Fachrichtung 6.2 — Informatik
Universität des Saarlandes
Tutorenteam der Vorlesung Programmierung 1

Programmierung 1 (Wintersemester 2015/16)
Wiederholungstutorium Lösungsblatt 14

(Speicherzugriff, Arrays)

Hinweis: Dieses Übungsblatt enthält von den Tutoren für das Wiederholungstutorium erstellte Aufgaben.
Die Aufgaben und die damit abgedeckten Themenbereiche sind für die Klausur weder relevant noch irrelevant.

Referenzen

Aufgabe 14.1 (*Speicherbefehle*)

(a) Deklarieren Sie einen Bezeichner `r: int ref`, der eine Zelle bezeichnet, die initial mit 10 belegt ist. Setzen Sie `r` auf 30. Setzen Sie `r` auf den um eins erhöhten Wert von `r`.

(b) Deklarieren Sie einen Bezeichner `l: α list ref`. Setzen Sie `l` auf `[1,2,3]`. Setzen Sie `l` auf `["a", "gh3"]`. Was passiert? Setzen Sie `l` auf `[6,3,2,8,5,3,5,7,8,9,4,2]`. Hängen Sie vorn an die Liste in `l` eine 4 an, ohne dabei den Rest der Liste noch einmal auszuschreiben. Setzen Sie `l` auf *nil*.

(c) Deklarieren Sie einen Bezeichner `b: (bool * int * (α list list * int list)) ref` und setzen Sie ihn auf einen anderen Wert. Geben Sie den Wert von `b` aus.

(d) Wonach werten `ref 0 = ref 0` und direkt danach ausgeführt `!(ref 0) = !(ref 0)` aus? Warum?

Lösung 14.1:

(a)
```
val r = ref 10
(r := 30; r := 1+!r)
```

(b)
```
val l = ref nil
(l := [1,2,3];
l := ["a","gh3"]; (* funktioniert nicht, weil l jetzt den Typ int list hat *)
l := [6,3,2,8,5,3,5,7,8,9,4,2]; l := 4::!l; l := nil)
```

(c)
```
val b = ref (true,3,(nil::nil,[2]))
(b := (false,1,(nil::nil,[2])); !b)
```

(d) Im ersten Fall wird auf jeder Seite eine 0 in eine Speicherzelle geschrieben, wobei jede Seite zur Referenz auf ihre jeweilige Speicherzelle auswertet. Da beide Speicherzellen verschieden sind, sind die Referenzen auf sie auch verschieden, weshalb die Aussage nach *false* auswertet. Im zweiten Fall wird die Referenz dereferenziert. Das heißt, dass jede Seite jeweils zum Inhalt ihrer Speicherzellen auswertet, der in beiden Fällen 0 ist. Daher wertet die zweite Aussage nach *true* aus.

Aufgabe 14.2 (*Swapper*)

Wozu wertet dieser Ausdruck aus?

```
let fun mystery a b c d = b := !(#2 (d := !a, d, d := !c))
    val (a,b,c,d) = (ref 1, ref 2, ref 3, ref 4)
in (mystery a b c d; (!a, !b, !c, !d)) end
```

Lösung 14.2:

$(1, 3, 3, 3)$

Aufgabe 14.3 (*Imperative Listen*)

(a) Schreiben Sie eine Prozedur `save : α list → α ref list`, welche eine Liste Element für Element speichert und eine Liste aller Referenzen zurückgibt.

(b) Schreiben Sie eine Prozedur `map : (α → β) → α ref list → unit`, welche eine Prozedur f auf alle Elemente einer imperativen Liste anwendet.

(c) Schreiben Sie eine Prozedur `foldl : (α * β → β) → β → α ref list → β`, welche wie foldl funktioniert, nur imperativ.

(d) Schreiben Sie eine Prozedur `rev : α ref list → unit`, die eine Liste im Speicher reversiert, das bedeutet die Speicherzellen bleiben die selben und auch in der selben Reihenfolge, nur der Inhalt wird vertauscht. Zum Beispiel: A[1] B[2] C[3] D[4] $\rightsquigarrow$ A[4] B[3] C[2] D[1] (Wobei X[] eine Speicherzelle darstellt)

Hinweis: Sie dürfen rev für funktionale Listen benutzen.

Lösung 14.3:

(a)
```
fun save nil = nil
  | save (x::xr) = (ref x) :: save xr
```

(b)
```
fun map f nil = nil
  | map f (x::xr) = (x := (f (!x); map f xr)
```

(c)
```
fun foldl f s nil = s
  | foldl f s (x::xr) = foldl f (f(!x,s)) xr
```

(d)
```
fun rev nil = ()
  | rev xs = let fun load nil = nil
                   | load (x::xr) = !x :: load xr
                fun update nil nil = ()
                  | update (x::xr) (y::ys) = (x:= y; update xr ys)
             in update xs (List.rev (load xs)) end
```

Generatoren

Aufgabe 14.4 (*Einführung Counter und Generator*)

(a) Weisen Sie der Referenz r den Anfangswert 0 zu.

