Fachrichtung 6.2 — Informatik
Universität des Saarlandes
Tutorenteam der Vorlesung Programmierung 1

Programmierung 1 (Wintersemester 2012/13)

Aufgaben aus den Übungsgruppen 12

(Kapitel 13)

Hinweis: Dieses Übungsblatt enthält von den Tutoren für die Übungsgruppe erstellte Aufgaben.

Die Aufgaben und die damit abgedeckten Themenbereiche sind für die Klausur weder relevant, noch irrelevant.

1 Überschrift

Aufgabe 12.1 (*Viele schöne Begriffe*)
Finden Sie kurze und prägnante Beschreibungen für die folgenden Begriffe:

(a) Parser

(b) lexikalische Syntax

(c) syntaktische Kategorie

(d) maximum munch rule

(e) Lexer

(f) Eindeutigkeit

(g) phrasale Syntax

(h) abstrakte Syntax

(i) konkrete Grammatik

(j) Ableitung

(k) Prüfer

(l) RA-tauglich

(m) konkrete Syntax

(n) abstrakte Grammatik

(o) Metavariable

(p) Syntaxbaum

(q) Affinität

Finden Sie Parallelen und stellen Sie Beziehungen zwischen den Begriffen auf.

Aufgabe 12.2
Überzeugen Sie sich, dass Sie mit Hilfe von Hilfskategorien tatsächlich linksklammerende Ausdrücke erhalten.
Sei dazu die folgende Grammatik und der folgende dazugehörige Parser gegeben:

plusexp	::=	[*plusexp* "+"] *unminusexp*
unminusexp	::=	["-"] *pexp*
pexp	::	"1" | "("*plusexp*")"

Der Parser benutzt die folgenden beiden Datentypen:

```
datatype token = OpPlus | OpMinus | RPAR | LPAR | EINS
datatype exp = Plus of exp * exp | Minus of exp | ConEins
```

```
fun plusexp ts = plusexp' (unminusexp ts)

and plusexp' (a, OpPlus::tr) = plusexp' (extend (a, tr) unminusexp Plus)
| plusexp' s = s

and unminusexp (OpMinus::tr) = let val (a, tr') = pexp tr in (Minus a, tr') end
| unminusexp ts = pexp ts

and pexp (EINS::tr) = (ConEins, tr)
| pexp (LPAR::tr) = match (plusexp tr) RPAR
| pexp s = raise Error ''pexp''
```

(a) Die obige Grammatik ist nicht RA-tauglich. Wandeln Sie diese in eine RA-taugliche Grammatik um, die die gleichen Ausdrücke akzeptiert!

(b) Parsen Sie nun gedanklich folgende Liste, indem Sie jeden Aufruf notieren bis Sie auf das Ergebnis kommen: $[EINS, OpPlus, EINS, OpPlus, EINS]$. Ihr Anfang sollte also so aussehen:

$plusexp'(unminusexp[EINS, OpPlus, EINS, OpPlus, EINS])$
$= plusexp'(pexp[EINS, OpPlus, EINS, OpPlus, EINS])$
$= \ldots$

(c) Betrachten Sie erneut die Grammatik und den darauf beruhenden Parser. Was fällt Ihnen auf? Basiert der Parser wirklich exakt auf der obigen Grammatik? Erklären Sie!

Aufgabe 12.3
Stellen Sie eine Grammatik für die wie folgt definierten Ausdrücke auf:

- Es gibt die Operatoren A und B.
- A ist ein unärer Operator. Ist φ ein gültiger Ausdruck, so ist es auch $A\varphi$.
- B ist ein binärer Operator. Sind φ und ψ gültige Ausdrücke, so ist es auch $\varphi B\psi$.
- A klammert stärker als B.
- B klammert implizit rechts.
- 1 ist immer ein gültiger Ausdruck.
- Ausdrücke dürfen Klammern enthalten, d.h. ist φ ein gültiger Ausdruck, so ist es auch (φ)

Schreiben Sie dann einen Parser für Ihre Grammatik mit dem folgenden Datentypen *exp* und den angegebenen Tokens:

```
datatype token = OpA | OpB | Eins
datatype exp = A of exp | B of exp * exp | ConEins
```

Aufgabe 12.4
Bearbeiten Sie Aufgabe 12.3 erneut, dabei soll jedoch der Operator B implizit links klammern.
Hinweis: Bevor Sie einen Parser schreiben können, müssen Sie Ihre Grammatik in eine RA-taugliche umwandeln.