Autoren Marcelo Pedernera, Jan Buehne

Aufgabe 1:

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 Beispiel für eine Funktion, die sich nicht
durch algebraische Umformung evaluieren lässt.

Manche Funktionen sind abhängig von äußeren Umständen und manche geben werde zurück oder legen Sachen in
Dateien ab. Das Ablegen von Dateien oder das auslesen von Daten lässt sich nicht algebraisch darstellen.

BSP: Eine Funktion die einen Wert einliest mit read-int und 2 drauf addiert. Und das Ergebnis wird dann zurück gegeben.

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

Ein cond-Anweisung hat den Vorteil, dass sie mehrere Bedingungen überprüfen kann. Wohingegen eine if-Anweisung nur eine Bedingung überprüft. Ein cond-Anweisung ist also, wie eine verschachtelte if-Anweisung.

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

Die Intervalle werden als Symbole benannt. Die Konstanten sind die Grenzen der Intervalle.
Alternativ:
Die Grenzwerte der Intervalle werden Symbolen zugeordnet. Diese werden am Anfang des Programmes als Konstanten defineiert.
Somit müssen Sie nicht bei jedem Grenzwert-Check neu eingetragen werden und sobald sich die Grenzen ändern kann man die Konstante ändern und
somit haben alle Verweise den selben Wert.

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

Welchen Zweck haben Intervalle als Symbole und was machen Symbole genau?

Aufgabe 2:

a) Welche Bezeichner verweisen auf Funktionen?

trennen, vermengen, Ei-zugeben und backen

b) Was machen die einzelnen Funktionen? Geben sie an, was (welche Datentypen) die einzelnen Funktionen als Parameter auf dem Stack erwarten, was (welche Datentypen) sie als Ergebnis auf den
Stack legen und was sie auf der Standardausgabe ausgeben. Ergänzen Sie in der Funktionssignatur
den Typ der Parameter und ggf. den Typ des Rückgabewertes.

trennen: #nimmt einen Integer und macht daraus einen Array der auf dem Stack landet. Dieser besteht zwei mal aus der gegebenen Zahl.
vermengen: #nimmt zwei Integer und addiert diese und legt einen Integer auf den Stack. Zudem wird "g vermengen" und "..." ausgegeben und eine Sekunde gewartet.
Ei-zugeben: #nimmt einen Integer und einen Array. Zuerst wird der nullte Eintrag aus dem Array ausgelesen und der Wert wird in einen String umgewandelt.
Dann wird der String mit dem String "Eigelb hinzufügen" zusammengefügt und dann geprintet.
Danach wird die Menge der Eigelber addiert mit der bisherigen Menge des Kuchen. Dafür wird die Menge eines Eigelb auf 10g geschätzt.
Danach wird das selbe noch einmal für Eiweiß durchgeführt, jedoch wird hierbei die Menge pro Ei auf 20g geschätzt.
backen: #nimmt drei Integer und gibt nichts zurück.
Zuerst wird der erste Integer in einen String umgewandelt und dann mit den Strings "vorheizen auf" und "°C" zusammengefügt und geprintet.
Danach wird der "vorgeheizt" und "backen" in jeweils eigenen Zeilen geprintet.
Dann wird "..." geprintet und der zweite Integer verwendet um eine bestimmte Zeit da zu pausieren.
Am Ende wird der dritte Integer in einen String umgewandelt und dann mit dem String "g Kuchen gebacken" und dann geprintet.

Verweis auf Datei:2b_postfix_code.pf

c) Ergänzen Sie die Funktionsdefinitionen mit einer aussagekräftigen Parameterbeschreibung("#< ...
>#"). Dazu gehören ein Purpose Statement, die Beschreibung der einzelnen Parameter sowie die
Beschreibung des Rückgabewertes.

Verweis auf Datei: 2c_postfix_code.pf !!!Nicht fertig!!

d) Beschreiben Sie in 1-2 Sätzen, was das Programm nachfolgend nach dem Kommentar #Rezept
macht.

