

Programmierpraktikum P1

2. Aufgabenzettel

Abnahme bis zum 14.01.

In den folgenden Praktikumsaufgaben sollen Sie ein Adressbuch implementieren. Hierfür benötigen Sie zunächst einen abstrakten Datentyp, welcher Funktionen zum Ändern der Einträge und Zugreifen auf die Einträge im Adressbuch zur Verfügung stellt.

Als erste Implementierung dieses Datentyps verwenden wir geordnete Listen. In den weiteren Aufgaben werden Sie eine effizientere Implementierung mittels der in der Vorlesung eingeführten, geordneten, binären Bäume erstellen.

Hinweis: Um zu verhindern, dass Sie Ihre Implementierung mit einer **falschen/unschönen Datenstruktur** vornehmen, sollten Sie alle entworfenen Datenstrukturen **frühzeitig** bei einem **Praktikumstermin** einem Betreuer zeigen.

Aufgabe 4

Definieren Sie einen Datentyp **Entry**, welchen wir in den folgenden Aufgaben für die Einträge im Adressbuch verwenden werden. Ein **Entry** besteht aus einem Schlüssel und einem zugehörigen Wert.

In unserer Anwendung sollen Nachname und Vorname als Schlüssel verwendet werden. Als Werte sollen Sie Adressen verwenden. Genauer gesagt soll eine Adresse die Informationen Straße, Hausnummer, Postleitzahl, Ort, Telefonnummer, EMail-Adresse und ein Feld für sonstige Bemerkungen enthalten.

Implementieren Sie Vergleichsoperationen **less** und **equal** für Listen von Zeichenketten. Zeichenketten, welche sich weiter hinten in der Liste befinden, müssen nur verglichen werden, wenn alle vorherigen Zeichenketten identisch waren. Mit Hilfe dieser Funktionen können Sie später Nachnamen und Vornamen als Schlüssel verwenden. Informationen zu Zeichenketten und Vergleichsoperationen für Zeichenketten finden Sie in der Hilfe des DrScheme-Systems.

Aufgabe 5

In dieser Aufgabe soll ein ADT **KeyMap** implementiert werden, der zur Speicherung unserer Adressen verwendet wird. Der ADT **KeyMap** soll die folgenden Funktionen bereitstellen.

- (**emptyKM less equal**) konstruiert eine leere **KeyMap**. In der Datenstruktur sollen auch die Vergleichsfunktionen abgelegt werden, welche später benötigt werden, um Schlüssel miteinander zu vergleichen.
- (**lookup key km**) sucht den unter dem Schlüssel **key** abgelegten Wert in der **KeyMap km**. Kommt **key** nicht in **km** vor, wird der Wert **null** zurückgeliefert.
- (**insert e km**) sortiert den **Entry e** in die **KeyMap km** ein. Ist der Schlüssel bereits belegt, wird der alte Eintrag überschrieben.
- (**delete key km**) löscht den Schlüssel **key** und den zugehörigen Wert aus der **KeyMap km**.
- (**toList km**) liefert eine Liste aller Einträge in der **KeyMap km**.

- (`isSorted km`) testet, ob die KeyMap `km` sortiert ist. Diese Funktion benötigen Sie, um die anderen Funktionen zu testen. Diese Funktion gehört eigentlich nicht zur Schnittstelle des abstrakten Datentyps.

Als erste Implementierung dieses ADTs wollen wir Listen von Schlüssel-Wert Paaren verwenden, welche bzgl. der Schlüssel sortiert sind (im folgenden *Suchlisten* genannt). Implementieren Sie alle Funktionen des ADT KeyMap mit Hilfe von *Suchlisten*.

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

Aufgabe 6

Zur generischen Eingabe bzw. Ausgabe von Daten nutzen wir im folgenden sogenannte *Masken*. Eine Maske ist eine Ein-/Ausgabebeschreibung eines Datensatzes einer Datenbank, in dem 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 Adresdatensatzes verwenden wir für die Implementierung einer Maske ebenfalls einen **Entry** von Listen von Zeichenketten. Für den Datentyp aus Aufgabe 4 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@mann.de
Sonstiges    :
```

Achten Sie darauf, dass die Anzahl der eingefügten Leerzeichen von der Länge der Schlüsselnamen abhängt.

Aufgabe 7

Implementieren Sie eine Prozedur `inputData`, welche einen Datensatz mit Hilfe einer Maske einliest. Für die Eingabe soll die gleiche Formatierung wie für die Ausgabe verwendet werden. Das heißt, nach Eingabe der einzelnen Einträge soll exakt die gleiche Darstellung auf dem Bildschirm sichtbar sein wie bei der Ausgabe (also gleiche Formatierung verwenden).

Hinweis: Die Prozedur `readline` aus dem Modul `io.scm` (siehe Internetseite zum Praktikum) liest eine Zeile (bis zum Zeilenumbruch) von der Tastatur ein und liefert diese als Zeichenkette zurück.