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

Eberhard Karls
Universität
Tübingen

Informatik II

Programmierübungsblatt 3

Ausgabe: Mo, 12.05.2014 Abgabe: 20te 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 24 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 ...)" oder "(check-expect ...)" oder ...)
d) Prozedur Gerüst und Rumpf
Benutzen Sie Hilfsprozeduren, wenn Teilprobleme gelöst werden müssen!

Notiz: Die aktuelle `Info2_SchemeEssentials_XX.rkt` Datei steht Ihnen als weitere Hilfestellung zur Verfügung.

3.1 Primzahlen Sieben [13]

In dieser Aufgabe werden Sie das *Sieb des Erathostenes* (auch: *Zahlensieb*) implementieren. Das Zahlensieb ist eine der bekanntesten Möglichkeiten, alle Primzahlen aus der Folge der natürlichen Zahlen zu ermitteln. Gegeben ist eine Teilmenge der natürlichen Zahlen: $L = \{2, .., n\} \subset \mathbb{N}$.
Aus der Menge L werden nun alle Vielfachen der Zahl 2 entfernt. In jedem weiteren Schritt werden die Vielfachen der verbleibenden Elemente der Menge L entfernt. Beispiel (markiert ist jeweils die Zahl, deren Vielfache entfernt wurden):

$$L_0 = \{2, 3, 4, 5, 6, 7, 8, 9\}$$
$$L_1 = \{\bar{2}, 3, 5, 7, 9\}$$
$$L_2 = \{2, \bar{3}, 5, 7\}$$
$$L_3 = \{2, 3, \bar{5}, 7\}$$
$$L_4 = \{2, 3, 5, \bar{7}\}$$

Es bleiben nur die Primzahlen übrig.

Lösen Sie für die Implementierung des Zahlensiebs folgende Teilprobleme.

a) Schreiben Sie eine Prozedur `from-to`, die zwei natürliche Zahlen f und t akzeptiert und die aufsteigend sortierte Liste der natürlichen Zahlen i zurückgibt, die größer oder gleich f und kleiner oder gleich t sind ($f \leq i \leq t$). [3]

Beispiel: `(check-expect (from-to 2 7) (list 2 3 4 5 6 7))`

b) Implementieren Sie eine Prozedur `filter-multiples`, die eine Liste von natürlichen Zahlen l und eine natürliche Zahl i akzeptiert und aus l alle Vielfachen von i entfernt.

Hinweis: Benutzen Sie die eingebaute Prozedur `modulo`, um den Rest einer ganzzahligen Division zu berechnen. [4]

Beispiel: `(check-expect (filter-multiples (list 3 4 5 6 7 8) 2) (list 3 5 7))`

c) Implementieren Sie eine Prozedur `sieve`. `sieve` akzeptiert eine Liste von natürlichen Zahlen und entfernt daraus alle Zahlen, die Vielfache von anderen Zahlen aus dieser Liste sind. [3]

Hinweis: Gehen Sie davon aus, dass l aufsteigend sortiert ist.

Beispiel: `(check-expect (sieve (list 3 4 5 6 7 8 9)) (list 3 4 5 7))`

d) Implementieren Sie das Sieb des Erathostenes mittels einer Prozedur `primes`. `primes` akzeptiert eine natürliche Zahl n und berechnet die Liste aller Primzahlen von 2 bis n. Gehen Sie davon aus, dass $n > 2$ gilt. [2]

Beispiel: `(check-expect (primes 9) (list 2 3 5 7))`

Abgabe: Programm: `sieve.rkt`

3.2 List And Or Zip [11]

Implementieren Sie die folgenden rekursiven Prozeduren auf Listen.

a) Programmieren Sie eine Prozedur `list-or`, die eine Liste von Booleans akzeptiert und genau dann `#t` zurückliefert, wenn *mindestens eines* der Listenelemente `#t` ist. `list-or` auf der leeren Liste soll `#f` liefern. [4]

b) Programmieren Sie eine Prozedur `list-and`, die eine Liste von Booleans akzeptiert und genau dann `#t` zurückliefert, wenn *alle* Listenelemente `#t` sind. `list-and` auf der leeren Liste soll `#t` liefern. [3]

c) Programmieren Sie eine Prozedur `zip`, die zwei Listen akzeptiert und eine Liste von 2-Tupeln zurückgibt. Das erste Tupel der Ergebnisliste soll das jeweils erste Element der beiden Eingabelisten enthalten, das zweite Tupel das jeweils zweite Element etc. Ist eine der beiden Listen länger als die andere, sollen die überstehenden Elemente der längeren Liste verworfen werden.

Beispiel: `(check-expect (zip (list 1 2) (list 3 4)) (list (make-tuple 1 3) (make-tuple 2 4)))`

Nutzen Sie die folgende Record-Definitionen für parametrisch polymorphe 2-Tupel (a, b): [4]

Abgabe: Programm: `andorzip.rkt`

```
; Parametrisch-polymorphe 2-Tupel fuer zip
(define-record-procedures-parametric
  tuple
  tuple-of
  make-tuple
  tuple?
  (tuple-first tuple-second))

;;;;; EIGENE LOESUNG AB HIER!
```