Prof. Dr. Ralf Hinze
Cass Alexandru, M.Sc.
Alexander Dinges, M.Sc.
Felix Winkler, M.Sc.

RPTU
Fachbereich Informatik
AG Programmiersprachen

Übungsblatt 11: Konzepte der Programmierung (WS 2025/26)

Ausgabe: 20. Januar 2026
Abgabe: 27./28./29. Januar 2026, siehe Homepage

Aufgabe 1 Kontrollstrukturen und Ausnahmen (Präsenzaufgabe)

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 836 bis 899 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-11-1.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**.

Aufgabe 2 Arrays (Präsenzaufgabe)

Motivation: In dieser Aufgabe sollen Sie den Umgang mit Arrays einüben. Sie können sich an den Vorlesungsfolien 404 bis 427 und 836 bis 852 sowie an den Kapiteln 4.4 und 7.3 im Skript orientieren.

Schreiben Sie Ihre Lösungen in die Datei `ArrayMap.fs` *aus der Vorlage* `Aufgabe-11-2.zip`.

a) Schreiben Sie eine Funktion `map<'a, 'b>: ('a -> 'b) -> Array<'a> -> Array<'b>`, welche eine Funktion `f` sowie ein Array `ar` nimmt und ein *neues* Array zurückgibt, welches die Anwendung von `f` auf jedes Element von `ar` enthält.

b) Schreiben Sie eine Funktion `inplaceMap<'a>: ('a -> 'a) -> Array<'a> -> Unit`, welche eine Funktion `f` sowie ein Array `ar` nimmt und das Array `ar` in-place verändert, sodass jedes Element von `ar` durch die Anwendung von `f` auf dieses Element ersetzt wird. Warum kann `f` nicht den Typ `'a -> 'b` haben?

Aufgabe 3 Ausnahmen (Einreichaufgabe, 6 Punkte)

Motivation: In dieser Aufgabe sollen Sie Ausnahmen einüben. Sie können sich an den Vorlesungsfolien 853 bis 899 sowie am Skript Kapitel 7.4 orientieren.

Unter Berücksichtigung dieser Typ- und Ausnahmedefinitionen

```
type A = | A1 | A2 | A3 of String | A4 of Bool

exception E
exception E1 of Nat
exception E2 of A
```

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

```
try
    match f() with
    | A1   -> 22N + raise (E2 A2)
    | A2   -> raise E
    | A3 x -> raise (E1 4711N)
    | A4 x when x -> 97N
    | A4 x -> raise (E2 A1)
with
| E    -> 4711N
| E1 n -> if n = 4711N then 50N else raise (E1 815N)
| E2 s -> match s with
           | A1 -> 1N
           | A2 -> 2N
           | A3 x -> 3N
           | A4 x -> raise (E2 (A4 (not x)))
```

Bestimmen Sie für die folgenden 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 (E1 4711N)` geworfene Ausnahme also durch `E1 4711N`.

a) `let f() = A1`

b) `let f() = A4 false`

c) `let f() = raise (E2 (A4 true))`

Aufgabe 4 Arrays und Zustand (Einreichaufgabe, 8 Punkte)

Motivation: In dieser Aufgabe sollen Sie Arrays und Kontrollstrukturen (Schleifen) einüben. Sie können sich an den Vorlesungsfolien 404 bis 427 und 836 bis 852 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-11-4.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, außer wenn explizit gefordert.*

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 `insertionsort<'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 Insertionsort[1] Algorithmus.

c) Schreiben Sie eine Funktion `rotate<'a>: Array<'a> -> Unit`, welche das übergebene Array in-place rotiert, also das erste Element an die zweite Stelle schiebt, das zweite an die dritte, usw. Das letzte Element wird an die erste Stelle geschoben.

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`**. Konvertieren Sie dazu das Array mithilfe einer Listenbeschreibung in eine Liste.

[1] s. z.B. https://en.wikipedia.org/wiki/Insertion_sort#Algorithm