9. Praktikum "Algorithmen und Programmierung II"

SS 2014

Vorbemerkungen. Auf der Web-Seite `http://www.gm.fh-koeln.de/ehses/ap` finden Sie alle nötigen Hilfsdateien.

Dieses Praktikum ist eine Variante des Praktikums 8. Anstelle der Codegenerierung wird hier der erzeugte Syntaxbaum direkt interpretiert. Es gibt zwei Baumoperationen: Darstellung des gespeicherten Ausdrucks durch einen String und Berechnung des Ausdruckswertes. Beides geschieht durch rekursive Aufrufe..

Aufgabe 1) (Baumoperationen) Vervollständigen Sie die Methode `value()` in den Klassen des Paketes `tree` (da wo als TODO angemerkt). Vervollständigen Sie entsprechend die Methode `toString()`. Die Klassen `IdNode` und `LetNode` dienen dem Zugriff auf und dem Speichern in Variablen. Beispiele für die Syntax sind:

```
let a = 3 * 4
let b = 2 + x
b + 1
```

Beachten Sie dabei auch den in `tree.TreeTest` angegebenen JUnit-Test.

Aufgabe 2) (Hashing) Die Klasse `util.Hashing` ist an den angegebenen Stellen zu ergänzen. Es geht nur um ein paar Zeilen, aber dafür muss man halt die Methode der direkten Verkettung (*bucket sort*) verstanden haben. Auch hier ist der JUnit-Test wieder eine Unterstützung.

Am Ende soll das Programm `Calculator.java` nicht nur die korrekten Ergebnisse berechnen, sondern auch die eingegebenen Zeile wieder in die (beinahe) Originalform aus dem Baum zurückübersetzt darstellen.

Frage 1: Welche Vorteile bietet Hashing zu der im Praktikum 3 programmierten `LinearMap` ?

Frage 2: Wann ist es sinnvoll Hashing einzusetzen und wann nicht? Welche Alternativen gibt es?

Anregungen für Interessierte: **(Builder-Pattern, Interpreter, Syntaxanalyse, Compiler)**
Die Klasse `main.Interpreter` zeigt, wie direkt während der Syntaxanalyse der Wert eines Ausdrucks berechnet werden kann. Was müssen Sie in `Calculator.main` ändern um das zu testen?

Natürlich können Sie die Baumklassen aus Praktikum 8) auch um das Speichern in Variablen erweitern. Entsprechend können Sie dann die `OpCode`'s und die Klasse `VM` so erweitern, dass auch der compilierte Code mit Variablen umgehen kann.

Die in den Praktika 8 und 9 verwendete Methode der Syntaxanalyse trägt den Namen „Rekursiver Abstieg“. Wenn Sie Theoretische Informatik verstanden haben, sollten Sie erkennen, dass die Methoden der Klasse `Parser` für Produktionsregeln einer kontextfreien Grammatik stehen (Methode = Nonterminal der linken Seite, Entscheidungen zwischen Regeln sind durch `lookahead`, Terminalsymbole durch `match` dargestellt und Nonterminalsymbols der rechten Seite der Regel sind Aufrufe (`nextOf` stellt eine Abkürzung dar), Verstanden? `~(:-)`

Die Klasse `Scanner` sorgt dafür, dass die einzelnen Wörter eines Programmtexts erkannt werden (Bezeichner, Zahlen, Sonderzeichen). Diese kann man auch durch reguläre Ausdrücke beschreiben.