14. Übungsblatt zu Programmierung 1, WS 2008/09

Prof. Dr. Gert Smolka, Mark Kaminski, M.Sc.
www.ps.uni-sb.de/courses/prog-ws08/

Lesen Sie im Buch: Kapitel 15

Aufgabe 15.7 Schreiben Sie eine Prozedur *sum* : *int array* → *int*, die zu einer Reihung die Summe ihrer Komponenten liefert. Verwenden Sie *Array.foldl*.

Aufgabe 15.10 (Histogramme) Ein Histogramm ist eine grafische Darstellung von Häufigkeiten in Form von Säulen. Schreiben Sie eine Prozedur *histo* : *int list* → *int* → *int list*, die zu *xs* und *n* eine Liste der Länge $n + 1$ liefert, deren i-te Position angibt, wie oft die Zahl i in der Liste *xs* vorkommt. Beispielsweise soll *histo* $[2, 0, 3, 2, 2, 0]$ $2 = [2, 0, 3]$ gelten.

Aufgabe 15.12 (Rotieren) Schreiben Sie eine Prozedur *rotate* : α *array* → *unit*, die die Komponenten einer Reihung um eine Position nach rechts verschiebt und die letzte Komponente an die Stelle der ersten Komponente rückt. Beispielsweise soll die Komponentenfolge [1, 2, 3, 4] zu [4, 1, 2, 3] geändert werden. Verwenden Sie eine Hilfsprozedur *rotate*′ : *int* → α → *unit*, die für l und x alle Positionen ab l um eins nach rechts verschiebt und die Position l auf x setzt. Die letzte Komponente soll dabei verloren gehen.

Aufgabe 15.16 Zeichnen Sie die Kastendarstellungen der Zustände, die die Schlange q gemäß den folgenden Deklarationen durchläuft.

```
open Queue
val q = queue ()
val _ = (insert q 7 ; insert q 13 ; remove q)
```

Aufgabe 15.24 Betrachten Sie den Haldenzustand, der sich durch die Ausführung des folgenden Ausdrucks ergibt:

```
(release 0 ; putList [13,17] ; putList [19,23] ; update 0 1 6)
```

a) Zeichnen Sie die verzeigerte Blockdarstellung der durch die Adresse 2 dargestellten Liste.
b) Welche Liste liefert *getList* für die Adresse 2?

Aufgabe 15.25 Deklarieren Sie Prozeduren, die mit der durch *putList* formulierten Haldendarstellung von Listen arbeiten:

a) Eine Prozedur *null* : *int* → *bool*, die testet, ob eine Liste leer ist.
b) Eine Prozedur *head* : *int* → *int*, die den Kopf einer Liste liefert. Falls die Liste leer ist, soll die Ausnahme *Empty* geworfen werden.
c) Eine Prozedur *tail* : *int* → *int*, die den Rumpf einer Liste liefert. Falls die Liste leer ist, soll die Ausnahme *Empty* geworfen werden.
d) Eine Prozedur *cons* : *int* → *int* → *int*, die zu einer Zahl *x* und zu einer Darstellung einer Liste *xr* eine Darstellung der Liste *x* :: *xr* liefert.
e) Eine Prozedur *append* : *int* → *int* → *int*, die die Konkatenation zweier Listen liefert.
f) Zeichnen Sie die verzeigerten Blockdarstellungen der durch die Ausführung der folgenden Deklarationen in der Halde dargestellten Listen. Nehmen Sie dabei an, dass die Halde zu Beginn leer ist.

```
val a0 = putList[]
val a1 = putList [1,2,3]
val a2 = append a1 a0
val a3 = append a1 a1
```

Aufgabe 15.26 Deklarieren Sie eine Prozedur *extend* : *int* → *int* → *int*, die zu zwei Listendarstellungen *a* und *b* eine Darstellung der Konkatenation der von *a* und *b* dargestellten Listen liefert. Dabei sollen keine neuen Zellen alloziert werden. Stattdessen darf die Darstellung von *a* in der Halde modifiziert werden. Hinweis: Schreiben Sie zuerst eine Prozedur *extend′*, die annimmt, dass die durch *a* dargestellte Liste nicht leer ist.

Aufgabe 15.27 Überlegen Sie sich, wie Optionen über *int* in der Halde dargestellt werden können. Schreiben Sie entsprechende Prozeduren *putOption* und *getOption*.

Aufgabe 15.28 Deklarieren Sie eine Prozedur *tree* : *int* → *int*, die für $n \geq 0$ den balancierten Binärbaum der Tiefe *n* in der Halde darstellt. Dabei sollen mehrfach vorkommende Teilbäume nur einmal in der Halde dargestellt werden, sodass für die Darstellung eines balancierten Binärbaums der Tiefe *n* genau $4n$ Zellen alloziert werden.

Aufgabe 15.30 Deklarieren Sie eine Prozedur *eval* : (*var* → *int*) → *int* → *int*, die einen in der Halde dargestellten Ausdruck gemäß einer Umgebung evaluiert.

Aufgabe 15.31 Schreiben Sie eine Prozedur *instantiate* : (*var* → *int*) → *int* → *unit*, die die Variablen eines in der Halde dargestellten Ausdrucks gemäß einer Umgebung in Konstanten umwandelt. Dabei sollen keine neuen Zellen alloziert werden.

Aufgabe 15.32 Wieviele Zellen allozieren die folgenden Prozeduren für Listen der Länge *n*?

a) *length*
b) *map* (*fn x* ⇒ *x*)
c) *foldl op*:: *nil*
d) *exists* (*fn x* ⇒ *x* > *5*)

Aufgabe 15.33 Schreiben Sie eine naive Reversionsprozedur *reverse* : *int* → *int*, die mit der durch *putlist* formulierten Haldendarstellung von Listen arbeitet. Verwenden Sie eine Prozedur *append*, die eine Darstellung der Konkatenation zweier Listen liefert. Zeichnen Sie die verzeigerte Blockdarstellung der durch die folgenden Deklarationen allozierten Haldenzellen:

```
val _ = release 0
val a = putList [3,7]
val b = reverse a
```

Markieren Sie die Zellen, die weder von *a* noch von *b* aus erreichbar sind.

Folgeaufgabe für Aufgabe 14.7 (Puzzle) Mit imperativen Prozeduren kann Aufgabe 14.7 effizienter gelöst werden. Seien die Struktur *Puzzle* und die Deklaration

```
val r = ref nil
```

gegeben. Schreiben Sie eine Prozedur *p* : *int* → *bool*, sodass *Puzzle.find p* eine Prozedur *Puzzle.set* → *int option* liefert, die bei der Anwendung auf eine Menge die Liste ihrer Argumente unter der Referenz *r* ablegt.