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

Philipps Universität Marburg

Übung zur Vorlesung

Deklarative Programmierung: Sommersemester 2022

Nr. 4, Abgabe bis 10.05.2022 23:55 Uhr

Achtung! Ab diesem Zettel wird gefordert, dass Sie sich bei der Definition von Funktionen immer an das Entwurfsrezept halten. Achten Sie daher darauf, dass Sie sich an die aus der Vorlesung bekannten Schreibweisen und Konventionen halten. Auch wenn in einer Aufgabenstellung nicht extra darauf hingewiesen wird, führt das nicht-Einhalten des Entwurfsrezepts zu Punkt-Abzug.

Aufgabe 1: Datentypen

0 Punkte

Dies ist eine Autotestat-Aufgabe. Die Daten für diese Aufgabenstellungen finden Sie im Archiv mit den Übungsmaterialien für diese Woche. Entpacken Sie entsprechend der Anleitung die Dateien `Aufgabenstellung-04.1a.zip`, `Aufgabenstellung-04.1b.zip` und `Aufgabenstellung-04.1c.zip`.

Wenden Sie das Entwurfsrezept für Funktionen über den angegebenen Arten von Datentypen an. Denken Sie an die korrekte Definition des Datentyps, an ausreichende Tests und wenden Sie die passende Schablone bei der Implementierung der Funktion an. Achten Sie auch auf Groß- und Kleinschreibung (Namen von Datentypen sollten immer groß geschrieben werden, Funktionsnamen immer klein).

(a) **Enumerationstyp:** In unserem Sonnensystem existieren vier innere Planeten: Merkur, Venus, Erde und Mars. Diese benötigen für eine Umdrehung um sich selbst 84456 Minuten (Merkur), 349947 Minuten (Venus), 1436 Minuten (Erde) und 1537 Minuten (Mars). Definieren Sie einen Enumerationstyp “InnererPlanet”, dessen Werte die Zeichenketten mit den Namen der Planeten (in Kleinbuchstaben) sind. Implementieren Sie weiterhin eine Funktion “rotationProErdtage”. Diese nimmt als ersten Parameter den inneren Planeten und als zweiten eine Zahl. Das Ergebnis ist die Anzahl der Umdrehungen, die der innere Planet in der angegebenen Anzahl an Erdtagen vollführt. **0**

(b) **Intervalltyp:** Ein Vergnügungspark berechnet den Eintrittspreis je nach Körpergröße des Besuchers. Ein Besucher bis zu 120cm (d.h. ein Kleinkind) ist frei, für Besucher über 120cm bis zu 140cm (d.h. ein Kind) kostet 12 Euro und ein Besucher über 140cm kostet 15 Euro. Definieren Sie einen Intervalltyp “Besucher” der auf dem Typ Number aufbaut **0**

und diesen in drei Intervalle einteilt. Implementieren Sie außerdem eine Funktion “preis”, die als Parameter einen Besucher entgegennimmt und den Eintrittspreis zurückliefert.

(c) **Summentyp:** Definieren Sie einen Summentypen “RomeOrArabic”, der Zahlen entweder als römische Zahlen (kodiert als Zeichenkette) oder als arabische Zahlen (kodiert als Number) enthält. Schreiben Sie weiterhin eine Funktion “romeOrNumber->String”, welche einen “RomeOrArabic” Wert als Parameter entgegennimmt und eine entsprechende Zeichenkette zurückliefert. Sie können die Funktion “number->string” benutzen. **0**

Aufgabe 2: Arbeiten mit Strukturen — 0 Punkte

Dies ist eine Autotestat-Aufgabe. Die Daten für diese Aufgabenstellungen finden Sie den Dateien `Aufgabenstellung-04.2a.zip` – `Aufgabenstellung-04.2d.zip`.

In der Vorlesung haben Sie die posn-Struktur kennengelernt. Wir möchten nun einige Funktionen auf dieser Struktur implementieren. Achten Sie dabei darauf, das Entwurfsrezept anzuwenden.

Dort wo die untenstehenden Funktionen einen Vektor erwarten, soll auch in der Signatur der Typ Vektor angegeben werden. Definieren Sie den Typ Vektor als Kommentar. Verwenden Sie hier die Struktur Posn wieder. Ein Vektor ist eine Posn-Struktur wobei die beiden Felder jeweils ein Number sind.

