Praktikum 4 Compilerbau WS14/15 Testat bis 02.12.2014

Ziele:
- Übersetzung von Mini-Java-Exp-Programmen in M32-Assembler als Vorbereitung der Übersetzung von vollständigem Mini-Java.
- Erweiterung der Symboltabellen-Implementierung auf Variablen.

Mini-Java-Exp:
Die Sprache Mini-Java-Exp ergibt sich aus der Definition der Sprache Mini-Java, indem man im Syntaxdiagramm für *statement* von Mini-Java nur den Fall `print(expression);` zulässt und das Syntaxdiagramm für *condition* löscht
(siehe Folie "Zwischencodeerz. 6").
Somit ist folgendes Programm ein gültiges Mini-Java-Exp-Programm:

```
final int c=2;
int x = 17, y = 23;
print(c*x+y*2);
```

Aufgabe 1: (Programmieren in Mini-Java-Exp)
Schreiben Sie ein Programm in Mini-Java-Exp, das die Summe einer Konstanten und einer Variablen hoch 3 ausgibt.
Übersetzen Sie das Programm mit dem Mini-Java-Compiler in M32-Assembler und führen Sie es schrittweise aus.

Aufgabe 2: (Übersetzen von Mini-Java-Exp-Programmen in M32-Assembler)
Implementieren Sie die Übersetzung von Mini-Java-Exp-Programmen in M32-Assemblercode. Verwenden Sie hierzu den Parser aus Versuch 3 Aufgabe 2, wobei Sie auf jeden Fall die vorgegebene Standardgrammatik aus der Musterlösung verwenden, und beschränken Sie ihn auf Mini-Java-Exp. Ersetzen Sie in der JavaCC-Konfigurationsdatei die Transformation in Postfix-Notation durch Konstrukte zur Codeerzeugung.
Gehen Sie hierzu analog zu der Übersetzung nach Postfix aus Versuch 3 Aufgabe 1 und der in der Vorlesung beschriebenen Übersetzung vor. Insbesondere muss die Auswertung des arithmetischen Ausdrucks analog zur Vorlesung auf dem Datenkeller durchgeführt werden!
Verwenden Sie zur Auswertung des Ausdrucks auf dem Datenkeller die in Übungsaufgabe 36 beschriebene Implementierung der Operationen.
Da nach Auswertung eines Ausdrucks dessen Wert auf dem Datenkeller steht, ist die `print`-Anweisung im Gegensatz zur Implementierung der Vorlesung, in der diese Anweisung gar nichts bewirkt, nach `PRN TOS` zu übersetzen, womit man den Wert auf der M32-Assemblerausgabe erhält.
Erweitern Sie die Implementierung der Symboltabelle so, dass neben den Konstanten jetzt auch Variablen zusammen mit dem referenzierten Speicherplatz eingetragen werden können. Stellen Sie hierzu zusätzlich folgende Methode zur Verfügung:

- `void addVariable`, die eine Variable mit dem referenzierten Speicherplatz in die Symboltabelle einträgt, falls die Variable noch nicht vorhanden ist, und sonst eine `SymbolAlreadyDefinedException` wirft.
  Außerdem stellt die Methode sicher, dass der referenzierte Speicherplatz mit dem Wert aus der Zuweisung initialisiert wird und bei einer leeren Zuweisung mit 0.

Schreiben Sie mindestens 4 unterschiedliche Mini-Java-Exp-Programme, übersetzen Sie diese mit Ihrem Übersetzer und überprüfen Sie das Ergebnis mit dem m32-Simulator auf Korrektheit.

Hinweise:
Sie können im M32-Assembler symbolische Adressen verwenden. Da diese in einem Extra-Block deklariert werden müssen, müssen Sie ggf. innerhalb der Konfigurationsdatei 2 nicht zusammenhängende Codestücke erzeugen.
Im Unterschied zur Symboltabelle der Vorlesung tragen Sie hierzu ggf. schon M32-Assemblercode in die Symboltabelle ein und nicht nur den Speicherplatz der Variablen.