

Programmierpraktikum zur Informatik I

Testat am Fr., 21.12., Raum 709

In den folgenden Praktikumsaufgaben wollen wir ein Adressbuch implementieren. Hierfür benötigen wir zunächst eine Datenbank, welche Funktionen zum Ändern und Zugreifen der Einträge im Adressbuch zur Verfügung stellt.

Als erste Implementierung dieser Datenbank verwenden wir sortierte Listen. In den weiteren Aufgaben werden sie eine effizientere Implementierung mittels der in der Vorlesung eingeführten sortierten, binären Bäume, im folgenden Suchbäume (siehe effiziente Implementierung von Mengen) erstellen.

Aufgabe 3

Definieren Sie einen Datentypen **Entry**, welchen wir in den folgenden Aufgaben für die Einträge der Datenbank verwenden werden. Ein **Entry** besteht aus einem Schlüssel und einem zugehörigen Wert. Im Kontext einer Adressdatenbank kann man sich unter dem Schlüssel z.B. den Namen und unter dem Wert die Adresse vorstellen.

Entwerfen Sie auf Grundlage des Datentyps **Entry** einen ADT **Address** für die Einträge im Adressbuch. Als Schlüssel sollen Name und Vorname verwendet werden. Geben Sie hierzu Implementierungen der Vergleichsoperationen **less** und **equal** für Listen von Zeichenketten an. Zeichenketten, welche sich weiter hinten in der Liste befinden, müssen nur verglichen werden, wenn alle vorherigen Zeichenketten identisch waren. Mit diesem Ansatz können Sie dann Namen und Vornamen als Schlüssel verwenden. Informationen zu Zeichenketten und Vergleichsoperationen für Zeichenketten finden Sie in der Hilfe des DrScheme-Systems.

Als weitere Einträge soll die Adresse noch Straße, Hausnummer, Postleitzahl, Ort, Telefonnummer, EMail-Adresse und ein Feld für sonstige Bemerkungen enthalten.

Aufgabe 4

In dieser Aufgabe soll ein ADT **KeyMap** definiert werden, der als Datenbank für unser Adressbuch verwendet wird. Definieren Sie einen abstrakten Konstruktor (**emptyKeyMap less equal**) zur Konstruktion einer leeren **KeyMap**. In der Datenstruktur **KeyMap** sollen auch die Vergleichsfunktionen abgelegt werden, welche später benötigt werden, um Schlüssel miteinander zu vergleichen.

Als erste Implementierung wollen wir Listen von Schlüssel-Wert Paaren, also **Entrys** aus Aufgabe 3, verwenden, welche bzgl. der Schlüssel sortiert sind (im folgenden *Suchlisten* genannt). Implementieren sie folgende Operationen:

- (**insert e l**) sortiert den **Entry e** in die Suchliste **l** ein. Ist der Schlüssel bereits belegt, wird der alte Eintrag überschrieben.
- (**lookup key l**) sucht den unter dem Schlüssel **key** abgelegten Wert in der Suchliste **l**. Kommt **key** nicht in **l** vor, wird der Wert **null** zurückgeliefert.

- `(modify key fun l)` wendet die Funktion `fun` auf den Wert, der mit dem Schlüssel `key` assoziiert ist an. Alle anderen Elemente der Liste bleiben unverändert.
- `(delete key l)` löscht den Schlüssel `key` und den zugehörigen Wert in der Liste `l`.
- `(filter p l)` liefert eine Liste der Elemente, die das Prädikat `p` erfüllen. Das Prädikat ist auf Werten des Typs `Entry` definiert.
- `(isSorted l)` testet, ob eine gegebene Liste sortiert ist. Diese Funktion benötigen Sie um die anderen Funktionen zu testen.

Überlegen Sie sich für jede Funktion einen Test, in dem die Funktion mindestens 50 mal verwendet wird und die Sortiertheitseigenschaft für die Ergebnisse geprüft wird. Hilfreich kann in diesem Zusammenhang die Verwendung der vordefinierten Scheme-Funktion `random` sein.

Aufgabe 5

Zur generischen Eingabe bzw. Ausgabe von Daten nutzen wir im folgenden sogenannte *Masken*. Eine Maske ist eine Ein-/Ausgabebeschreibung eines Datensatzes einer Datenbank, indem insbesondere die Komponenten des Datensatzes mit Beschreibungen versehen werden. In dieser einfachen Implementierung einer Datenbank bestehen die Masken aus Zeichenketten, welche die einzelnen Komponenten benennen. In Anlehnung an die Paarstruktur eines Adressdatensatzes verwenden wir für die Implementierung einer Maske ebenfalls einen `Entry` von Listen von Zeichenketten. Für den ADT aus Aufgabe 3 kann z.B. folgende Maske definiert werden:

```
(define addressMask (make-entry (list "Name:" "Vorname:")
                                (list "Strasse:" "Hausnummer:" ...)))
```

Implementieren Sie eine Prozedur `showData`, welche einen Datensatz mit Hilfe einer Maske in eine Zeichenkette umwandelt. Diese Zeichenkette kann dann mit Hilfe der Funktion `display` ausgegeben werden. Die Ausgabe eines entsprechenden Datensatzes mit der obigen Maske soll folgende Ausgabe liefern:

```
Name          : Mustermann
Vorname       : Karl

Strasse       : Musterallee
Hausnummer    : 0
Postleitzahl  : 12345
Ort           : Musterhausen
Telefon       : 01234/56789
EMail         : muster@man.de
Sonstiges     :
```