ÜBUNG 3

calc mit lex und yacc

In dieser Übung werden sie ein vorgegebenes Programm **calc** selbständig erweitern. calc dient dazu, arithmetische Ausdrücke die sie in Textform eingeben, auszuwerten. Das vorgegebene Programmgerüst kennt ganze Zahlen (Integers), die Addition **+**, die Multiplikation *****, den Zuweisungsoperator **:=** und Variablen die aus einem Buchstaben (a-z) bestehen. Arbeiten sie in Zweiergruppen.

Programm kompilieren und starten

Entpacken sie die Datei calc.tgz und wechseln sie auf der Kommandozeile ins Verzeichnis calc

```
> tar xvfz calc.tgz
> cd calc
```

Um das Programm zu kompilieren verwenden sie das Kommando **make**

```
> make
```

Wurden keine Fehlermeldungen angezeigt, dann können sie das Programm nun mittels **calc** starten, und arithmetische Ausdrücke eingeben, die nach jedem <ENTER> ausgewertet werden, z.B:

```
> ./calc
1+2
3
a:=5
7*a+9
44
```

Um das Programm zu beenden drücken sie <Strg>-<C>.

calc.l

Das File calc.l enthält die Definition für die lexikalische Analyse, d.h. das Zerlegen einer Eingabe in Token (z.B. Zahlen, Identifier, Operatoren, etc.). Das Tool lex erzeugt aus dieser abstrakten Definition ein passendes C-Programm.
Das File besteht im Wesentlichen aus regulären Ausdrücken gefolgt von einem Block mit C-Code, der immer dann ausgeführt wird, wenn der reguläre Ausdruck im Input-Text gematcht wird.

calc.y

Das File calc.y enthält die Definition für die syntaktische Analyse, d.h. das Zusammensetzen der Token aus der lexikalischen Analyse, zu sinnvollen Sprachkonstrukten.

Aufgabe A.1 (1): Operatoren für Subtraktion und Division und Modulo

Fügen sie analog zu den bereits vorhanden Operatoren + und *, die Operatoren –, / und % (modulo) hinzu. Sie müssen dazu in calc.l neue Token definieren und in calc.y passende Regeln hinzufügen.

Aufgabe A.2 (1): Klammer Operator

Erweitern sie den Parser um die Möglichkeit geklammerte Expressions auszuwerten, z.B.: 3*(1+3)
Definieren sie Token für die Klammern und fügen sie eine passende Regel hinzu.

Hinweis: ein Klammerausdruck ist eine beliebige Expression die von Klammern umschlossen ist: <expression> ::= **(** <expression> **)**

Aufgabe A.3 (1): Unäre Operatoren + und -

Erweitern sie den Parser um die unären + und - Operatoren.
z.B.: +3--5 ergibt 8

Aufgabe A.4 (1): Relationale Operatoren

Erweitern sie den Parser um die relationalen Operatoren <, <=, ==, !=, >= und >.
Der Wert eines solchen Ausdrucks soll 0 für FALSE und 1 für TRUE sein (analog zu C/C++). Z.B.: 3 < 5 ergibt 1, 7 == 3 ergibt 0, etc.

Aufgabe A.5 (2): Minimum und Maximum

Erweitern sie den Parser um Operatoren die das Minimum bzw. Maximum zweier Expressions zurückliefern:

```
min(a, b)
max(a, b)
```

Beispiele:

`min(5, 3)`	ergibt: 3
`max(2*2*12, 19+31)`	ergibt: 50

Aufgabe A.6 (2): Minimum und Maximum beliebig vieler Werte

Erweitern sie die Operatoren min und max so, dass sie beliebig viele Parameter verarbeiten können:

Beispiele:

`min(5, 3, 0, 17)`	ergibt: 0
`max(17, 4*4, 19+31, 7>3, 100/2)`	ergibt: 50
`min(123)`	ergibt: 123
`max()`	Fehler!
`min()`	Fehler!

Aufgabe A.7 (2): Conditional Operator

Erweitern sie den Parser um den conditional (ternary) Operator den sie aus der Programmiersprache C kennen:

```
cond ? x : y
```

Der Wert ist x wenn cond TRUE (ungleich 0) ist und y wenn cond FALSE ist.

Aufgabe A.8 (2): Hex

Erweitern sie den Parser um den die Möglichkeit, Integer auch als Hexadezimalwerte angeben zu können. Verwenden Sie dazu das Prefix 0x wie in der Programmiersprache C. (A-F kann groß oder klein geschrieben werden). Die Ergebnisse sollen weiterhin im Dezimalsystem ausgegeben werden.
Beispiel:

```
> ./calc
0x5+0xF
20
0xa*0xb
110
```

Abgabe

Abzugeben ist ein tgz File mit den modifizierten lex und yacc Files, dem Makefile und dem ausführbaren Programm.

Die Abgabe muss beim zweiten Code Review präsentiert werden, es können für diese Übung maximal 12 Punkte erreicht werden. Erzeugen Sie keine zusätzlichen Ausgaben, das Programm wird auch mit Testskripts automatisiert überprüft.