Prof. Dr. Christoph Bockisch
Fachbereich Mathematik und Informatik
AG Programmiersprachen und -werkzeuge

Philipps Universität Marburg

Übung zur Vorlesung

Deklarative Programmierung: Sommersemester 2022

Nr. 7, Abgabe bis 07.06.2022 23:55 Uhr

Aufgabe 1: Bambis Bester Buddy

0 Punkte

Dies ist eine Autotestat-Aufgabe. Die Daten für diese Aufgabenstellungen finden Sie im ILIAS Ordner für diesen Übungszettel unter dem Namen `Aufgabenstellung-07.1a.zip` – `Aufgabenstellung-07.1c.zip`.

In der Vorlesung haben Sie bereits Listen kennengelernt und gesehen, wie Sie damit arbeiten. In dieser Aufgabe sollen Sie einige nützliche Listenfunktionen implementieren:

Sie dürfen nur die eingebauten Funktionen `empty?`, `cons`, `first`, `rest`, `cons?` und `equal?` benutzen, sowie die eingebaute Konstante `empty` und die eingebauten `check-*` Funktionen. In den Teilaufgaben b und c dürfen Sie zusätzlich auf die eingebaute `append`-Funktion zurückgreifen. Ansonsten dürfen Sie nur selbst definierte Funktionen aufrufen.

(a) `(list-remove e l)` entfernt alle Vorkommen von e aus der Liste l und liefert die resultierende Liste. Benutzen Sie für den Vergleich `equal?`. **0**

(b) `(list-reverse l)` dreht den Inhalt der Liste um. Sie dürfen zusätzlich die eingebaute Funktion append benutzen. **0**

Beispiel:

`(list-reverse (list 1 2 3))` $\xrightarrow{*}$ `(list 3 2 1)`

(c) `(list-flat l)` klopft die Liste l flach. Sie dürfen zusätzlich die eingebaute Funktion append benutzen. **0**

Verwenden Sie als Typ des Funktionsparameters:

```
; [X] Nested-List-of X is one of:
; - X
; - (list-of (Nested-List-of X))
```

Beispiel:

`(list-flat (list (list 1) 2 (list (list 3 4) 5) (list (list empty))))` $\xrightarrow{*}$ `(list 1 2 3 4 5)`

Aufgabe 2: Häkeln 4 Punkte

In dieser Aufgabe soll ein Programm entwickelt werden, das zu einer "Häkelreihe" das Muster als String erzeugt. Eine Häkelreihe soll durch den algebraischen Datentyp *Masche* dargestellt werden. Eine Reihe beginnt immer mit einer *Anfangsschlinge* und wird mit einer *Fortsetzungsschlinge* fortgeführt. Es gibt verschiedene Fortsetzungsschlingen, z.B. die *Luftmasche* oder das *Stäbchen*, die jeweils eine Vorgänger-Masche haben.

(a) Definieren Sie zunächst alle Strukturen, die für die Informations-Repräsentation benötigt werden, inklusive beschreibender Kommentare, sowie alle Summentypen die zu den algebraischen Datentypen gehören. **1**

(b) Implementieren Sie nun eine Funktion `(define (masche->string masche) ...)`. Implementieren Sie hierzu auch jeweils die benötigten Hilfsfunktionen und geben Sie bei jeder Funktion auch deren Signatur an. Die verschiedenen Maschen sollen folgendermaßen durch Zeichenketten dargestellt werden. **1**

- Anfangsschlinge: `"-o"`
- Luftmasche: `"x|"`
- Stäbchen: `"=|"`

Geben Sie außerdem bei jeder Funktion an, ob es sich bei der Implementierung um einen "abstrakten Algorithmus", "Dispatch", oder "konkrete Funktionalität" handelt.

(c) Schreiben Sie einen Aufruf der Funktion `masche->string`, der folgende Ausgabe erzeugt: `"-ox|x|=|x|x|"` **1**

(d) Eine Ihrer (Hilfs-)Funktionen muss rekursiv sein. Argumentieren Sie, warum diese Funktion terminiert. **1**

Aufgabe 3: Natürliche Zahlen 4 Punkte

Verwenden Sie in dieser Aufgabe den aus der Vorlesung bekannten Datentyp `Nat` für die natürlichen Zahlen. In dieser Aufgabe dürfen Sie außer `add1` und `sub1` **keine eingebauten arithmetischen Operatoren** verwenden.

