Lehrstuhl für Kognitive Modellierung
Prof. Dr. Martin V. Butz
Dr. Jan Kneissler

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Informatik II

Programmierübungsblatt 1 - LÖSUNG

Ausgabe: Do, 17.04.2014 Abgabe: 17te Kalenderwoche

Für die Bearbeitung der Übungsaufgaben haben Sie 90 Minuten Zeit. Bitte geben Sie Ihre Lösungen der Aufgaben rechtzeitig im Vorlesungssystem unter dem Punkt Präsenzübungen ab. Dazu müssen Sie sich mit Ihrem Benutzernamen und Passwort am Vorlesungssystem anmelden. Sie können mehrmals innerhalb der Bearbeitungszeit Ihre Lösungen abgeben, nur die letzte Version Ihrer Abgabe wird gespeichert. Wenn Sie mit den Übungen fertig sind, bleiben Sie bitte an ihrem Platz sitzen und warten, bis ein Tutor ihre Lösungen bewertet. **Bei Fragen zur Rechnerbenutzung oder zu den Übungsaufgaben wenden Sie sich bitte jederzeit an einen der TutorInnen!**

Aufgaben (insgesamt 22 Punkte)

Notiz: Halten Sie sich beim Programmieren in Scheme an die eingeführte Konstruktionsanleitung:
a) Kurzbeschreibung als Kommentar (mittels: "; ..."");
b) Signatur (der "Vertrag"mittels: "(: ...)");
c) Testfälle (mittels "(check-within ...)öder "(check-expect ...)öder ...)
d) Prozedur Gerüst und Rumpf
Benutzen Sie Hilfsprozeduren, wenn Teilprobleme gelöst werden müssen!

- LÖSUNG

1.1 Pyramide-Kantenlänge (10 Punkte)

Schreiben Sie Prozeduren, die die Kantenlänge einer Pyramide berechnen! Von der Pyramide ist die Seitenlänge der Grundfläche a und die Höhe h gegeben.
Die Kantenlänge einer Pyramide ist die Summe der Kantenlänge der quadratischen Grundfläche (mit Seitenlänge a) und der Länge der vier nach oben gehenden Kanten s. Schreiben Sie Prozeduren, die folgende Teilprobleme lösen:

a) Schreiben Sie eine Prozedur `square-circumference`, die den Umfang der quadratischen Grundfläche berechnet.

b) Um h_s und s zu berechnen, brauchen Sie jeweils den *Satz des Pythagoras*: $a^2 + b^2 = c^2$
Schreiben Sie eine Prozedur `pythagoras`, die das c der obigen Gleichung berechnet. Erkennen und abstrahieren Sie weitere Teilprobleme!

c) Schreiben Sie eine Prozedur `pyramid-one-edge-length`, die die Länge s berechnet (dazu brauchen Sie die Länge h_s).

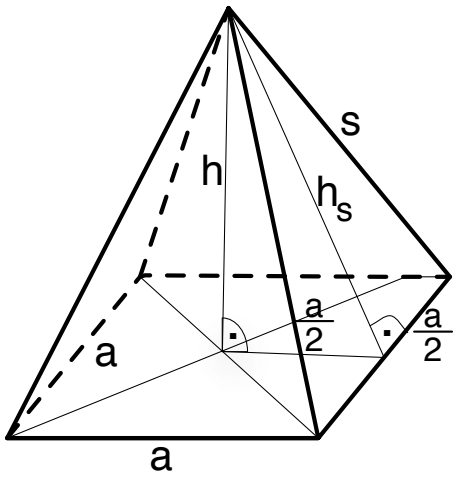

Abbildung 1: Pyramide

d) Schreiben Sie schließlich eine Prozedur `pyramid-edge-length`, die die Kantenlänge der Pyramide berechnet. Benutzen Sie dafür die bisher geschriebenen Prozeduren.

Hinweis: Zur Lösung können Sie die eingebaute Scheme-Prozedur `sqrt`, die die Signatur `(number -> number)` besitzt, verwenden: `(sqrt x)` liefert die Quadratwurzel von x.