(b) Deklarieren Sie nun eine Prozedur `excounter : unit → int`, welche auf die externe Deklaration von r zugreifen kann und bei jedem Aufruf den Wert der Referenz r um 1 erhöht.

(c) Damit die Referenz r nicht allen Programmen zugänglich ist, schreiben Sie nun eine Prozedur `incounter : unit → int`, welche r lokal deklariert. Sie soll wie eben bei jedem Aufruf den Wert der Referenz r um 1 erhöht.

(d) Wenn Sie nun `incounter` 5 mal hintereinander aufrufen, welchen Wert liefert dann !r ?

(e) Wir wollen nun das Erstellen von Countern automatisieren, dazu benötigen wir sogenannte Generatoren. Schreiben Sie eine Prozedur `newgen : unit → unit → int` die bei jedem Aufruf einen neuen Counter erzeugt.

(f) Erweitern Sie `newgen` zu `newgen' : int → unit → int`, sodass Sie nun den Anfangswert beliebig setzten können.

Lösung 14.4:

(a)
```
val r = ref 0
```

(b)
```
fun excounter () = (r:= !r +1; !r)
```

(c)
```
val incounter = let val r = ref 0 in fn () ⇒ (r:= !r +1; !r)
```

(d)
```
5
```

(e) `fun newgen () = let val r = ref 0 in fn () ⇒ (r:= !r +1; !r)`

(f) `fun newgen' x = let val r = ref x in fn () ⇒ (r:= !r +1; !r)`

Aufgabe 14.5 (*Viele Generatoren*)

(a) Schreiben Sie eine Prozedur `q: unit → unit → int`, die eine Prozedur zurückgibt, die nach n Aufrufen die n-te Quadratzahl zurückgibt.

(b) Schreiben Sie eine Prozedur `impiter: (α → α) → α → unit → α`, die eine Prozedur zurückgibt, die nach n Aufrufen das Ergebnis der n-fachen Anwendung einer Prozedur f auf einen Startwert s zurückgibt.

(c) Schreiben Sie eine Prozedur `fak: unit → unit → int`, die eine Prozedur zurückgibt, die beim n-ten Aufruf $n!$ zurückgibt.

(d) Schreiben Sie eine Prozedur `tausch: α ref → α ref → unit`, die den Inhalt zweier Speicherzellen miteinander vertauscht. Nutzen Sie keine lokalen Deklarationen.

(e) Schreiben Sie eine Prozedur `fib: unit → unit → int`, die eine Prozedur zurückgibt, die beim n-ten Aufruf die n-te Fibonacci-Zahl zurückgibt.

Lösung 14.5:

(a)
```
fun q () = let val a = ref 0
           in fn () ⇒ (a := (!a+1); !a * !a)
           end
```

(b)
```
fun impiter f s = let val a = ref s in (a := (f(!a)); !a) end
```

(c)
```
fun fak () = let val zaehler = ref 0
                 val fakultaet = ref 1
             in fn () ⇒ (zaehler:=(1+!zahler);
                  fakultaet:=(!zaehler * !fakultaet);
                  !fakultaet)
             end
```

(d)
```
fun tausch a b = a:= #1(!b, b:=!a)
```

(e)
```
fun fib () = let val eins = ref 0
                 val zwei = ref 1
             in fn () ⇒ #1(!eins, zwei:=#1(!eins+!zwei, eins:=!zwei))
            end
```

Aufgabe 14.6 (*Schafe zählen*)

Vervollständigen Sie die Deklaration `val (schwarzesSchaf,zaun,weissesSchaf) = ...` , sodass sie einen eingekapselten Zähler mit Anfangswert 0 und drei Prozeduren wie folgt liefert:

- `schwarzesSchaf : unit → int`
  Bei einem schwarzen Schaf soll die erreichte Zahl halbiert (abgerundet) werden.
- `zaun : unit → int`
  Bei einem Zaun soll von der erreichten Zahl wieder 5 abgezogen werden.
- `weissesSchaf : unit → int`
  Beim ersten weißen Schaf soll zunächst 1 addiert werden, wenn direkt danach wieder ein weißes Schaf folgt 2, beim dritten 3 usw. Falls ein schwarzes Schaf dazwischen kommt, soll beim nächsten weißen wieder mit +1 angefangen werden.
- Wenn man die 1000 erreicht, soll eine Ausnahme Dream geworfen werden und man schläft ein.

Lösung 14.6:

```
exception Dream
val (schwarzesSchaf,zaun,weissesSchaf) =
  let val r  = ref 0
      val t  = ref 1
  in (fn () ⇒ #1(!r div 2, r:= !r div 2, t:= 1),
      fn () ⇒ #1(!r - 5, r:= !r - 5),
      fn () ⇒ #3(r:= !r + !t , t:= !t + 1, if !r < 1000 then !r else raise Dream))
  end
```

Arrays

Aufgabe 14.7 (*Faltung auf Arrays*)

(a) Schreiben Sie eine Prozedur `minmax: int array → unit`, die das Minimum und Maximum einer Reihung von Zahlen bestimmt und sie miteinander tauscht.

