Prof. Dr. Gerhard Reinelt
Dipl.-Math. Achim Hildenbrandt
Institut für Informatik
Universität Heidelberg

Vorlesung "Compilerbau" WS 2011/2012

Weihnachtsübungsblatt

Abgabe: 25. Januar 2012 bis 10.45 Uhr

Aufgabe 1 (12 Punkte)

Die unten angegebene Grammatik definiert **tiny**, eine einfache Sprache für Deklarationen und Zuweisungen, welche als einzigen Datentyp Integer-Zahlen verarbeitet.

$P \rightarrow$ `program id ;` V `begin` B `end`
$V \rightarrow$ `var id` D `;` $\mid \varepsilon$
$D \rightarrow$ `,` `id` $D \mid \varepsilon$
$B \rightarrow S\ R \mid \varepsilon$
$R \rightarrow$ `;` $S\ R \mid \varepsilon$
$S \rightarrow$ `id :=` $E \mid$ `input id` $\mid$ `print id`
$E \rightarrow E$ `+` $T \mid E$ `-` $T \mid T$
$T \rightarrow T$ `*` $F \mid T$ `/` $F \mid F$
$F \rightarrow$ `(` E `)` $\mid$ `id` $\mid$ `intconst`

Die Terminalsymbole `id` und `intconst` bezeichnen Identifier bzw. Integer-Konstanten.
Erstellen Sie mit Hilfe von `lex` und `yacc` einen Compiler, der die Sprache **tiny** in eine Assemblersprache übersetzt, welche die folgenden Merkmale hat:

- Alle Berechnungen finden in einem einzigen Register statt, dem Akkumulator.
- Zwischenergebnisse müssen im Speicher abgelegt werden.
- Den Speicher fassen wir als genügend großes Array von Integer-Zahlen `mem[0]`, `mem[1]`, ... auf.
- In den ersten Speicherzellen `mem[0]`, `mem[1]`, ..., `mem[`$m-1$`]` werden jeweils die m im Programm deklarierten Variablen verwaltet. Der Bereich ab `mem[`m`]` steht für Zwischenergebnisse zur Verfügung.

Folgende Befehle der Assemblersprache stehen zur Verfügung:

`read`	liest eine Zahl aus der Eingabe in den Akkumulator ein.
`write`	gibt den Inhalt des Akkumulators auf der Ausgabe aus.
`load` n	kopiert den Inhalt der Speicherzelle `mem[`n`]` in den Akkumulator.
`loadc` n	schreibt den Wert n in den Akkumulator.
`store` n	kopiert den Inhalt des Akkumulators in die Speicherzelle `mem[`n`]`.
`add` n	addiert den Inhalt der Speicherzelle `mem[`n`]` zum Inhalt des Akkumulators.
`addc` n	addiert den Wert n zum Inhalt des Akkumulators.

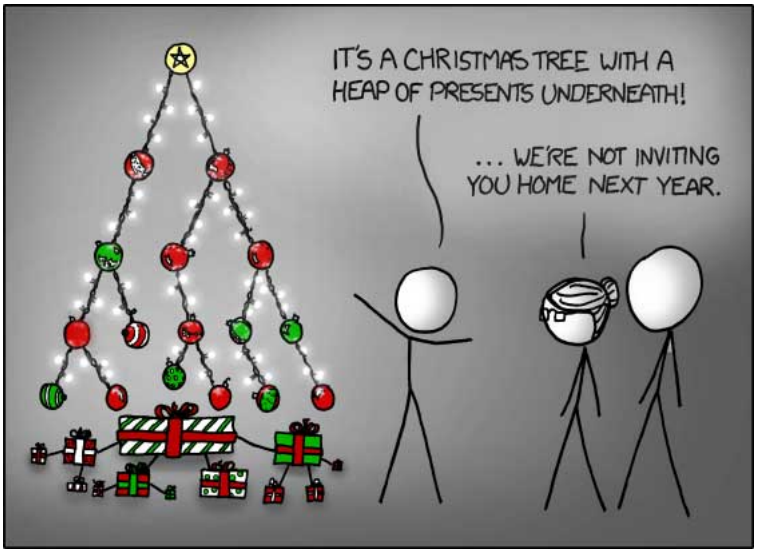
Für die Subtraktion, Multiplikation und Division gibt es zur Addition analoge Befehle `sub`, `subc`, `mul`, `mulc`, `div` und `divc`.
Das Beispielprogramm

```
program test;
var a,b;
begin
  input a;
  input b;
  a := a*b-a/b;
  print a
end
```

sollte in den Assembler Code

```
read
store 0
read
store 1
load 0
mul 1
store 2
load 0
div 1
store 3
load 2
sub 3
store 0
load 0
write
```

übersetzt werden.



Wir wünschen Ihnen ein besinnliches Weihnachtsfest und ein guten Start in das Jahr 2012