Abgabe: Programm `pyramid-edge.rkt`

Lösungsvorschlag:

Punkteverteilung:

- 4 Punkte Kurzbeschreibungen, Verträge & Testfälle
- 1 Punkt Gerüst & Rumpf `square-circumference`
- 2 Punkte Gerüst & Rumpf `pythagoras`
- 2 Punkte Gerüst & Rumpf `pyramid-one-edge-length`
- 1 Punkt Gerüst & Rumpf `pyramid-edge-length`

```
;; Die ersten drei Zeilen dieser Datei wurden von DrRacket eingefügt. Sie enthalten Metadaten
;; über die Sprachebene dieser Datei in einer Form, die DrRacket verarbeiten kann.
#reader(lib "DMdA-beginner-reader.ss" "deinprogramm")((modname pyramid-edge) (read-case-sensitive #f)
     (teachpacks ()) (deinprogramm-settings #(#f write repeating-decimal #f #t none explicit #f ())))
; compute the circumference of a square with side length a
(: square-circumference (real -> real))

(check-expect (square-circumference 1) 4)
(check-within (square-circumference 0.25) 1 0)

(define square-circumference
  (lambda (side-length)
    (* 4 side-length)))

; compute the square of a number
(: square (number -> number))
(check-expect (square 3) 9)
(check-expect (square 1) 1)
(check-within (square 0.5) 0.25 0.000001)

(define square
  (lambda (x)
    (* x x)))

; solve formula of pythagoras
(: pythagoras (real real -> real))
(check-within (pythagoras 1 1) (sqrt 2) 0.0001)
(check-within (pythagoras 3 3) (sqrt 18) 0.00001)

(define pythagoras
  (lambda (a b)
    (sqrt (+ (square a) (square b)))))

; compute the pyramid side length s
(: pyramid-side-length (real real -> real))
(check-within (pyramid-side-length 5 10) 10.6 0.01)
(check-within (pyramid-side-length 2 1) 1.73 0.01)

(define pyramid-side-length
  (lambda (a h)
    (pythagoras (/ a 2)
                (pythagoras (/ a 2) h))))

; compute the overall pyramid edge length
(: pyramid-overall-edge-length (real real -> real))
(check-within (pyramid-overall-edge-length 5 10) 62.42 0.01)
(check-within (pyramid-overall-edge-length 2 1) 14.92 0.01)
(check-within (pyramid-overall-edge-length 23 5)
              (+ (* 4 23)
                 (* 4 (pyramid-side-length 23 5)))
              0.0001)

(define pyramid-overall-edge-length
  (lambda (a h)
    (+ (square-circumference a)
       (* 4 (pyramid-side-length a h)))))
```

Listing 1: Pyramidenproblem - Lösung

- LÖSUNG

1.2 Bussgelder (12 Punkte)

Schreiben Sie ein Programm, mit dem Sanktionen bei Verkehrsverstössen bestimmt werden.

a) (3 Punkte) Programmieren Sie eine Prozedur `zu-langes-parken` für die Bewertung von zu langem Parken auf einem kostenpflichtigen Parkplatz. Die Prozedur bekommt eine Zeitspanne in Minuten übergeben und gibt das entsprechende Verwarngeld zurück.

Diese Verwarnungen sind wie folgt festgelegt:

- Überschreitung der Höchstparkdauer bis einschließlich 30 Minuten: €5
- bis zu einer Stunde: €10
- bis zu zwei Stunden: €15
- bis zu drei Stunden: €20

- länger als drei Stunden: €25

Test: `(check-expect (zu-langes-parken 55) 10)`

b) (9 Punkte - jeweils 3) Das Überfahren einer roten Ampel kostet je nach Gefährdungslage mehr, gibt Punkte und Fahrverbot. Schreiben Sie eine Prozedur `rote-ampel-bussgeld`, die das Bußgeld berechnet und eine Prozedur `rote-ampel-punkte` für die Punkte in Flensburg. Schreiben Sie außerdem eine Prozedur `rote-ampel-fahrverbot`, die ausgibt, ob ein Fahrverbot erteilt wird. Übergeben Sie den Prozeduren, wie lange die Ampel schon rot war (in Sekunden) und ob eine Gefährdung oder Sachbeschädigung vorlag.

Die Sanktionen sind wie folgt definiert:

- Bei Rot über die Ampel innerhalb der ersten Sekunde: €50 und 3 Punkte
- Bei Rot über die Ampel innerhalb der ersten Sekunde mit Gefährdung oder Sachbeschädigung: €125, 4 Punkte und 1 Monat Fahrverbot
- Bei Rot über die Ampel nach der ersten Sekunde: €125, 4 Punkte und 1 Monat Fahrverbot
- Bei Rot über die Ampel nach der ersten Sekunde mit Gefährdung oder Sachbeschädigung: €200, 4 Punkte und 1 Monat Fahrverbot

Tests:
```
(check-expect (rote-ampel-bussgeld 1 #f #t) 125)
(check-expect (rote-ampel-fahrverbot 15 #f #f) #t)
(check-expect (rote-ampel-punkte 1 #f #f) 3)
```

Befolgen Sie die Konstruktionsanleitung für Prozeduren und Fallunterscheidungen! Achten Sie bei den Testfällen auf eine vollständige Überdeckung!

