun oikea alipuu

   Binääripuun tulee täyttää seuraavat ehdot:

   - kaikki solmun vasemmassa alipuussa olevat alkiot ovat (aidosti) pienempiä kuin solmun sisältö
   - kaikki solmun oikeassa alipuussa olevat alkiot ovat (aidosti) suurempia kuin solmun sisältö

   Binääripuun solmuille voidaan lähettää viestejä:

   - `element-of-set?`: `((node 'element-of-set?) element)` palauttaa `true`, jos `element` kuuluu solmun edustamaan joukkoon ja muuten `false`
   - `adjoin-set!`: `((node 'adjoin-set!) element)` muuttaa puuta sivuvaikutuksellisesti siten, että siinä on lisäyksen jälkeen alkio `element` binääripuun ehdot täyttävässä paikassa. Proseduurin paluuarvo ei ole tärkeä (eli se voi eri tilanteissa palauttaa sen mikä on helpointa).

   Tehtävässä ei tarvitse huolehtia puun tasapainottamisesta.

   Puuta pitää pystyä käsittelemään viestejä lähettämällä seuraavan esimerkin mukaisesti:

```
(define a (make-node 5))
((a 'element-of-set?) 5)
true
((a 'element-of-set?) 4)
false
((a 'adjoin-set!) 4)
((a 'element-of-set?) 4)
true
```

   Vihje: Solmun määrittely (ja koko tehtävän ratkaisu) tehdään kirjoittamalla konstruktori `make-node`, joka ottaa yhden argumentin (`entry`) ja jossa määritellään muut tilamuuttujat, tarpeelliset paikalliset proseduurit sekä viestien käsittely.

   (6 pistettä)