Compilerbau **Übungsblatt 1**

DHBW Campus Horb **Besprechungstermin: 17. März 2021**

Prüfungsleistung

Die Prüfungsleistung Compilerbau besteht aus zwei Teilen:

- **Vorlesung/Übung:** In 1er/2er Teams Bearbeiten einer Prüfungsaufgabe (10%)

  Auf jedem Übungsblatt werden ein bis zwei Prüfungsaufgaben ausgezeichnet. Jedes Team muss in Verlaufe der Vorlesung eine Prüfungsaufgabe präsentieren. Dabei kann jede Prüfungsaufgabe **NUR von einem 2er Team** präsentiert und auch nur für dieses bewertet werden. Daher empfiehlt es sich in der Übungsgruppe zu besprechen wer welche Aufgabe präsentiert.

  Sollten dennoch mehrere Teams die jeweilige Aufgabe einreichen lost der Dozent, wer die Aufgabe vorführen darf.

- **Labor:** In 7er/8er Teams: Erstellen eines Java-Comilers (90%)

1.1 Logische Operationen

a) Geben Sie in Haskell Definitionen für folgende logische Operatoren an:

```
and, or, xor :: (Bool, Bool) -> Bool
neg :: Bool -> Bool
```

b) Programmieren Sie mithilfe der Operatoren Funktionen, die einem Halb– bzw. Volladdierer entsprechen.

1.2 Symbolisches Differenzieren

a) Schreiben Sie einen Datentyp `Term` der (gebrochen)rationale Funktionen in einer Variablen repräsentiert.

   Hinweis: Sie benötigen jeweils einen Konstruktor für *Monome*, *Addition*, *Multiplikation* und *Division*.

b) Programmieren Sie eine Funktion `diff: Term -> Term`, welche die erste Ableitung bestimmt.

   Hinweis: Sie benötigen für jeden Konstruktor des Typs `Term` eine Gleichung für die Funktion `diff`.

1.3 `curry3`, `uncurry3`

Schreiben Sie die Funktionen

```
curry3 :: ((a, b, c) -> d) -> (a -> b -> c -> d)
uncurry3 :: (a -> b -> c -> d) -> ((a, b, c) -> d)
```

die die jeweiligen Funktionen mit dem Tupel Typ–Konstruktor in äquivalente Funktionen mit dem `->` Typ–Konstruktor umwandeln und umgekehrt.

1.4 Die Funktion `map`

Schreiben Sie eine Funktion, `toListofLists :: [a] -> [[a]]`, die jedes Element einer Liste zu einer einelementigen Liste macht. Nutzen Sie dazu die Funktion `map`.

1.5 Die Funktionen `foldl` und `foldr`

Programmieren Sie eine Funktion `listStringConcat :: [String] -> String`, die alle Elemente einer `String`–Liste aneinander hängt.

Nutzen Sie dazu entweder die Funktion `foldl` oder `foldr`.

Erhalten Sie unterschiedliche Ergebnisse?

1.6 Abstrakter Datentyp `BBaum` (Prüfungsaufgabe 1, 10 Punkte)

a) Programmieren Sie den polymorphen abstrakten Datentyp `BBaum a`. `BBaum a` steht für einen Binärbaum und soll zwei Konstruktoren enthalten. Einen Konstruktor für einen leeren Binärbaum und einen Konstruktor der zwei Teilbäume und ein Element zu einem neuen Binärbaum zusammenfügt.

b) Schreiben Sie eine Funktion `mapTree :: BBaum a -> (a -> b) -> BBaum b`. Der Funktionsaufruf `mapTree b f` soll auf jedes Element des Eingabebaums `b` die Funktion `f` anwenden.

c) Programmieren Sie in Java eine Klassenstruktur für den abstrakten Datentyp `BBaum a` und implementieren Sie die Funktion `mapTree`. Verwenden Sie dazu **KEINE** Lambda–Ausdrücke.