Prof. Bernd Krieg-Brückner
Shi Hui
Kolyang
Axel Lankenau
Christoph Lüth
Lutz Zegartowski

5. Übungsblatt

Ausgabe: 17.12.98
Abgabe: Woche 25.01. – 29.01.99

---

1 *Neuester Trend aus den USA: Extreme Palindroming…* *Wichtung: 30%*

Seit einiger Zeit entwickelt sich in Amerika ein neuer Volkssport: *Extreme Palindroming*. Die Regeln sind selbstverständlich denkbar einfach: Nimm eine natürliche Zahl, drehe sie um und addiere beide. Der Kick dabei ist (ähnlich wie beim Bungee Jumping ohne Seil), daß das Ergebnis nach einiger Zeit von vorne genauso aussieht wie von hinten.

1. Schreibt eine Funktion `checkPali :: Int -> (Int,[Int])`, die eine natürliche Zahl übergeben bekommt und als Ergebnis die Anzahl der bis zur Entstehung eines Palindroms nötigen Additionen sowie das Palindrom selbst als Liste seiner Ziffern liefert.
   Wichtig: Da die Zahlen unter Umständen sehr lang werden können, müßt Ihr sie als Listen von Ziffern repräsentieren. So läßt sich 1024 z.B. schreiben als `[1,0,2,4]`.

   Beispiel: `calcPali 84` sollte ausgewertet werden zu `(2,[3,6,3])`, da 84+48=132. 132 ist kein Palindrom, also weiter addieren: 132+231=363. Fertig, da 363 von vorne und hinten gelesen gleich ist. Nach zwei Additionen ergibt sich das Palindrom 363.

2. Welche Zahl $n < 100$ benötigt die meisten Additionen, bis sie zum Palindrom wird?

2 *Haskell's not UNIX …* *Wichtung: 40%*

Das UNIX-Dateisystem ist als Baum strukturiert. Hier soll vereinfachend angenommen werden, daß eine Datei in jedem Fall einen Namen besitzt und entweder eine Textdatei oder ein Verzeichnis (eine Liste von Dateien) ist. Das aktuelle Arbeitsverzeichnis läßt sich über eine Pfadangabe (z.B. `/usr/local/bin`) finden. Damit läßt sich das ganze Dateisystem als Paar von einer Datei und dem Pfad zum aktuellen Arbeitsverzeichnis darstellen!

1. Definiert die Datentypen `File` für Dateien (Textdatei oder Verzeichnis), `Fsys` für Dateisysteme und `Path` für Pfadangaben.

2. Implementiert folgende Operationen:

   `pwd :: Fsys -> String` liefert das aktuelle Arbeitsverzeichnis in einem Dateisystem als Zeichenkette. In der Ausgabe sind Verzeichnisse dabei durch `/` voneinander getrennt.

   `ls :: Fsys -> String` gibt eine Auflistung der Einträge (Dateien und Unterverzeichnisse) im durch den Pfad bestimmten Arbeitsverzeichnis. Strings lassen sich übrigens mit der `putStr` Funktion schön formatiert ausgeben.

   `edit :: Fsys -> String -> String -> Fsys` erzeugt im aktuellen Arbeitsverzeichnis eine Textdatei mit dem Namen, der durch das erste `String`-Argument übergeben wurde. Der Inhalt wird durch das zweite `String`-Argument spezifiziert. Falls bereits eine gleichnamige Datei existiert, wird deren Inhalt ersetzt. Es wird das veränderte Dateisystem zurückgegeben.

   `mkdir :: Fsys -> File -> Fsys` erzeugt im aktuellen Arbeitsverzeichnis ein Unterverzeichnis. Gibt das veränderte Dateisystem zurück.

   `cd :: Fsys -> String -> Fsys` wechselt das Verzeichnis. Das Argument vom Typ `String` kann dabei neben den Namen der Unterverzeichnisse auch den Sonderfall `".."` als Wert haben. Die Rückgabe ist das veränderte Dateisystem.

