7. Übungsblatt „Programmiersprachen“ SS 99

Leif Kornstaedt, Prof. Gert Smolka

Ausgabe: 4. Juni 1999 **Abgabe:** 11. Juni 1999

Strukturen und Funktoren, wie Sie sie in Aufgabe VII.2 schreiben sollen, werden von Moscow ML in dieser Form nicht unterstützt. Sie müssen daher SML/NJ oder MLWorks verwenden.

Aufgabe VII.1: Objekte in SML ($5 + 5$ Punkte)

Sie sollen Schlangen auf zwei Arten als imperative Datenstruktur in SML realisieren. Die Einfüge- und Ausfügeoperationen sollen dabei konstante Laufzeit haben.

a. Realisieren Sie Schlangen objektorientiert mit einer Prozedur

```
exception EmptyQueue

type 'a queue =
    {enqueue: 'a -> unit,
     dequeue: unit -> 'a,      (* EmptyQueue *)
     isEmpty: unit -> bool}

val queue: unit -> 'a queue
```

b. Realisieren Sie Schlangen als abstrakten Datentyp mit einer Signatur und einer Struktur.

Aufgabe VII.2: Funktoren ($5 + 2 + 3 + 4 + 4 + 5$ Punkte)

a. Schreiben Sie einen Funktor

```
functor Table(type key
              val compare: key * key -> order
              type entry) :>
    sig
        type table
        val empty: unit -> table
        val insert: table * key * entry -> unit
        val lookup: table * key -> entry option
    end =
    struct
        ...
    end
```

der Tabellen (Dictionaries) mit binären Suchbäumen implementiert.

Die Typen

```
datatype order = LESS | EQUAL | GREATER
datatype 'a option = NONE | SOME of 'a
```

sind vordefiniert.

b. Geben sie die prinzipale Signatur der Struktur

```
structure T =
    Table(type key = string
          type entry = int
          val compare = String.compare)
```

an.

c. Schreiben Sie eine Folge von Deklarationen, die mithilfe des Funktors `Table` eine Tabelle `t` mit folgendem Inhalt erzeugt:

```
"Maria"   [1, 2, 1]
"Johann"  [5, 2]
"Egon"    []
"Gabi"    [1, 1, 1, 1]
```

d. Schreiben Sie durch Rückgriff auf `Table` einen Funktor `Table'`, der Tabellen mit der zusätzlich Operation

```
val insertL: table * (key * entry) list -> unit
```

erzeugt. Öffnen Sie dabei die Hilfsstruktur mit **`open`**.

e. Geben Sie den Kopf (so wie in Aufgabe VII.2a) eines Funktors `HashTable` an, der Tabellen wie in Aufgabe VII.2a mit einer Hash-Tabelle der Größe

```
val size: int
```

und einer Hash-Funktion

```
val hash: key -> int
```

implementiert.

f. Implementieren Sie den Funktor `HashTable` mit Reihungen. Lesen Sie dazu die entsprechenden Stellen im Buch von Ullman oder im Info-1-Skript.