4. Übungsblatt zu Programmierung

Prof. Gert Smolka, Thorsten Brunklaus
www.ps.uni-sb.de/courses/prog-ws00/

Abgabe: 24. November 2000 in der Vorlesungspause (per Email)

Allgemeine Hinweise: Die Übungsblätter sollen in Zweiergruppen bearbeitet werden. Die Lösungen schicken Sie bitte bis Freitag 10 Uhr per Email an Ihren Übungsgruppenleiter. Jede Gruppe soll nur eine Lösung einreichen, versehen mit den Namen und den Matrikelnummern der Gruppenmitglieder, sowie der Übungsgruppennummer.

Aufgabe 4.1: Last (3) Schreiben Sie eine Prozedur

```
last : 'a list -> 'a
```

die das letzte Element einer Liste liefert. Wenn die Liste leer ist, soll die Ausnahme `Empty` geworfen werden.

Aufgabe 4.2: Enum (3) Schreiben Sie eine Prozedur

```
enum : int * int -> int list
```

die für zwei Zahlen $m \leq n$ die Liste $[m, m+1, \ldots, n]$ liefert. Für $n < m$ soll `enum` die leere Liste liefern.

Aufgabe 4.3: Nth, Take, Drop (4+4+4) Schreiben Sie drei Prozeduren

```
nth  : 'a list * int -> 'a
take : 'a list * int -> 'a list
drop : 'a list * int -> 'a list
```

wie folgt:

- `nth(`*xs*`,`*n*`)` liefert das *n*-te Element der Liste *xs*. Dabei sollen die Elemente von *xs* mit 0 beginnend nummeriert sein. Falls $n < 0$ oder $length(xs) \leq n$ gilt, soll die Ausnahme `Subscript` geworfen werden.
- `take(`*xs*`,`*n*`)` liefert die ersten *n* Elemente der Liste *xs*. Falls $n < 0$ oder $length(xs) < n$ gilt, soll die Ausnahme `Subscript` geworfen werden.
- `drop(`*xs*`,`*n*`)` liefert die Liste, die man aus *xs* erhält, wenn man die ersten *n* Elemente weglässt. Falls $n < 0$ oder $length(xs) < n$ gilt, soll die Ausnahme `Subscript` geworfen werden.

Hinweis: Verwenden Sie `orelse` und die Prozeduren `hd`, `tl` und `null`.

Aufgabe 4.4: Member (3+3+3) Sie sollen drei Varianten einer Prozedur

```
member : ''a -> ''a list -> bool
```

schreiben, die testet ob ein Wert in einer Liste vorkommt.

(a) Die erste Variante soll keine andere Prozedur benutzen.

(b) Die zweite Variante soll mit `List.exists` und ohne Rekursion geschrieben werden.

(c) Die dritte Variante soll mit `foldl` und ohne Rekursion geschrieben werden.

Aufgabe 4.5: Partition (4+4) Schreiben Sie zwei Varianten einer Prozedur

```
partition : int -> int list -> int list * int lis
```

die zu einer Zahl x und einer Liste *xs* zwei Listen *us* und *vs* wie folgt berechnet:

(a) *us@vs* ist eine Permutation von *xs*.

(b) Alle Elemente von *us* sind echt kleiner als x und alle Elemente von *vs* sind größer gleich als x.

Die erste Variante soll keine andere Prozedur benutzen. Die zweite Variante soll mit `foldl` und ohne Rekursion geschrieben werden.

Aufgabe 4.6: Quicksort (8) Quicksort ist ein klassischer Sortieralgorithmus, der auf der folgenden Beobachtung beruht.
Seien $x \in \mathbb{Z}$ und $xs, us, vs \in \mathcal{L}(\mathbb{Z})$ wie folgt:

(a) *us@vs* ist eine Permutation von *xs*.

(b) Alle Elemente von *us* sind echt kleiner als x und alle Elemente von *vs* sind größer gleich als x.

Weiter sei *sort* die Sortierfunktion $\mathcal{L}(\mathbb{Z}) \to \mathcal{L}(\mathbb{Z})$. Dann gilt die Gleichung

$$sort(x :: xs) \;=\; sort(us) \;@\; [x] \;@\; sort(vs)$$

Schreiben Sie mithilfe dieser Gleichung und der Prozedur `partition` aus Aufgabe 4.5 eine Sortierprozedur `qsort : int list -> int list`.

Aufgabe 4.7: Polymorphes Mergesort (7) Schreiben Sie eine polymorphe Sortierprozedur

```
pmsort : ('a * 'a -> bool) -> 'a list -> 'a list
```

die Listen durch Mischen sortiert.