`cp :: Fsys -> File -> Path -> Fsys` kopiert die übergebene Datei entsprechend dem Zielpfad. Als Sonderfall ist dabei der Punkt (`"."`) als Zielpfad für das aktuelle Verzeichnis zu berücksichtigen. Alle anderen Zielpfadangaben sind relativ zum aktuellen Arbeitsverzeichnis zu verstehen. Rückgabewert ist das veränderte Dateisystem.

`rm :: Fsys -> File -> Fsys` löscht die angegebene Datei und liefert als Ergebnis das veränderte Dateisystem.

3. Testet die Funktionen, indem Ihr ein Dateisystem aufbaut, daß mindestens drei Hierarchieebenen hat. In jedem Verzeichnis sollen mindestens zwei Unterverzeichnisse und zwei Textdateien sein.

Zusatzaufgabe: Implementiert eine rekursive Variante von `ls`. Die Funktion `lsr :: Fsys -> String` soll eine ansehnliche Auflistung aller Dateien und Unterverzeichnisse ergeben (vergleichbar mit dem `ls -R` Kommando unter UNIX!).

3 *Ready,* ***set*** *... GO!* — *Wichtung: 30%*

Der von Aufgabenblatt 4 bekannte ADT `Set` soll mit folgender Signatur implementiert werden:

```
type Set a = [a]
  in
  empty     :: Set a,
  singleton :: a -> Set a,
  makeSet   :: Eq a => [a] -> Set a,
  isElem    :: Eq a => a -> Set a -> Bool,
  subSet    :: Eq a => Set a -> Set a -> Bool,
  union     :: Eq a => Set a -> Set a -> Set a,
  intersect :: Eq a => Set a -> Set a -> Set a,
  setdiff   :: Eq a => Set a -> Set a -> Set a,
  showSet   :: Show a => Set a -> String
```

Der Typ `Set` wird als Haskell–Liste implementiert. Im Gegensatz zu herkömmlichen Listen darf aber ein Element *bei der Ausgabe* der Menge nur einmal vorkommen. Dies läßt sich auf mindestens zwei Weisen umsetzen. Welche wählt Ihr? Begründung! Die Reihenfolge der Elemente ist irrelevant!

Die auf dem ADT `Set` definierten Funktion arbeiten wie folgt: `empty` liefert eine leere Menge, während `singleton x` eine einelementige Menge erzeugt. Die Funktion `makeSet` wandelt eine Liste in eine Menge um, `isElem` prüft ob ein Element in der Menge enthalten ist. Das Prädikat `subSet` gibt an, ob eine Menge Teilmenge einer anderen ist. Die Operationen `union`, `intersect` und `setdiff` bestimmen aus zwei Mengen deren Vereinigungs–, Schnitt– bzw. Differenzmenge. Schließlich soll die Funktion `showSet` die Möglichkeit bieten, eine Menge in einen String zu verwandeln, der dann ausgegeben werden kann.

1. Implementiert die in der Signatur aufgeführten Funktionen.

2. Schreibt eine Funktion `setEq :: Eq a => Set a -> Set a -> Bool`, die `True` liefert, falls die beiden übergebenen Mengen identisch sind und ansonsten `False`. Vorsicht: `setEq (makeSet [2,1,3]) (makeSet [3,2,1])` sollte `True` ergeben!

3. Diskutiert die Vorteile der Kapselung der Mengen und der darauf definierten Operationen in einem ADT! Ein Jahr nach der Veröffentlichung Eurer Implementierung des ADT `Set` finden Informatik–Studenten an der renommierten Universität in Bremen heraus, daß es wesentlich effizienter ist, Mengen als *geordnete* Listen darzustellen. Welche Updates müssen die Millionen Anwender Eures ADTs an ihrem eigenen Code vornehmen, falls die interne Repräsentation der Mengen im ADT geändert wird?