Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2020/21

Prof. Dr. Michael Rohs, Tim Dünte, M.Sc.

Übungsblatt 3

Alle Übungen (bis auf die erste) müssen in Zweiergruppen bearbeitet werden. Beide Gruppenmitglieder müssen die Lösung der Zweiergruppe einzeln abgeben. Die Namen beider Gruppenmitglieder müssen sowohl in der PDF Abgabe als auch als Kommentar in jeglichen Quelltextabgaben genannt werden. Wenn Sie für Übungsblatt 1 noch keinen Gruppenpartner haben, geben Sie alleine ab und nutzen Sie das erste Tutorium dazu, mit Hilfe des Tutors einen Partner zu finden. Plagiate führen zum Ausschluss von der Veranstaltung.

Abgabe bis Donnerstag den 05.11. um 23:59 Uhr über https://assignments.hci.uni-hannover.de/WiSe2020/Programmieren1. Die Abgabe muss aus einer einzelnen zip-Datei bestehen, die den Quellcode, ein pdf bei Freitextaufgaben und alle weiteren nötigen Dateien (z.B. Eingabedaten oder Makefiles) enthält. Lösen Sie Umlaute in Dateinamen bitte auf.

Lesen Sie das PostFix-Tutorial (https://postfix.hci.uni-hannover.de/postfix-lang.html) weiter bis zum Abschnitt Input and Output (einschließlich).

Aufgabe 1: Skript

Das Skript unter http://hci.uni-hannover.de/files/prog1script-postfix/script.html beschreibt verschiedene Vorgehensweisen bei der Lösung von Programmieraufgaben.

a) Lesen Sie Kapitel 3 (Evaluating Functions Without Side-Effects) und beantworten Sie folgende Frage: Warum können Funktionen mit Seiteneffekt nicht wie Funktionen ohne Seiteneffekte durch algebraische Umformung evaluiert werden? Geben Sie ein Gegenbeispiel.

b) Lesen Sie Kapitel 4 (Recipe for Enumerations) und beantworten Sie folgende Frage: Welche Vorteile hat die cond-Anweisung gegenüber der if-Anweisung?

c) Lesen Sie Kapitel 5 (Recipe for Intervals) und beantworten Sie folgende Frage: Welche Rolle spielen Symbole und Konstanten bei Intervallen?

d) Was war Ihnen beim Lesen der Kapitel 3-5 unklar? Wenn nichts unklar war, welcher Aspekt war für Sie am interessantesten?

Aufgabe 2: Tempomat

Ein Tempomat hat die Aufgabe, ein gegebenes Tempo möglichst genau zu halten. Entwickeln Sie eine Funktion zur Regelung eines Tempomats, welche abhängig vom aktuellen Tempo und dem Zieltempo entweder bremst, nichts tut, beschleunigt oder eine Notbremsung auslöst.

- Für den Fall, dass das aktuelle Tempo oder das Zieltempo < 0 km/h ist, soll in jedem Fall eine Notbremsung ausgelöst werden.
- Für den Fall, dass das aktuelle Tempo größer als 250 km/h ist, soll der Tempomat bremsen.

- Der Tempomat soll bremsen, falls das aktuelle Tempo mehr als 2% größer ist als das Zieltempo.
- Der Tempomat soll beschleunigen, falls das aktuelle Tempo mehr als 2% kleiner ist als das Zieltempo.
- Andernfalls (aktuelles Tempo innerhalb ±2% vom Zieltempo) soll der Tempomat nichts tun.

Führen Sie die im Skript unter Recipe for Intervals beschriebenen Schritte durch. Verwenden Sie die Template-Datei `tempomat-control.pf`. Bearbeiten Sie die mit `todo` markierten Stellen. Geben Sie mindestens 6 sinnvolle Beispiele (Eingaben und erwartete Resultate) in Form von Testfällen an.

Aufgabe 3: Auktionshaus

Ein Prototyp für ein Auktionshaus soll in Postfix implementiert werden (`auction.pf`). Vervollständingen Sie die folgenden Funktionen. Lesen Sie im Skript das Kapitel Recipe for Itemization.

a) Implementieren Sie die Funktion `bid: (auction :Obj, bidder :Str, offer :Num -> :Obj)`, um das Bieten auf eine Auktion zu ermöglichen. Die Funktion bekommt eine Auktion übergeben, die vom Typ Objekt ist, da eine Auktion verschiedene Zustände annehmen kann. Eine Auktion kann vom Typ `:Int` sein, wenn Sie noch nicht begonnen hat. Dann entspricht der Wert dem Startzeitpunkt der Auktion (in Sekunden). Weiter kann sie vom Typ `:Arr` sein, dann enthält das Array das aktuelle Gebot, den aktuellen Bieter und den Endzeitpunkt (in Sekunden). Wenn die Auktion beendet ist so ist die Auktion vom Typ `:Str` und enthält den Namen des Höchstbietenden. Beispiele dazu sind im Code genannt. Ein Gebot soll in folgenden Fällen **nicht** möglich sein:

   - Wenn die Auktion noch nicht begonnen hat oder beendet ist
   - Wenn das neue Gebot kleiner oder gleich dem alten Gebot ist (Ausnahme das erste Gebot darf gleich dem Startgebot sein)
   - Wenn der gleiche Bieter versucht sein Gebot zu erhöhen
   - Wenn der Bieter der leere String ist `""`

