Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2020/21

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

Präsenzübung 1

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

Aufgabe 1: PostFix-Notation

Aus der Schule ist die Infix-Notation bekannt. Ein Term in Infix-Notation ist z.B. `3 * (7 – 5)`. In der Programmiersprache Lisp wird Präfix-Notation verwendet. Die Operatoren werden dabei vor den Operanden geschrieben und der obige Ausdruck wird zu `(* 3 (- 7 5))`. Postfix-Notation wird von Sprachen wie Forth und PostScript verwendet. Der Beispielausdruck lautet in Postfix-Notation `3 7 5 - *`. Auch in Kochrezepten wird häufig Postfix-Notation verwendet: „Eier und Mehl verrühren“ (Operanden: Eier, Mehl; Operator: verrühren).

a) Diskutieren Sie die Vor- und Nachteile der PostFix-Notation im Vergleich zur üblichen, aus der Schule bekannten Infix-Notation.

b) Wieso sind in Infix-Notation Klammern notwendig, in Postfix-Notation aber nicht?

c) Wieso ist ein Stapel (Stack) notwendig, um einen Term in Postfix-Notation auszuwerten?

Zu a): Vorteile: keine Klammern notwendig, jeder im Programm auftretende Operator kann sofort angewendet werden, weil die Operanden bereits auf dem Stack liegen; Nachteile: ungewohnte Schreibweise, Operator unter Umständen in der Notation weit entfernt von den Operanden (Beispiel oben: 3 ist weit weg von *).

Zu b): Weil Operatoren verschiedene Ränge haben. Aus der Schule kennt man Punkt-vor-Strich-Rechnung. Wenn Operatoren in Infix-Termen in einer anderen Reihenfolge angewandt werden sollen, sind Klammern notwendig, wie im obigen Beispiel.

Zu c): In einem größeren Term verarbeitet jeder Operator nur einen Teil der Operanden. Es werden jeweils Zwischenergebnisse erzeugt, die gespeichert werden müssen, bevor sie vom nächsten Operator weiterverarbeitet werden.

Aufgabe 2: Übungspunkte

Schreiben Sie ein Programm, das ausrechnet, ob Sie den Übungsteil bestanden haben oder nicht und ob Sie den Bonus erworben haben. Es gibt 12 Übungen. Zum Bestehen muss jede Übung mit mindestens einem Punkt bewertet worden sein. Für den Bonus sind mindestens 21 Punkte notwendig. Die Eingabe ist die Anzahl der Übungen mit zwei Punkten (n2) und die Anzahl Übungen mit einem Punkt. Das Programm soll ausgeben, ob bestanden wurde (true/false) und ob der Bonus erlangt wurde (true ´/false). Hier der Quelltext mit Lücke:

```
read-int n2! # Anzahl Übungen mit 2 Punkten, 0 <= n2 <= 12
```

```
read-int n1! # Anzahl Übungen mit 1 Punkt, 0 <= n1 <= 12
12 n2 - n1 - n0! # Rest ist Anzahl Übungen mit 0 Punkten

false bestanden!
false bonus!

<LÜCKE>

bestanden println
bonus println
```

a) Füllen Sie die Lücke so, dass `bestanden` und `bonus` richtig gesetzt werden. Verwenden Sie if-Anweisungen. Zur Erinnerung: Die PostFix-Syntax lautet:
`Bedingung {Then-Teil} {Else-Teil} if`

b) Füllen Sie die Lücke. Verwenden Sie diesmal keine if-Anweisungen.

zu a):

```
n0 0 = {
    true bestanden!
    n2 2 * n1 + punkte!
    punkte 21 >= {
        true bonus!
    } {
        false bonus!
    } if
} if
bestanden println
bonus println
```

zu b):

```
n0 0 = bestanden!
n2 2 * n1 + 21 > bonus!
```

Aufgabe 3: 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, -4, 9
- Eingabe n = 4, Ausgabe: 1, -4, 9, -16
- Eingabe n = 5, Ausgabe: 1, -4, 9, -16, 25

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

`{… Abbruchbedingung breakif … } loop`

```
read-int n!
1 i!
```

```
1 s!
{
    i n > breakif
    i i * s * print
    i n < { ", " print } if
    s -1 * s!
    i 1 + i!
} loop
```