Prof. Dr. Ralf Hinze
Sebastian Schloßer, M.Sc.
Markus Heinrich, M.Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Programmiersprachen

Übungsblatt 10: Grundlagen der Programmierung (WS 2020/21)

Ausgabe: 15. Januar 2021
Abgabe: 22. Januar 2021, 15 Uhr

Aufgabe 1 Arrays und Zustand (10 Punkte)

Motivation: In dieser Aufgabe sollen Sie Arrays und Kontrollstrukturen (Schleifen) einüben. Sie können sich an den Vorlesungsfolien 403 bis 426 und 835 bis 851 sowie an den Kapiteln 4.4 und 7.3 im Skript orientieren.

Schreiben Sie Ihre Lösungen in die Datei `Arrays.fs` *aus der Vorlage* `Aufgabe-10-1.zip`.

In den folgenden Teilaufgaben betrachten wir Funktionen, die auf Arrays arbeiten. Es steht Ihnen frei die Funktionen rekursiv oder imperativ zu implementieren.

Hinweis: Beachten Sie, dass Arrays mit nichtnegativen Ganzzahlen vom Typ `Int` *indiziert werden. Der Typ der natürlichen Zahlen* `Nat` *wird als Index leider nicht unterstützt.*

Hinweis: Es gibt mehrere Möglichkeiten ein Array mit Hilfe von Schleifen zu durchlaufen. Einerseits kann mit **`for`** *oder* **`while`** *und einer Zählvariable der Zugriff auf die Arrayelemente direkt über deren „Hausnummer“ erfolgen. Andererseits kann für ein Array* `ar` *mit* **`for`** `x` **`in`** `ar` *auch direkt über die Arrayelemente selbst iteriert werden. Beispiele dazu finden Sie im Skript auf den Seiten 408 und 409.*

Hinweis: Sie können Funktionen aus vorherigen Teilaufgaben verwenden, wenn Sie möchten. Verwenden Sie in Ihrer Lösung ***keine Bibliotheksfunktionen****. Davon ausgenommen sind das* `Length`*-Attribut von Listen und Arrays bzw. die Funktionen* `List.length` *und* `Array.length`*. Konvertieren Sie in Ihrer Lösung nicht zwischen Arrays und Listen hin und her.*

a) Schreiben Sie eine Funktion `swap<'a>: Array<'a> -> Int * Int -> Unit`, die ein Array sowie zwei Indizes nimmt und die Elemente an den Positionen der Indizes vertauscht.

b) Schreiben Sie eine Funktion `bubblesort<'a` **`when`** `'a: comparison>: Array<'a> -> Unit`, die ein Array nimmt und dieses in-place sortiert (also ohne dabei ein neues Array zu konstruieren). Implementieren Sie den Bubblesort[1] Algorithmus.

c) Schreiben Sie eine Funktion `reverse<'a>: Array<'a> -> Unit`, welche das übergebene Array in-place spiegelt.

d) Schreiben Sie eine Funktion `same<'a` **`when`** `'a: equality>: List<'a> -> Array'<a> -> Bool`, die eine Liste sowie ein Array nimmt und zurückgibt, ob die Elemente an korrespondierenden Stellen in der Liste und im Array gleich sind. Wenn die Liste und das Array unterschiedlich lang sind, ist das Ergebnis der `same` Funktion **`false`**.

[1] s. z.B. https://de.wikipedia.org/wiki/Bubblesort#Algorithmus

Aufgabe 2 Ausnahmen (8 Punkte)

Motivation: In dieser Aufgabe sollen Sie Ausnahmen einüben. Sie können sich an den Vorlesungsfolien 852 bis 898 sowie am Skript Kapitel 7.4 orientieren.