(a) `(vec-add v1 v2)`, die die übergebenen Vektoren addiert. **0**

(b) `(vec-sub v1 v2)`, die die übergebenen Vektoren subtrahiert. **0**

(c) `(vec-skal-mult s v)`, die den übergebenen Vektor mit dem Skalar s multipliziert. **0**

(d) `(vec-norm v)`, die den übergebenen Vektor normiert. **0**

Aufgabe 3: Die Liga der außergewöhnlichen Summentypen — 4 Punkte

(a) In dieser Aufgabe sollen Sie **schrittweise** die fünf Schritte des Entwurfsrezepts für Summentypen durchführen. Achten Sie darauf, dass Sie jeden einzelnen Schritt benennen und dokumentieren. Geben Sie einen geeigneten Summentypen an, der die folgenden Bedingungen erfüllt: **4**

Ein Auto verbraucht bei einer Geschwindigkeit von unter durchschnittlich 120 km/h 8 Liter auf 100 km. Befindet sich die Geschwindigkeit zwischen 120 km/h und 160 km/h, so liegt der Verbrauch bei 10 Liter auf 100 km. Ist die Geschwindigkeit allerdings höher als 160 km/h, so steigt der Verbrauch linear an (0.55 Liter pro 10 km/h).

Aufgabe 4: Batchprogramme — 4 Punkte

In dieser Aufgabe sollen Sie ein lauffähiges Programm entwickeln, welches für eine Dateigröße eine lesbare Darstellung mit der Einheit Byte, KB oder MB bestimmt. Passen Sie dazu das zur Verfügung gestellte Programmgerüst (`batch-skel.rkt`) an, das Sie in den Vorgaben auf ILIAS finden. Eine Beschreibung der genauen Aufgabe der Funktion finden Sie in dem Programmgerüst.

(a) Definieren Sie zunächst geeignete Testfälle für die `sizeWithUnit`-Funktion. **1**

(b) Implementieren Sie die `sizeWithUnit`-Funktion **1**

(c) Rufen Sie das Programm mit einem gültigen Wert als Parameter von der Kommandozeile aus auf und fertigen Sie einen Screenshot der Ausgabe an. Die Kommandozeile hierzu sieht folgendermaßen aus: **1**

```
"<path-to-racket>" <ihr-programm.rkt> <kommandozeilen-parameter>
```

Als `<path-to-racket>` geben Sie den Pfad zum Programm “racket” an, genauso wie im AutoAssessment-Werkzeug. Benutzen Sie Anführungszeichen für den Fall, dass der Pfas Leerzeichen enthält.

(d) Übersetzen Sie das Programm in eine **ausführbare** Datei. Sehen Sie sich dazu die Dokumentation zu `raco` (`https://docs.racket-lang.org/raco/exe.html`) an. Rufen Sie nun auch dieses Programm mit einem gültigen Datum auf und fertigen Sie ebenfalls einen Screenshot an. **1**

Hinweis: Gehen Sie bei der Implementierung der sizeWithUnit-Funktion davon aus, dass Sie stets gültige und vergangene Daten erhalten. Halten Sie sich bei der Abgabe der Screenshots an die Abgaberichtlinien und geben Sie die beiden Screenshots in einer einzigen PDF-Datei ab.

Aufgabe 5: Rocket interactive — 4 Punkte

In dieser Aufgabe sollen Sie die Animation der Rakete vom letzten Zettel ein wenig interaktiver gestalten. Bei einem Klick in die Animation, soll sich die Richtung der Rakete umkehren. Passen Sie dazu Ihre Lösung von Aufgabe 3.4 entsprechend an. Wenn Sie die die Aufgabe 4 auf dem letzten Zettel nicht gelöst haben, können Sie den Code in der Datei `rocket-v8.rkt` weiterentwickeln.

(a) Der `WorldState` soll ein einfacher Zähler sein. Geben Sie die entsprechende Datendefinition an. **1**

(b) Definieren Sie die Funktionen on-tick-event, end-of-world und render. Das Programm soll enden, falls die Rakete das obere bzw. untere Ende der Szene erreicht. Jeder Tick soll den WorldState um 1 erhöhen, die render-Funktion zeichnet – wie bisher – die Rakete **1**

abhängig vom übergebenenWorldState. Ist das Programm beendet, soll der Text „GAME OVER“ mittig in der Szene platziert werden.

(c) Definieren Sie einen Mouse-Handler (`on-mouse-event`), welcher den WorldState bei einem Klick so modifiziert, dass die Rakete ihre Richtung ändert. **1**

(d) Verbinden Sie alle Funktionen durch einen geeigneten Aufruf der `big-bang`-Funktion. Achten Sie darauf, dass die Rakete in der Mitte der Szene startet. **1**

Geben Sie zu jeder Funktion Signatur und Aufgabenbeschreibung in einem Kommentar an und definieren Sie geeignete Testfälle.