PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen
M. Dieterle

D-35032 Marburg
Hans Meerwein Straße

19.Dezember 2007

10. Übung zu „Grundlagen des Compilerbaus“, WS 2007/08

Abgabe der Aufgaben: Mi, 16.Januar 2008 (vor der Vorlesung)

**Frohe Weihnachten
und alles Gute in 2008**

Hinweis: Die Bearbeitungszeit für dieses Blatt ist verlängert und erstreckt sich über die Weihnachtspause. Daher sind statt 12 Punkten 18 Punkte erreichbar. Zudem ist eine Zusatzaufgabe enthalten, um einen Ausgleich von Punktrückständen bzw. die Präsentation weiterer Aufgaben zu ermöglichen.
Im Tutorium am 11. Januar können Fragen zu den Aufgaben geklärt werden.

10.1 **PSA-Übersetzung mit Attributgrammatik** 10 Punkte

Die auf der nächsten Seite beschriebene While-Sprache ist der Sprache PSA sehr ähnlich. Die Ausgabe eines While-Parsers kann als abstrakter Syntaxbaum eines PSA-Programms betrachtet werden.

(a) Skizzieren Sie eine L-Attributierung der obigen Grammatik, welche in einem synthetischen Attribut der Wurzel die Übersetzung des beschriebenen Programms in MA-Code erzeugt. Sie benötigen als weitere Attribute mindestens eine Symboltabelle sowie die Codelänge (siehe Aufgabe 9.3).

(b) Schreiben Sie einen Parser mit Happy, welcher an Stelle eines abstrakten Syntaxbaumes MA-Code in einem Attribut erzeugt. Auf der VL-Seite finden Sie einen While-Scanner, der die Eingabe für den Parser liefert.

(c) Realisieren Sie durch eine Attributierung eine Definiertheits- und Typprüfung (siehe Aufgabe 9.1).

Auf der Webseite zur Vorlesung finden Sie eine Implementierung der MA-Maschine, mit der Sie den generierten MA-Code ausführen können. Testen Sie, ob Ihr Compiler korrekten Code erzeugt.

Für eine kleine WHILE-Programmiersprache seien folgende Operationen definiert:

- Arithmetische Operatoren (ArithOp): + − * /
- Relationale Operatoren (RelOp): = != < <= > >=

Die Syntax der Sprache wird durch die folgende Grammatik beschrieben:

$$
\begin{array}{lcl}
\textit{program} & \rightarrow & \textbf{program}\ \text{Id '\{'}\ \textit{dec stmts}\ \text{'\}'} \\
\textit{dec} & \rightarrow & \textbf{var}\ \textit{varlist} \\
\textit{varlist} & \rightarrow & \text{Id}\ B \\
B & \rightarrow & \text{',' Id}\ B \mid \varepsilon \\
\textit{stmt} & \rightarrow & \text{Id ':='}\ \textit{expr} \\
 & \mid & \textbf{if}\ \textit{cond}\ \textbf{then}\ \textit{stmt} \\
 & \mid & \textbf{while}\ \textit{cond}\ \textbf{do}\ \textit{stmt} \\
 & \mid & \text{'\{'}\ \textit{stmts}\ \text{'\}'} \\
\textit{stmts} & \rightarrow & \textit{stmt}\ C \\
C & \rightarrow & \text{';'}\ \textit{stmts} \mid \varepsilon \\
\textit{cond} & \rightarrow & \textit{expr}\ \text{RelOp}\ \textit{expr}\ \text{D} \\
D & \rightarrow & \text{'\&'}\ \textit{cond} \mid \text{'|'}\ \textit{cond} \mid \varepsilon \\
\textit{expr} & \rightarrow & \text{(Analog zum Beispiel } G'_{AE} \text{ (S.37) der Vorlesung)}
\end{array}
$$

Die Sprache kann außerdem Zeilenkommentare erlauben, welche mit '//' eingeleitet werden.

10.2 **Aktivierungsblöcke in PSP** 4 Punkte

(a) Gegeben sei der folgende *MP*-Zustand

$$(5, 4, 8 : 3 : 10 : 7 : 4 : 3 : 27 : 9 : 8 : 3 : 54 : 1 : ...) \in ZR$$

Ermitteln Sie den Zustand nach Ausführung der nächsten vier Befehle des folgenden *MP*-Codes:

```
      ⋮
 5: LOAD(1,1)
 6: ADD
 7: RET
 8: LOAD(0,2)
 9: STORE(2,1)
10: CALL (84,1,3);
11: RET
      ⋮
```

(b) Welche der folgenden Prozedurkeller können das Ergebnis der Ausführung eines übersetzten PSP-Programms sein?

i. `15:4:9:-3:5:5:6:3:2:1:5:4:12:3:4:4:3:25:0:0:0:0:12`
ii. `15:4:9:-3:5:10:4:3:2:1:5:4:12:3:4:4:3:25:0:0:0:0:12`
iii. `15:4:9:-3:5:14:4:3:2:1:5:4:12:3:4:4:3:25:0:0:0:0:12`

10.3 PSP-Erweiterung: Funktionsaufrufe

4 Punkte

Häufig bieten prozedurale Sprachen neben (parameterlosen) Prozeduren auch die Möglichkeit, Funktionsprozeduren zu verwenden, deren Resultat als Ausdruck behandelt wird. Eine entsprechende Erweiterung der Sprache PSP könnte wie folgt aussehen:

$$\begin{aligned}
\Delta &::= \Delta_C \Delta_V \Delta_p \Delta_F \\
\Delta_F &::= \varepsilon \mid \textbf{fun } I_1; B_1; \ldots I_n; B_n; \\
E &::= \ldots \mid I() \\
\Gamma &::= \ldots \mid \textbf{return } E
\end{aligned}$$

Funktionsprozeduren werden wie parameterlose Prozeduren, aber mit **fun** statt **proc** deklariert. Ihr Aufruf ist innerhalb von Ausdrücken erlaubt. Die neue Anweisung **return** beendet den Aufruf und liefert gleichzeitig das Ergebnis der Funktion zurück.

Geben Sie entsprechende Erweiterungen der Übersetzungsfunktionen up, dt, ct und et an.

10.4 Zusatzaufgabe

9 Punkte

Implementieren Sie Aufgabe 10.1 b) und c) mit Parsec und vergleichen Sie die Laufzeit beider Programme.