Unter Berücksichtigung dieser Typ- und Ausnahmedefinitionen

```
type A = | A1 | A2 of Nat

exception E
exception F of String
exception G of Nat
```

betrachten wir den folgenden Ausdruck. Dabei ist `f` eine Funktion vom Typ `Unit -> A`.

```
try
    match f() with
    | A1   -> raise (F "Harry")
    | A2 x -> x
with
| E   -> 4711N
| F s -> if s = "Harry" then raise E else raise (G 7N)
```

Bestimmen Sie für die folgenden vier Implementierungen der Funktion `f` jeweils, zu welchem Wert obiger Ausdruck auswertet. Kennzeichnen Sie geworfene Ausnahmen dabei wie in der Vorlesung eingeführt mit einem Kästchen, die durch `raise (F "Harry")` geworfene Ausnahme also durch `F "Harry"`.

a) `let f() = raise (G 815N)`

b) `let f() = raise E`

c) `let f() = A1`

d) `let f() = raise (F "Lista")`

Aufgabe 3 Kontrollstrukturen und Ausnahmen (12 Punkte)

Motivation: In dieser Aufgabe sollen Sie dasselbe algorithmische Problem aus verschiedenen Blickrichtungen betrachten, um sich mit den Unterschieden der funktionalen und der imperativen Programmierung vertraut zu machen. Sie können sich an den Vorlesungsfolien 835 bis 898 sowie an den Kapiteln 7.3 und 7.4 im Skript orientieren.

Schreiben Sie Ihre Lösungen in die Datei `Find.fs` *aus der Vorlage* `Aufgabe-10-3.zip`.

In den folgenden Teilaufgaben sollen Sie Funktionen schreiben, die das letzte Element einer Liste zurückgeben, für welches ein vorgegebenes Prädikat zu **`true`** auswertet. In zwei der Teilaufgaben verwenden wir dabei die folgende Ausnahme:

```
exception NotFound
```

Hinweis: Da anhand derselben Problemstellung verschiedene Konzepte eingeübt werden sollen, ist es naheliegend, dass es ***nicht erlaubt ist die Funktionen der einzelnen Teilaufgaben gegenseitig aufzurufen****. Verwenden Sie in Ihrer Lösung außerdem* ***keine Bibliotheksfunktionen****. Davon ausgenommen sind das* `Length` *Attribut von Listen, bzw.* `List.length`*, sofern Sie diese verwenden möchten.*

a) Schreiben Sie eine Funktion `tryFindLast<'a>: ('a -> Bool) -> List<'a> -> Option<'a>`, welche ein Prädikat `pred` sowie eine Liste `xs` nimmt und das letzte Element der Liste zurückgibt, für das `pred` zu **`true`** auswertet. Wenn es in der gesamten Liste kein solches Element gibt, soll `None` zurückgegeben werden. Schreiben Sie eine **rekursive Funktion**, verwenden Sie **keine Kontrollstrukturen (Schleifen) oder Ausnahmen**.

b) Schreiben Sie eine Funktion `findLast<'a>: ('a -> Bool) -> List<'a> -> 'a`, welche ein Prädikat `pred` sowie eine Liste `xs` nimmt und das letzte Element der Liste zurückgibt, für das `pred` zu **`true`** auswertet. Wenn es in der gesamten Liste kein solches Element gibt, soll die **Ausnahme** `NotFound` geworfen werden. Schreiben Sie eine **rekursive Funktion**, verwenden Sie **keine Kontrollstrukturen**.

c) Schreiben Sie eine Funktion `tryFindLast2<'a>: ('a -> Bool) -> List<'a> -> Option<'a>`, welche ein Prädikat `pred` sowie eine Liste `xs` nimmt und das letzte Element der Liste zurückgibt, für das `pred` zu **`true`** auswertet. Wenn es in der gesamten Liste kein solches Element gibt, soll `None` zurückgegeben werden. Schreiben Sie die Funktion imperativ mit Hilfe von **Kontrollstrukturen**, verwenden Sie **keine rekursiven Funktionen oder Ausnahmen**.

d) Schreiben Sie eine Funktion `findLast2<'a>: ('a -> Bool) -> List<'a> -> 'a`, welche ein Prädikat `pred` sowie eine Liste `xs` nimmt und das letzte Element der Liste zurückgibt, für das `pred` zu **`true`** auswertet. Wenn es in der gesamten Liste kein solches Element gibt, soll die **Ausnahme** `NotFound` geworfen werden. Schreiben Sie die Funktion imperativ mit Hilfe von **Kontrollstrukturen**, verwenden Sie **keine rekursiven Funktionen**.