

Übungen zu Einführung in die Informatik I

Aufgabe 13 **Listen in OCaml**

In dieser Aufgabe werden wir die Datenstruktur der Liste selbst definieren.

- a) Definieren Sie rekursiv einen neuen Typ `myList` der entweder leer (`Empty`) sein kann oder ein Element (`Elem`) das aus einem Wert 'a' und einer Liste besteht.
- b) Definieren Sie eine Funktion `prepend`, welche an eine vorhandene Liste einen Wert vorne anfügt. Testen Sie ihre neue Datenstruktur indem Sie eine Liste mit den Zeichen a, h, a aufbauen.
- c) Schreiben Sie eine (rekursive) Funktion `append`, welche an eine vorhandene Liste einen Wert hinten anhängt. Testen Sie Ihre Funktion indem Sie das Zeichen ! an die Liste der letzten Teilaufgabe anhängen.
- d) Schreiben Sie eine (rekursive) Funktion `isElement`, welche für eine Liste und einen Wert testet, ob der Wert in der Liste enthalten ist. Testen Sie Ihre neudefinierte Funktion für die Liste aus den vorherigen Teilaufgaben und die Zeichen x, h und a.
- e) Schreiben Sie eine (rekursive) Funktion `insert`, welche einen Wert in einer Liste **sortiert** einfügt. Die kleinsten Elemente sollen dabei am Ende der Liste stehen. Testen Sie Ihre Funktion mit den Werten 23, 12, 42, 24, 1, 23.

Aufgabe 14 **Schach in OCaml**

Schach (v. persisch: Schah: König; stehende Metapher: das Königliche Spiel) ist ein strategisches Brettspiel für zwei Spieler, bei dem der Zufall keine Rolle spielt (außer beim Losen um die Farbe, das heißt um den ersten Zug), sondern lediglich das Können der Spieler über den Spielausgang entscheidet.

- a) Definieren Sie einen neuen Typ `piece`, der die möglichen Figuren des Schachspiels enthält.
- b) Definieren Sie einen neuen Typ `color`, der die Farben der Schachfiguren enthält.
- c) Definieren Sie einen neuen Typ `tile`, der den Inhalt (eine Figur, oder `Empty`) eines Schachfeldes enthält.
- d) Ein Schachbrett ist eine zweidimensionale Fläche mit Spalten von a – h und Reihen von 1 – 8. Wenn man die einzelnen Reihen hintereinanderhängt kann man das Schachbrett auch als Liste von einzelnen Feldern verstehen. Definieren Sie einen neuen Typ `tileList` mit Hilfe einer selbstdefinierten Liste.
- e) Kreieren Sie das Schachbrett mit Hilfe einer Initialisierungsfunktion `initBoard`. Beachten Sie hierbei eine linearisierte Form des Schachbretts zu verwenden.
- f) Zur Strukturierung großer Programmsysteme bietet OCaml Strukturen an:

```
module Calendar = struct
  type month = January | February | March | April | Mai | June |
    July | August | September | Oktober | November | December
end;;
```

Kapseln Sie Ihre Arbeit in dem Modul Chess.

g) Testen Sie Ihr Modul geeignet.

Aufgabe 15 Das Prinzip der Induktion

In dieser Aufgabe lernen Sie die Induktion kennen, ein mathematisches Hilfsmittel, das im weiteren Verlauf der Übungen noch häufig verwendet werden wird.

Es sei für $x \in \mathbb{N}$ die boolesche Funktion $\text{gerade}(x)$, die ermittelt, ob x durch 2 teilbar ist oder nicht, wie folgt spezifiziert:

$$\text{gerade}(x) = (\text{mod}(x, 2) \stackrel{?}{=} 0)$$

(Hinweis: Die Funktion $\text{mod}(x, 2)$ berechnet den Rest bei ganzzahliger Division von x durch 2.) Darüber hinaus sei die Funktion $\text{gerade}'(x)$ durch

$$\text{gerade}'(x) = \begin{cases} \text{True}, & \text{falls } x = 0 \\ \neg(\text{gerade}'(x-1)), & \text{falls } x \in \mathbb{N} \setminus \{0\} \end{cases}$$

gegeben.

Aufgabenstellung: Zeigen Sie mit Hilfe der Induktion, dass die beiden Funktionen $\text{gerade}(x)$ und $\text{gerade}'(x)$ äquivalent sind.

Aufgabe 16 (H) Schach in OCaml: Setzen von Spielsteinen (7 + 3 + 0 = 10 Punkte)

Verwenden Sie das Modul Chess, das Sie bereits in der Übung entwickelt haben.

- a) Implementieren Sie eine Funktion `setTile tile board column row`, die die Figur `tile` in die Reihe `row` und die Spalte `column` setzt. Der Rückgabewert ist das Schachbrett `board`.
- b) Stellen Sie die Schachfiguren gemäß der Grundstellung auf.
- c) Testen Sie Ihr Modul geeignet.

Aufgabe 17 (H) Induktion über den natürlichen Zahlen (7 + 3 = 10 Punkte)

Seien $n \in \mathbb{N}_0, k \in \mathbb{N}$ und das Prädikat

$$\text{divides}(k, n) = (\exists m \in \mathbb{N}_0 : m \cdot k = n)$$

gegeben. Die Funktion

$$\text{divides}_{\text{Imp}}(k, n) = \begin{cases} \text{true} & n = 0 \\ \text{false} & 0 < n < k \\ \text{divides}_{\text{Imp}}(k, n - k) & n \geq k \end{cases}$$

sei eine Implementierung des Prädikates *divides*.

- a) Zeigen Sie durch vollständige Induktion über n , dass

$$\mathit{divides}(k, n) = \mathit{divides}_{Imp}(k, n)$$

für $n \in \mathbb{N}_0, k \in \mathbb{N}$ gilt. (**Hinweis:** Beachten Sie, dass hier mehrere Induktionsanfänge zu bearbeiten sind!)

- b) Definieren Sie eine rekursive Funktion in OCaml-Syntax, die die Anzahl der rekursiven Aufrufe der Funktion $\mathit{divides}_{Imp}$ berechnet.