Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion
CAU Kiel, Institut für Informatik
Prof. Dr. M. Hanus

24118 Kiel
Christan-Albrechts-Platz 4
Tel.: 0431 / 880-7271

11. Übung „Übersetzerbau“
Bearbeitung bis zum 1. Juli 2008

Bitte senden Sie die Programmieraufgaben zusätzlich zur Abgabe in ausgedruckter Form auch per Email an Axel Stronzik (`axs@informatik.uni-kiel.de`)!

Aufgabe 32
Bei der bisherigen Zwischencodeerzeugung wurden in der Vorlesung neue Marken mittels

`newlabel`/`extendlabel` und `newtemp`/`extendtemp`

erzeugt. Hierbei sind Marken Strings, die bei größeren Programmen natürlich recht groß werden können.
Als effizientere Alternative können `Int`s als Marken verwendet werden:

```
data Label = Label Int
data Temp = Temp Int
```

Diese können dann in der Berechnung durchgereicht (ähnlich wie bei der Implementierung L-attributierter Grammatiken) und bei Bedarf verwendet werden. Hierbei sollte darauf geachtet werden, dass möglichst kleine Marken verwendet werden.
Geben Sie entsprechende Übersetzungen für Zuweisung, `if-then-else`, `repeat-until` und `for`-Schleife[1] an.

Aufgabe 33
Wir erweitern die Bedingungen des Programms aus Aufgabe 30 wie folgt:

```
var x,n : Int;
x := 7;
n := 0;
while x>4 && n>=0 do
  n := n+1;
  if x mod 2 = 0 && x>=0
    then x := x div 2
    else x := 3 * x + 1
od;
print(n)
```

Passen Sie zunächst den erzeugten Zwischencode an. Verwenden Sie das in der Vorlesung vorgestellte Verfahren, um aus dem entstehenden Zwischencode die Basisblockdarstellung zu gewinnen.

[1] mit Pascal-Semantik: `for` i:=n `to` k `step` d `do` wobei $i, n, k, d \in$`Int` und i der Reihe nach die Werte $n, n+d, n+2d, \ldots$ annimmt. Die Schleife terminiert, falls der Wert der Zählvariablen i größer als k ist.

Aufgabe 34

Die Übersetzung von Prozeduren, welche sich wechselseitig aufrufen, ist nicht mit Hilfe einer L-Attributierung möglich. Aus diesem Grund wurden in älteren Programmiersprachen wie Pascal `forward`-Deklarationen verwendet. In dieser Aufgabe soll eine Übersetzung ohne solche Deklarationen realisiert werden.

Als vereinfachtes Modell von Prozeduren und Prozeduraufrufen betrachten wir folgende Sprache:

$$
\begin{aligned}
Prg &\longrightarrow Proc\ ;\ Prg \mid \varepsilon \\
Proc &\longrightarrow \textbf{proc}\ Id\ Stm \\
Stm &\longrightarrow \textbf{call}\ Id\ Stm \mid \varepsilon \\
Id &\longrightarrow \textbf{a} \mid \ldots \mid \textbf{z}
\end{aligned}
$$

Wir betrachten als Vereinfachung nur parameterlose Prozeduren, deren Rumpf nur den Aufruf einer anderen Prozedur ermöglicht. Implementieren Sie einen Compiler, welcher Code für ein gegebenes Programm berechnet. Der erzeugte Code bestehe nur aus einer Folge von Programmadressen und `call/return`-Befehlen, wie in folgendem Beispiel:

proc a; proc c call a call b; proc b call c liefert den Code:

```
0: return
1: call 0
2: call 4
3: return
4: call 1
5: return
```

Verwenden Sie also keine symbolischen Adressen, sondern erzeugen Sie direkt absolute Programmadressen.