Nach dem Kommentar #Rezept werden die einzelnen Funktionen mit den dazugehörigen Variablen aufgerufen und ausgeführt. Dazu wird die Variable vor der Funktion eingefügt oder Werte liegen von vorherigen Operationen auf dem Stack

e) Was ist Ihre Meinung zu dem Code hinter dem Kommentar #Rezept. Sollten Programme so geschrieben sein, sodass der Programmcode annähernd wie natürliche Sprache klingt? Begründen Sie kurz.

Ich bin der Meinung das ist nicht sinnvoll den Code so zu schreiben, da die Namen der Funktionen nichts darüber aussagen, was diese nun tun.
Dadurch muss man die Funktion selbst betrachten um zu verstehen was genau diese tut. Und richtige Sprache enthält mehr als nur logische Ausdrücke
und zum verstehen von Programmen brauch man nicht das ganze andere Gedöns was Sprache noch so enthält, sondern für programmieren brauchen wir nur kurze kompakte Aussagen oder Namen.

Aufgabe 3:

a) Für die Lesbarkeit von Quelltext ist die Formatierung sehr wichtig, denn Quelltext wird häufiger gelesen, als geschrieben. Gegeben sei folgender unformatierter Quelltext:

```
f: (i) { "called f" println i 0 < { i -1 * } { i 2 * } if }
fun -3 f f
```

Formatieren Sie diesen Quelltext nach folgenden Regeln:
...

Verweis auf Datei: 3a_postfix_code.pf

b) Erklären Sie das Verhalten der Funktion f möglichst kurz und prägnant. Welcher Wert liegt nach Ausführung des gegebenen Codes auf dem Stack?

Function:
Wenn negativ, dann mach positiv.
Wenn positiv, dann verdopple.

-3 -> 3 -> 6

6 liegt am Ende auf dem Stack

c) Das Heron-Verfahren wurde in der iterativen Variante in PostFix implementiert (siehe: https://de.wikipedia.org/wiki/Heron-Verfahren). Die genügend gute Näherung gilt: Absolutwert(Näherung * Näherung - Zahl) < 0.01. Leider haben sich einige Fehler in die Funktion geschlichen. Formatieren Sie zuerst die Funktion, sodass sie gut lesbar ist. Finden Sie dann die Fehler und korrigieren Sie diese. Beschreiben Sie die Fehler.

Verweis auf Datei: 3c_postfix_code.pf

Aufgabe 4:

Implementieren Sie ein Programm zum Zahlenraten in PostFix. Der Computer soll eine Zahl zwischen 0
und 99 würfeln, die der Spieler in möglichst wenigen Zügen herausfinden soll. Dabei gibt der Computer
jeweils nur den Hinweis, ob die vom Spieler eingegebene Zahl größer oder kleiner als die gesuchte Zahl
ist.

a) Implementieren Sie das Programm zum Zahlenraten wie beschrieben. Achten Sie darauf, dass der
Stack nach erfolgreicher Beendigung des Spiels leer ist (ohne die Verwendung von clear).

Verweis auf Datei: 4a_postfix_code.pf

b) Spielen Sie das Spiel selbst fünf Mal. Notieren Sie jeweils wie viele Züge Sie gebraucht haben
um die Zahl zu erraten. Schreiben Sie ebenfalls auf wie viele Züge Sie in Durchschnitt gebraucht
haben.

1. Versuch: Benötigte Züge 5
2. Versuch: Benötigte Züge 6
3. Versuch: Benötigte Züge 8
4. Versuch: Benötigte Züge 7
5. Versuch: Benötigte Züge 3
Durchschnitt: 5,8 Züge

c) Beschreiben Sie eine Strategie um die Zahl zu erraten, mit der möglichst wenige Züge benötigt
werden.

Der schnellste Weg zuverlässig ist immer die hälfte zu nehmen. Also man startet mit 50 bei zu klein würde man 75 nehmen bei dann zu groß würde man dann 62 usw.