Informatik I, Blatt Nr. 6, Abgabe: 21.01.2009 um 11 Uhr
http://proglang.informatik.uni-freiburg.de/teaching/info1/2009/

Hinweise zur Abgabe

Bitte reichen Sie Ihre Abgaben bis zum 21.01.2009 um 11 Uhr ein. Abgaben in elektronischer Form schicken Sie **per Email** an **Ihren** Tutor. Abgaben in Papierform werfen Sie bitte in den **Briefkasten** Ihrer Übungsgruppe im Geb. 051 im Erdgeschoss. Bei jeder Aufgabe ist angegeben, ob sie elektronisch oder auf Papier abgegeben werden muss.

Bei allen Aufgaben, die Sie per Mail abgeben, müssen Sie sich an die Namenskonventionen der Aufgaben halten. Dies gilt sowohl für die Dateinamen der Abgabe, als auch für Namen von Funktionen. Bitte geben Sie bei der elektronischen Abgabe nur eine Zip-Datei ab. Diese muss alle in den Aufgaben angegebenen `.scm` Dateien (DrScheme) enthalten. Alle Dateien müssen sich in der Zip-Datei in einem Ordner befinden. Der Name dieses Ordners muss Ihrem Loginnamen für den Rechnerpool des Instituts für Informatik entsprechen. Geben Sie unter keinen Umständen Worddokumente usw. ab!

Achten Sie bei der Papierabgabe darauf, dass jedes Blatt Papier Ihrer Abgabe Ihren Namen, Ihre Übungsgruppe, die Blattnummer und den Namen Ihres Tutors trägt. Falls Ihre Papierabgabe aus mehreren Seiten besteht, tackern Sie die Blätter.

Sie können DrScheme im Pool verwenden (starten mit `drscheme`). Achten Sie darauf, dass Sie jeweils das richtige Sprachlevel ausgewählt haben!

Punktevergabe

Um für die Programmieraufgaben Punkte zu erhalten, folgen Sie den Konstruktionsanleitungen der Vorlesung.

Einleitung

Auf diesem Blatt nähern wir uns den in der Vorlesung besprochenen abstrakten Datentypen am Beispiel des ADT "Priority Queue". Eine Priority Queue kann man sich grob vorstellen als einen Behälter, in den man einerseits Werte hineinlegen kann und andererseits den größten enthaltenen Wert entnehmen kann.

Der Datentyp `(pq %a)` von Priority Queues mit Elementen vom Typ `%a` sei gegeben durch die Verträge der Operationen

```
(: pq-empty? ((pq %a) -> boolean))    ; ist die PQ leer?
(: pq-insert ((pq %a) %a -> (pq %a) )) ; fügt einen Wert ein
(: pq-getmax ((pq %a) -> %a))         ; ermittelt das Maximum
(: pq-removemax ((pq %a) -> (pq %a) )) ; entfernt das Maximum
```

(Achtung: Diese Verträge gehören zur *Spezifikation* des ADT und sind, obwohl sie syntaktisch die Form von Scheme-Code haben, nicht direkt in der Implementation verwendbar!). Im folgenden werden Sie eine Implementation dieses ADT erstellen.

1 Aufgabe *[Sprache: Die Macht der Abstraktion, (1+1+1+1.5+1+1+1.5) Punkte]*

Eine einfache Implementation von Priority Queues verwendet eine absteigend geordnete Liste, um die Elemente zu speichern. Der User kann den Elementtyp und die Ordnung selbst bestimmen durch Angabe einer Vergleichsprozedur `my>=`.

(a) Definieren Sie einen zusammengesetzten Datentyp `list-pq`, der eine Liste und die Vergleichsprozedur enthält. Auch wenn in der Einleitung etwas von `(pq %a)` steht, reicht für diese Aufgabe ein nicht-parametrischer Typ `list-pq` (statt `(list-pq %a)`).

(b) Schreiben Sie als öffentlichen Konstruktor eine Prozedur `(: make-empty-list-pq ((%a %a -> boolean) -> list-pq)`, die eine leere `list-pq`-Instanz mit der gegebenen Vergleichsprozedur erzeugt.

(c) Schreiben Sie eine Prozedur `(: list-pq-empty? (list-pq -> boolean))`, die entscheidet, ob die Priority Queue leer ist. Testen Sie sie mit zwei Testfällen. Zunächst reichen Testfälle, in denen die zu testende Prozedur auf Testdaten trifft, die Sie mit `make-list-pq` zusammengebaut haben. Da Sie Ihre `list-pq` hier nicht als "black box" testen, heißen solche Tests *white-box*-Tests[1].

(d) Definieren Sie eine Prozedur `(: list-pq-insert (list-pq %a -> list-pq))`, die ein Element so in die Priority Queue einfügt, dass die absteigende Sortierung erhalten bleibt, und die aktualisierte Priority Queue zurückgibt. Testen: zwei Testfälle, white-box genügt.

(e) Definieren Sie eine Prozedur `(: list-pq-getmax (list-pq -> %a))`, die das größte Element der Priority Queue zurückliefert. Falls die PQ leer ist, soll mittels der bekannten Prozedur `violation` ein Fehler mit Text `"empty pq"` ausgelöst werden. Testen: zwei Testfälle, white-box genügt.