Hinweis: Zur Lösung können Sie die eingebauten Racket-Prozeduren `and` und `or` verwenden.

Abgabe: Programm `bussgeld.rkt`

Lösungsvorschlag:

```
;; Die ersten drei Zeilen dieser Datei wurden von DrRacket eingefügt. Sie enthalten Metadaten
;; über die Sprachebene dieser Datei in einer Form, die DrRacket verarbeiten kann.
#reader(lib "DMdA-beginner-reader.ss" "deinprogramm")((modname bussgeld) (read-case-sensitive #f)
     (teachpacks ()) (deinprogramm-settings #(#f write repeating-decimal #f #t none explicit #f ())))
; Berechnet das Verwarngeld, wenn zu lange geparkt wird.
(: zu-langes-parken (natural -> natural))
(check-expect (zu-langes-parken 0) 0)
(check-expect (zu-langes-parken 15) 5)
(check-expect (zu-langes-parken 30) 5)
(check-expect (zu-langes-parken 55) 10)
(check-expect (zu-langes-parken 65) 15)
(check-expect (zu-langes-parken 120) 15)
(check-expect (zu-langes-parken 180) 20)
(check-expect (zu-langes-parken 181) 25)

(define zu-langes-parken
  (lambda (ueberschreitung)
    (cond ((= ueberschreitung 0) 0)
          ((<= ueberschreitung 30) 5)
          ((<= ueberschreitung 60) 10)
          ((<= ueberschreitung 120) 15)
          ((<= ueberschreitung 180) 20)
          (else 25))))

; gibt das Bussgeld beim Überfahren einer roten Ampel aus
; (abhaengig von der Zeit, die die Ampel schon rot ist und
; dem Vorliegen von Gefährdungen und Sachbeschädigungen)
(: rote-ampel-bussgeld (natural boolean boolean -> integer))

(check-expect (rote-ampel-bussgeld 0 #t #t) 0)
(check-expect (rote-ampel-bussgeld 1 #f #f) 50)
(check-expect (rote-ampel-bussgeld 1 #f #t) 125)
(check-expect (rote-ampel-bussgeld 1 #t #t) 125)
(check-expect (rote-ampel-bussgeld 15 #f #f) 125)
(check-expect (rote-ampel-bussgeld 13 #t #f) 200)
(check-expect (rote-ampel-bussgeld 23 #t #t) 200)
```

```
(define rote-ampel-bussgeld
  (lambda (zeit-rot gefährdung sachbeschädigung)
    (cond ((= zeit-rot 0) 0)
          ((= zeit-rot 1)
           (if (and (not gefährdung) (not sachbeschädigung))
               50
               125))
          ((> zeit-rot 1)
           (if (and (not gefährdung) (not sachbeschädigung))
               125
               200)))))

; Berechnet, ob ein Fahrverbot erteilt wird.
(: rote-ampel-fahrverbot (natural boolean boolean -> boolean))

(check-expect (rote-ampel-fahrverbot 0 #f #f) #f)
(check-expect (rote-ampel-fahrverbot 1 #f #f) #f)
(check-expect (rote-ampel-fahrverbot 1 #t #f) #t)
(check-expect (rote-ampel-fahrverbot 1 #t #t) #t)
(check-expect (rote-ampel-fahrverbot 15 #f #f) #t)
(check-expect (rote-ampel-fahrverbot 13 #t #f) #t)
(check-expect (rote-ampel-fahrverbot 23 #t #t) #t)

(define rote-ampel-fahrverbot
  (lambda (zeit-rot gefährdung sachbeschädigung)
    (not (or (= zeit-rot 0)
             (and (= zeit-rot 1)
                  (not gefährdung)
                  (not sachbeschädigung))))))

; Berechnet, wie viele Punkte in Flensburg erteilt werden
(: rote-ampel-punkte (natural boolean boolean -> natural))

(check-expect (rote-ampel-punkte 0 #f #f) 0)
(check-expect (rote-ampel-punkte 1 #f #f) 3)
(check-expect (rote-ampel-punkte 1 #t #f) 4)
(check-expect (rote-ampel-punkte 1 #t #t) 4)
(check-expect (rote-ampel-punkte 15 #f #f) 4)
(check-expect (rote-ampel-punkte 13 #t #f) 4)
(check-expect (rote-ampel-punkte 23 #t #t) 4)

(define rote-ampel-punkte
  (lambda (zeit-rot gefährdung sachbeschädigung)
    (cond ((= zeit-rot 0) 0)
          ((and (= zeit-rot 1) (not gefährdung) (not sachbeschädigung)) 3)
          (else 4))))
```

Listing 2: Bußgelder - Lösung