Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion
CAU Kiel, Informatik und Prakt. Mathematik
Dr. Frank Huch

24118 Kiel
Christan-Albrechts-Platz 4
Tel.: 0431 / 880-7277

1. Übung „Funktionale Programmierung“
Abgabe am 13. April 2005 vor der Vorlesung

In dieser ersten Übung sollen Sie sich mit dem Hugs-System vertraut machen.
Public domain Versionen und weitere Informationen zur Vorlesung und den Übungen finden Sie im WWW unter der Adresse:

```
http://www-ps.informatik.uni-kiel.de/~fhu/FP05
```

Beachten Sie folgende Regeln bei dieser und den kommenden Abgaben:

- Abgabe in Zweiergruppen.
- Drucken Sie den gesamten programmierten Code aus.
- Drucken Sie mindestens zwei *sinnvolle* Testläufe (z.B. copy+paste aus dem Hugs-System) aus.
- Kommentieren Sie Ihren Code, falls die definierte Funktion etwas komplizierter wird oder sie zusätzliche Hilfsfunktionen definieren.
- Annotieren Sie jede definierte Funktion mit einem Typ.

Aufgabe 1 2 Punkte

Die Funktion `double` sei wie folgt definiert:

```
double x = x + x
```

Geben Sie *alle* möglichen Reduktionen von `double(double 3)` an.

Aufgabe 2 4 Punkte

In der Vorlesung wurde die Fibonacci-Funktion intuitiv und mittels Akkumulatortechnik definiert.
Berechnen Sie `fib1`/`fib2` `5` und `fib1`/`fib2` `20` und geben Sie die Zahl der benötigten Reduktionsschritte (entspr. Zeitbedarf) und der benötigten Zellen (entspr. Platzbedarf) an. Hugs gibt beides nach Eingabe der Anweisung '`:set +s`' aus.
Bestimmen Sie anhand der benötigten Reduktionsschritte die Zeitkomplexitäten der Implementierungen (`fib1`/`fib2` liegt in $O(??)$). Die einfachste Methode ist eine graphische Auswertung.

Aufgabe 3 7 Punkte

Programmieren Sie folgende Funktionen in Haskell und geben Sie jeweils auch den Typ der definierten Funktionen an:

a) `ggT` zur Berechnung des größten gemeinsamen Teilers zweier ganzer Zahlen. Am einfachsten ist eine Implementierung des Euklidischen Algorithmus.
Beispiel: $ggT(66, 24) = 6$ mit der Herleitung:

$$\begin{aligned} 66 &= 2 \cdot 24 + 18 \\ 24 &= 1 \cdot 18 + \ 6 \\ 18 &= 3 \cdot \ \ 6 + \ 0 \end{aligned}$$

b) `kgV` zur Berechnung des kleinsten gemeinsamen Vielfachen zweier Zahlen.
Nutzen Sie folgenden Zusammenhang zwischen ggT und kgV für Ihre Implementierung:

$$ggT(a, b) \cdot kgV(a, b) = ab$$

c) Definieren Sie Funktionen `ggTL` und `kgVL`, welche die ggT/kgV-Berechnung auf Listen von Zahlen (also beliebig viele Zahlen) erweitern.
Nutzen Sie hierbei aus, dass ggT und kgV kommutativ und assoziativ sind und sich die Berechnung für drei Zahlen wie folgt auf die Berechnung für zwei Zahlen zurückführen lässt.

$$\begin{aligned} ggT(m, ggT(n, o)) &= ggT(ggT(m, n), o) = ggT(m, n, o) \\ kgV(m, kgV(n, o)) &= kgV(kgV(m, n), o) = kgV(m, n, o) \end{aligned}$$

Aufgabe 4 7 Punkte

Implementieren Sie einen algebraischen Datentypen `Tree`, welcher binäre Bäume repräsentiert, die in den Knoten Zahlen enthalten.
Definieren Sie folgende Funktionen über dem Datentyp `Tree`:

a) `sumTree` zur Berechnung der Summe aller Zahlen des Baumes. Geben Sie eine zweite Definition von `sumTree` an, welche die Summe in einem zusätzlichen Akkumulator-Argument berechnet. Ist eine der beiden Implementierungen effizienter?

b) `mirrorTree` zum spiegeln eines Baumes (an einem vertikalen Spiegel).

c) `treeToList` zur Umwandlung eines Baum in eine Liste aller Elemente des Baums. Welche (natürlichen) Möglichkeiten gibt es die Liste aufzubauen? Programmieren Sie eine Variante für jede Möglichkeit und beschreiben Sie umgangssprachlich den Aufbau der sich ergebenden Liste.