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 3

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

Aufgabe 1: Logische Operatoren und Sonderfälle

Gegeben seien folgende Funktionen:

```
#<
Checks if a is a two-element array whose first element is
smaller than its second element.
@param  array of elements that can be compared with
        the < operator
@return true if a is a two-element array whose first element
        is smaller than its second element, false otherwise
>#
f: (a :Arr -> :Bool) {
    a length 2 =
    a 0 get  a 1 get  <
    and
} fun

max_elem: (a :Arr -> :Obj) {
    a 0 get m!
    a { x! m x < {x m!} if } for
    m
} fun

[1 2 4 3] max_elem 4 test=
["hello" "world" "abc"] max_elem "world" test=
```

a) Beschreiben Sie ein Problem in der vorliegenden Implementierung der Funktion f. Ändern Sie die Implementierung so, dass das Problem gelöst wird.

b) Schreiben Sie einen Dokumentationsstring für `max_elem` als Kommentar (`#< Purpose... @param... @return... >#`) vor die Funktion. Spezifizieren Sie erlaubte Argumentwerte mit einer geeigneten Beschreibung nach `@param` und `@return`.

c) Welches prinzipielle Problem taucht bei der Implementierung von max_elem auf und warum lässt es sich nicht einfach beheben? Wie kann man die Funktion dennoch verbessern?

Aufgabe 2: Stack-Operationen

a) Implementieren Sie das executable array `count-spaces: {…} !`, das die Leerzeichen in einer Zeichenkette zählt. Verwenden Sie zur Implementierung keine lokalen Variablen und kein Dictionary, sondern nur Operationen auf dem Stack. Benötigt werden: =, +, `swap`, `if` und `for`. Hinweis: `for` kann über die Zeichen einer Zeichenkette iterieren, z.B. `"abc" {print} for`. Hier sind einige Testbeispiele für `count-spaces`:

```
"ab" count-spaces 0 test=
"a b" count-spaces 1 test=
"a b " count-spaces 2 test=
"a b  " count-spaces 3 test=
```

b) Implementieren Sie das executable array `join: {…} !`, das ein Array von Strings zu einem einzelnen String zusammenfügt. Dabei sollen die Array-Elemente durch Kommata getrennt werden. Verwenden Sie zur Implementierung keine lokalen Variablen und kein Dictionary, sondern nur Operationen auf dem Stack. Benötigt werden: >, +, `swap`, `if` und `fori`. Hinweis: fori kann über die Zeichen einer Zeichenkette iterieren und liefert mit jedem Durchlauf den Index und das Zeichen. Hier sind einige Testbeispiele für `join`:

```
["hello"] join "hello" test=
["hello", "world"] join "hello, world" test=
["hello", "good", "bye"] join "hello, good, bye" test=
```

Aufgabe 3: Enumerationen und Mehrfachauswahl

Beim Kartenspiel „Skat" haben die Farben Karo, Herz, Pik und Kreuz die Grundwerte 9, 10, 11 und 12. Schreiben Sie eine Funktion, die für eine gegebene Farbe den Grundwert zurückgibt. Die Farben sollen als Symbole (`:Sym`) spezifiziert werden. Wenn eine ungültige Farbe an die Funktion übergeben wurde, soll eine Fehlerausgabe (mit `"<Fehlerbeschreibung>" err`) erfolgen.

Das Programm soll mit geeigneten Testfällen (`actual expected test=`) überprüft werden. Es soll ein Dokumentationsstring (Purpose Statement) formuliert werden und als Kommentar (`#< Purpose… @param… @return… >#`) 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): Matrixoperationen

Gegeben seien Matrizen der Form:

```
[ [-1, 2, 3, 4],
  [5, 6, 2, 8],
  [9, 10, -1, 5],
  [3, 1, 5, 11],
] matrix!
```

a) Schreiben Sie eine Funktion `column: (a :Arr, i :Int -> :Arr)`, die die i-te Spalte des Arrays zurückgibt. Beispiele für die obige Matrix:

```
matrix 0 column [-1, 5, 9, 3] test=
matrix 2 column [3, 2 -1 5] test=
```

b) Schreiben Sie eine Funktion `vec-minus: (v :Arr, w :Arr -> :Arr)`, die zwei Vektoren (gleicher Länge) elementweise voneinander subtrahiert. . Beispiele für die obige Matrix:

```
matrix 0 get  matrix 1 get  vec-minus [-6 -4 1 -4] test=
matrix 0 column  matrix 1 column  vec-minus [-3 -1 -1 2] test=
```

c) Schreiben Sie eine Funktion `vec-dot: (v :Arr, w :Arr -> :Num)`, die das Skalarprodukt zweier Vektoren (gleicher Länge) berechnet. Beispiele für die obige Matrix:

```
matrix 0 get  matrix 1 get  vec-dot 45 test=
matrix 0 column  matrix 1 column  vec-dot 121 test=
```

d) Schreiben Sie eine Funktion `transpose: (a :Arr -> :Arr)`, die die gegebene Matrix transponiert (i-te Zeile wird zur i-ten Spalte). Beispiel für die obige Matrix:

```
matrix transpose [ [ -1 5 9 3 ]
                   [ 2 6 10 1 ]
                   [ 3 2 -1 5 ]
                   [ 4 8 5 11 ] ] test=
```