(f) Definieren sie eine Prozedur `(: list-pq-removemax (list-pq -> list-pq))`, die das größte Element der Priority Queue entfernt und die so aktualisierte PQ zurückliefert. Falls die PQ leer ist, soll wie oben ein Fehler mit Text `"empty pq"` ausgelöst werden. Testen: zwei Testfälle, white-box genügt.

(g) Schreiben Sie nun, um Ihr Vertrauen in die Korrektheit Ihrer ADT-Implementation zu erhöhen, drei *black box*-Testfälle. Sie dürfen also keine Konstruktoren oder Selektoren aus (a) verwenden, sondern nur die öffentlichen in (b)–(f) entwickelten Prozeduren. Die Testfälle müssen alle vier PQ-Operationen abdecken. Tipp: `let*` kann die Lesbarkeit verbessern.

2 Aufgabe

[Sprache: Die Macht der Abstraktion, (4+2) Punkte]

Verpacken Sie nun Ihre Priority-Queue-Implementation in einer ADT-Konstruktion wie in der Vorlesung:

(a) Schreiben Sie (analog zu `make-generic-list-set`) eine ADT-Factory `(: make-generic-list-pq ((%a %a -> boolean) -> (string -> %X)))`, die, gegeben eine Vergleichsprozedur, eine ADT-Prozedur zurückliefert, die kontrollierten Zugriff auf die PQ bietet. Diese Prozedur soll einen String m akzeptieren und

- für $m =$ `"empty?"` eine 0-äre[2] Prozedur liefern, die bei Aufruf zurückliefert, ob die PQ leer ist.
- für $m =$ `"insert"` eine 1-äre Prozedur liefern, die, mit einem Wert x aufgerufen, diesen Wert in die PQ einfügt und eine ADT-Prozedur für die neue PQ zurückliefert.
- für $m =$ `"getmax"` eine 0-äre Prozedur liefern, die bei Aufruf das größte Element der PQ zurückliefert oder, wenn die PQ leer ist, einen Fehler mit Text `"empty pq"` erzeugt.
- für $m =$ `"removemax"` eine 0-äre Prozedur liefern, die bei Aufruf das Maximum der PQ entfernt und eine ADT-Prozedur für die neue PQ zurückliefert. Wenn die PQ leer ist, soll ein Fehler mit Text `"empty pq"` erzeugt werden.

(b) Testen Sie Ihre ADT-Factory mit

```
(check-expect (let* ([pq1 (make-generic-list-pq >=)])
                ((pq1 "empty?")) ) #t)
```

und zwei bis drei weiteren Black-Box-Tests, die alle vier Operationen abdecken.

[1] manchmal auch *clear-box* oder *glass-box*

[2] $n-$äre Prozeduren sind Prozeduren mit n Parametern. 1-är heißt auch unär, 2-är binär.

3 Aufgabe *[Sprache: Die Macht der Abstraktion, (6) Punkte]*

Eine Anwendung von Priority-Queues ist das Sortieren. Schreiben Sie eine Prozedur

```
(: heap-sort ((string -> %X) (list %a) -> (list %a)))
```

die eine Priority-Queue-ADT-Prozedur und eine Liste nimmt und die Liste sortiert zurückliefert, so dass das, was die PQ fuer das Maximum hält, an letzter Stelle steht. (Strategie: alles in die PQ einfügen und dann immer wieder das Maximum herausholen).

Testen: drei Testfälle.

Wenn Sie Aufgabe 2 nicht gelöst haben, können Sie in Ihrer Sortierprozedur auch die Implementation aus Aufgabe 1 direkt verwenden.

4 Aufgabe *[Sprache: Die Macht der Abstraktion, (1+1+1 Bonuspunkte) Punkte]*

Schwager Konstantin hat frei nach den Illustrationen in seinem Algorithmenbuch improvisiert und eine, wie er sagt, hocheffiziente Priority Queue mit Bäumen geschrieben. Herunterladbar unter
`http://proglang.informatik.uni-freiburg.de/teaching/info1/2009/uebungen/cf-pq.scm`. Die Prozeduren heißen wie in Aufgabe 1, aber im Namen steht statt `list-pq` jeweils `cf-pq`.

(a) Schreiben Sie auch für diese Implementation eine ADT-Factory wie in Aufgabe 2. Wenn Sie gemeinsamen Code in Aufgabe 2 und hier entdecken, können Sie ihn auch gerne in eine Hilfsprozedur auslagern.

(b) Passen Sie die Testfälle aus Aufgabe 2 an diese ADT-Implementation an und testen Sie so, dass die neue ADT-Implementation mit Ihrer listenbasierten Implementation austauschkompatibel ist.

(c) Testen Sie mit drei Testfällen, dass Ihre Sortierprozedur aus Aufgabe 3 auch mit Konstantins toller PQ richtig sortiert. Wiederverwendung von Aufgabe 3 ist erlaubt.

Versionsgeschichte

2010-01-07 Erste Fassung

2010-01-11 Fehlerhafte Verträge von list-pq-removemax und pq-removemax korrigiert. Klarstellung, dass die Verträge in der Einleitung nur Spezifikation sind.

2010-01-12 Fehlerhafter Vertrag von pq-getmax korrigiert.

Abgabe: elektronisch als Datei `pq.scm`. Wenn Sie noch Fragen haben, können Sie auch gerne das Forum benutzen.