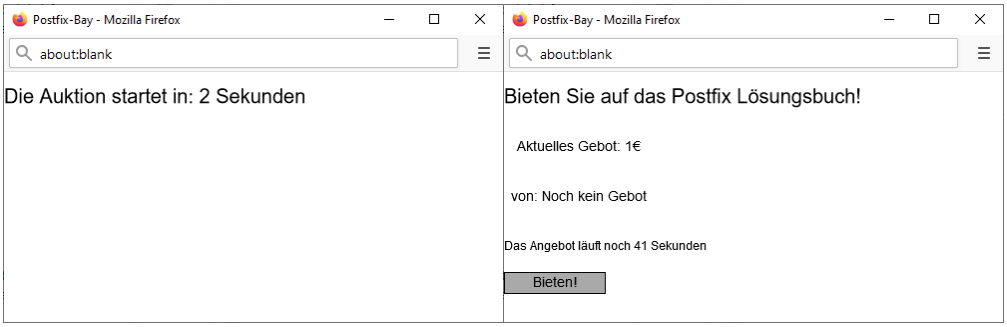
   In den oben genannten Fällen soll das alte Gebot erhalten bleiben. Schauen Sie sich dazu auch die Testfälle an.

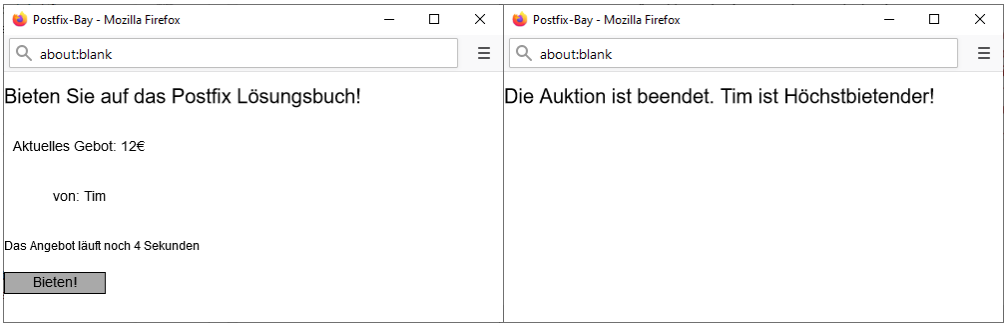
b) Implementieren Sie die Funktion `update-auction: (auction :Obj, time :Int -> :Obj)`. Diese soll den Zustand einer Auktion aktualisieren. Wenn die übergebene Zeit in `time` größer als die Startzeit ist soll die Auktion initialisiert und gestartet werden. Der Endzeitpunkt entspricht der Startzeit plus der Auktionszeit. Schauen Sie sich auch hier die Testfälle an.

c) Wenn alle Testfälle fehlerfrei funktionieren kommentieren Sie die letzte Zeile aus und starten Sie das Programm:

```
# todo: remove comment to animate
#animate
```

Bieten Sie ein paar Mal in dem Sie auf den Button „Bieten!“ klicken und prüfen Sie, ob Ihre Implementation funktioniert. Nachfolgend sind einige Screenshots gezeigt. Ihre Version sollte ähnlich aussehen.
HINWEISE: Die Eingabe des Namens und des Gebots sind leider aktuell nur über die Eingabe der Postfix IDE möglich. Bitte Beenden Sie die Applikation immer über den roten Stopp Button in der IDE.





Aufgabe 4: Schachpositionen

In dieser Aufgabe sollen Funktionen implementiert werden, um verschiedene Darstellungen von Figurpositionen auf einem Schachbrett ineinander zu konvertieren. Verwenden Sie als Template-Datei `chess.pf`.

a) Implementieren Sie die Funktion `pos-to-point: (pos :Str -> :Arr)`, die eine Positionsangabe auf dem Schachbrett der Form "A1" in einen 2D-Punkt der Form [1, 1] konvertiert. Auf dem Schachbrett sind die Spalten von "A" bis "H" und die Zeilen von 1 bis 8 benannt. Ergänzen Sie in der Testfunktion zwei weitere Beispiele. Gehen Sie davon aus, dass die Positionsangabe immer zwei Zeichen lang ist. Dies braucht also nicht überprüft zu werden.

b) Implementieren Sie die Funktion `pos-valid: (p :Str -> :Bool)`, die prüft, ob es sich bei der Eingabe um einge gültige Position auf dem Schachbrett handelt. Auf dem Schachbrett sind die Spalten von "A" bis "H" und die Zeilen von 1 bis 8 benannt. Ergänzen Sie in der Testfunktion zwei weitere Beispiele.

c) Implementieren Sie die Funktion `point-to-pos: (a :Arr -> :Str)`, die einen 2D-Punkt der Form [2, 3] in eine Schachbrettposition der Form "B3" konvertiert. Ergänzen Sie in der Testfunktion zwei weitere Beispiele.

d) Erläutern Sie kurz die Funktionsweise der Funktion knight-next-positions. Erläutern Sie insbesondere die letzte Zeile.