Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2022/23

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

Präsenzübung 2

Diese Aufgaben sind zur Lösung während der einstündigen Präsenzübung gedacht. Sie können die Aufgaben auf einem mitgebrachten Laptop oder auf Papier lösen.

Aufgabe 1: Executable Arrays und Funktionen

Mit Executable Arrays, notiert als {…}, lassen sich Code-Blöcke schreiben, die nicht sofort ausgeführt werden, sondern erst nach Auswahl durch eine if-Anweisung, in einer Schleife oder nach Referenzierung im Dictionary. Einem Executable Array kann ein eigenes Dictionary zugeordnet werden, in dem lokale Variablen bei der Ausführung des Executable Arrays untergebracht werden. Executable Arrays mit eigenem Dictionary werden in PostFix als Funktionen bezeichnet. Diese können eine Parameterliste, notiert als (…), haben.

a) Was ist der Effekt dieses Programms? `{ -1 * } g! 3 g`
b) Was ist der Effekt dieses Programms? `{ -1 * } g! true g`
c) Ist es möglich, das Programm aus (a) zu schreiben als…? `[ -1 * ] g! 3 g`
d) Was ist der Effekt dieses Programms? Wie funktioniert h im Detail?
```
{ dup 0 < {-1 *} if } h! 1.2 h -3.4 h
```
e) Was ist der Effekt dieses Programms?
```
w: (x y) { x y div x y mod } fun 10 3 w
```
f) Gibt es Eingaben, für die das Resultat des Programms aus (e) nicht definiert ist?

Aufgabe 2: Executable Arrays mit lokalem Dictionary

In dieser Aufgabe soll der Unterschied zwischen Executable Arrays und Funktionen beleuchtet werden, sowie die Rolle von lokalen Variablen. Gegeben seien drei Programme:

1)
```
1 x! f: { 1 + x! x print } ! 1 f x println
```
2)
```
1 x! g: { 1 + x! x print } lam ! 1 g x println
```
3)
```
1 x! h: { 1 + x! x print } fun 1 h x println
```

a) Was ist der Unterschied zwischen Programm (1) und Programm (2)?
b) Was ist der Unterschied zwischen Programm (2) und Programm (3)?

Aufgabe 3: Funktionen mit Parametern und Testfälle

Implementieren Sie eine Funktion, die zwei Zeichenketten (Strings, `:Str`) als Eingaben nimmt und eine ganze Zahl (Integer, `:Int`) als Ergebnis zurückgibt. Die Datentypen sollen in der Parameterliste explizit angegeben werden. Die Funktion soll 0 zurückgeben, wenn die beiden Zeichenketten gleich lang sind (`length`-Operator), -1 zurückgeben, wenn das erste Argument kürzer ist als das zweite und 1 zurückgeben, wenn das erste Argument länger ist als das zweite.

Das Programm soll mit mindestens drei Testfällen (`actual expected test=`) überprüft werden. Es soll ein Dokumentationsstring (Purpose Statement) formuliert werden und als Kommentar (`#...`) vor die Funktion geschrieben werden. Verwenden Sie die Vorgehensweise aus der Vorlesung:

1. **Problem Statement:** Durch den Aufgabentext gegeben. Diskutieren Sie, ob die Problembeschreibung eindeutig ist oder nicht. (Nur mündlich, nicht aufschreiben.)
2. **Data Definition:** Diskutieren Sie, welche Daten im Programm repräsentiert werden müssen. (Nur mündlich, nicht aufschreiben.)
3. **Function Name and Parameter List:** Finden Sie einen geeigneten und aussagekräftigen Namen für die Funktion und eine geeignete Parameterliste.
4. **Function Stub and Purpose Statement:** Schreiben Sie einen Funktionsrumpf, der zunächst nur aus einem beliebigen Wert aus dem Wertebereich der Funktion besteht.
5. **Examples with Expected Results:** Überlegen Sie sich einige Beispiele für Werte, die der Funktion übergeben werden könnten und was Sie als Ergebnis erwarten.
6. **Implementation:** Implementieren Sie die Funktion. Überlegen Sie, ob verschiedene Fälle unterschieden werden müssen.
7. **Test and Revision:** Prüfen Sie Ihr Programm an Hand der zuvor aufgeschriebenen Beispiele.

Aufgabe 4 (optional): noch ein Zahlenfolge

Schreiben Sie ein Programm, das bei Eingabe einer Zahl n ($n \geq 0$) eine Folge von n bestimmten Zahlen ausgibt. Hier sind einige Beispiele:

- Eingabe n = 3, Ausgabe: 1, 2, 6
- Eingabe n = 5, Ausgabe: 1, 2, 6, 24, 120
- Eingabe n = 10, Ausgabe: 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800

Verwenden Sie eine Schleife. Zur Erinnerung: Die PostFix-Syntax für Schleifen lautet:

```
{  …  Abbruchbedingung breakif  …  } loop
```

PostFix Entwicklungsumgebung:

https://postfix.hci.uni-hannover.de