(a) Schreiben Sie eine Funktion `(add-nat n1 n2)` mit der Signatur `Nat Nat -> Nat`, die die Summe der Parameter n1 und n2 zurück gibt. **1**

(b) Schreiben Sie eine Funktion `(mul-nat n1 n2)` mit der Signatur `Nat Nat -> Nat`, die das Produkt der Parameter n1 und n2 zurück gibt. **1**

(c) Schreiben Sie die Funktion `(is-odd? n)` mit der Signatur `Nat -> Boolean`, die zurückgibt, ob der Parameter n ungerade ist. **1**

(d) Schreiben Sie die Funktion `(sum l)` mit der Signatur `(list-of Nat) -> Nat`, die Summe der natürlichen Zahlen berechnet, die in in dem Listen-Parameter l übergeben werden. **1**

Aufgabe 4: *"You can do it."* – Coffee 4 Punkte

(a) Schreiben Sie eine Funktion `evalQuote`, die ein übergebenes Quote auswertet. Die Funktion soll sich dabei auf Listen von Number beschränken und die vier Grundrechenarten (+, -, / und *) beherrschen. Sie können davon ausgehen, dass es sich bei diesen Grundrechenarten um binäre Operationen handelt. **2**

Beispiel: `(evalQuote '(1 2 (+ 2 1)))` $\xrightarrow{*}$ `(list 1 2 3)`

(b) Farben lassen sich als Tripel darstellen, die jeweils den Rot-, Grün- und Blauanteil des Farbwertes enthalten, zum Beispiel: **2**

Farbname	Rot	Grün	Blau
Lachsrosa	250	128	114
Himmelblau	135	206	235

Gemäß der folgenden Formel lässt sich dann zu einer Farbe deren Grauwert berechnen (beachten Sie, dass das Ergebnis hier abgerundet wird):

$$Grauwert = \lfloor 0.299 \cdot Rot + 0.587 \cdot Gruen + 0.114 \cdot Blau \rfloor$$

Schreiben Sie eine Funktion, die mittels Quasiquoting implementiert ist und vier Parameter übernimmt: Den Namen einer Farbe, sowie den Rot-, Grün- und Blauanteil. Der Rückgabewert der Funktion soll eine Liste von drei Strings sein: Name der Farbe, `":"` und der Grauwert. Für Lachsrosa soll also die Liste `(list "Lachsrosa" ":" "162")` erzeugt werden.

```
(check-expect (length (farbe->grau "Lachsrosa" 250 128 114))3)
(check-expect (first (farbe->grau "Lachsrosa" 250 128 114))  "Lachsrosa")
(check-expect (second (farbe->grau "Lachsrosa" 250 128 114)) ":")
(check-expect (third (farbe->grau "Lachsrosa" 250 128 114)) "162")

(define (farbe->grau name r g b) ...
```

Bonusaufgabe 5: Rocket of the Storm 2.0 4 Bonuspunkte (*)

(*) Bonusaufgabe: *Diese Aufgabe ist eine Bonusaufgabe. Das heißt, Sie können hierauf bis zu vier Bonuspunkte bekommen. Neben der erfolgreichen Bearbeitung der Aufgabe setzte das zusätzlich voraus, dass Sie Ihre Lösung Ihrem Tutor oder Ihrer Tutorin vorstellen und erklären.*

Das Vermeiden von Kollisionen mit dem Rand der Szene ist auf Dauer etwas langweilig. In der nächsten Ausbaustufe soll der Spieler die Rakete dazu benutzen, Gegenstände einzusammeln. Und das in verschiedenen Levels. Die erreichten Punkte werden nun nicht mehr mit jedem Tick erhöht, sondern mit jedem eingesammelten Gegenstand.

(a) Die einzusammelnden Gegenstände sollen unterschiedliche Formen haben (mindestens Kreise und Quadrate). Mit jedem Gegenstand ist zudem eine Anzahl an Punkten verknüpft, die der Spieler erhält, wenn er ihn einsammelt. Modellieren Sie einen algebraischen Datentyp, der einen solchen Gegenstand abstrahiert. 1(*)

(b) Ein Level besteht aus einem Namen und einer Liste von einzusammelnden Gegenständen. Modellieren Sie Levels durch eine entsprechende Struktur und passen Sie den World-State an, sodass er eine Liste von Levels führt, die der Spieler noch meistern muss. 1(*)

(c) Bei jedem Tick muss nun überprüft werden, ob die Rakete einen Gegenstand aus dem aktuellen Level berührt. Ist dies der Fall, muss der Gegenstand aus dem Level entfernt werden. Implementieren Sie die nötigen Funktionen. Beachten Sie, dass hierbei Bilder (z.B. die Rakete) als Rechtecke behandeln werden. 1(*)

(d) Passen Sie den Rest des Spiels an: Vor jedem Level, soll zunächst ein Text mit dem Level-Namen angezeigt werden. Nach einem Klick beginnt der Spieler in der Mitte der Szene mit dem aktuellen Level. Achten Sie darauf, die Geschwindigkeit der Rakete zurückzusetzen! Sind alle Gegenstände eingesammelt, rückt der Spieler ins nächste Level (mit vorherigem Screen). Sind alle Level geschafft, wird eine entsprechende Meldung ausgegeben. Das Spiel ist weiterhin verloren, sobald die Rakete den Rand der Szene berührt. Weiter soll auch die Geschwindigkeit – wie bisher – mit jedem Klick zunehmen. 1(*)

Hinweis: Eine Musterlösung zur letzten Version des Spiels finden Sie im ILIAS.