(b) Schreiben Sie eine Prozedur `copy:` α `array →` α `array`, die eine Kopie einer Reihung erstellt.

Lösung 14.7:

(a)
```
fun minmax xs = let val (posmin,posmax,_) =
                            Array.foldl (fn (x,(pmin,pmax,pos)) ⇒
                                if x>Array.sub(xs,pmax) then (pmin,pos,pos+1)
                                else if x<Array.sub(xs,pmin) then (pos,pmax,pos+1)
                                else (pmin,pmax,pos+1)) (0,0,0) xs
                    val (min,max) = (Array.sub(xs,posmin),Array.sub(xs,posmax))
                in (Array.update(xs,posmin,max); Array.update(xs,posmax,min))
                end
```

(b)
```
fun copy xs = Array.fromList (Array.foldr op:: nil xs)
```

Alternative:

```
fun copy xs = if Array.length xs = 0 then Array.fromList nil
              else let val erstes = Array.sub(xs, 0)
                       val initial = Array.array(Array.length xs, erstes)
                   in (Array.foldl (fn (x,i) ⇒
                        (Array.update(initial, i, x); i+1)) 0 xs; initial)
                   end
```

Aufgabe 14.8 (*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.

Lösung 14.8:

```
fun rotate a =
    let fun rotate' x l u =
          if l > u then ()
          else rotate' (#1(Array.sub(a,l),Array.update(a,l,x))) (l+1) u
    in rotate' (Array.sub(a,Array.length a -1)) 0 (Array.length a -1) end
```

Aufgabe 14.9 (*Reversieren von Arrays*)

(a) Schreiben Sie eine Prozedur `display : α array → α list`, welche ihnen eine Array als Liste ausgibt. Diese ist hilfreich, wenn Sie am Interpreter arbeiten und schauen wollen, ob ihre Array-Prozedur korrekt ist.

(b) Schreiben Sie eine Prozedur `arrev : α array → unit`, welche eine Reihung in situ reversiert (= Datenkonstrukt bleibt erhalten aber die Werte werden vertauscht).

Lösung 14.9:

(a)
```
fun display a = Array.foldr (op::) nil a
```

```
fun arrev a = let fun swap l u = let val x = Array.sub(a,l)
                                 in (Array.update(a,l,Array.sub(a,u));
                                     Array.update(a,u,x)) end
                  fun rev l u = if u <= l then ()
                                else (swap l u; rev (l+1) (u-1))
              in rev 0 (Array.length a - 1) end
```

Aufgabe 14.10 (*Sortieren durch Auswählen*)

Schreiben Sie eine Prozedur `ssort : α array → unit`, welche einen Array (von l bis u) durch Auswählen sortiert. Gehen Sie wie folgt vor:

(a) Bestimmen Sie die Position m einer Komponente minimaler Größe.

(b) Vertauschen Sie die Komponenten an den Positionen l und m.

(c) Sortieren Sie den Array von (l+1,u)

(d) Geben Sie die Laufzeit ihrer Prozedur an.

Lösung 14.10:

```
fun ssort a = let fun swap l u = let val x = Array.sub(a,l)
                                 in (Array.update(a,l,Array.sub(a,u));
                                     Array.update(a,u,x)) end
                  fun min l u m = if l > u then m
                     else if Array.sub(a,l) < Array.sub(a,m) then min (l+1) u l
                     else min (l+1) u m
                  fun ssort' l u = if l > u then ()
                                   else let val m = min l u l
                                        in (swap m l; ssort' (l+1) u) end
              in ssort' 0 (Array.length a - 1) end
```

Aufgabe 14.11 (*Inferenzregeln*)

Ergänze folgende Inferenzregeln für die Speicheroperationen *Allokation*, *Dereferenzierung* und *Zuweisung*:

$$\text{Sref}\ \frac{}{T \vdash \qquad :}$$

$$\text{S!}\ \frac{}{T \vdash \qquad :}$$

$$\text{S:=}\ \frac{}{T \vdash \qquad :}$$

Lösung 14.11:

$$\text{Sref}\ \frac{T\ \vdash\ x\ :\alpha}{T\ \vdash\ \mathit{ref}\ x\ :\alpha\ \mathit{ref}}$$

$$\text{S!}\ \frac{T\ \vdash\ x\ :\alpha\ \mathit{ref}}{T\ \vdash\ !x\ :\alpha}$$

$$\text{S:=}\ \frac{T\ \vdash\ a\ :\alpha\ \mathit{ref} \qquad T\ \vdash\ x\ :\alpha}{T\ \vdash\ a := x\ :\mathit{unit}}$$