Karlsruher Institut für Technologie
Lehrstuhl für Programmierparadigmen
Sprachtechnologie und Compiler WS 2010/2011
Dozent: Prof. Dr.-Ing. G. Snelting
Übungsleiter: Matthias Braun `matthias.braun@kit.edu`

Übungsblatt 7 Ausgabe: 05.06.2012 Besprechung: 11.06.2012

Aufgabe 1: *Auswertungsreihenfolgen*

1.1 *Eigenschaften*

Geben Sie attributierte Grammatiken die die folgenden Bedingungen erfüllen. Geben Sie zusätzlich zur Verdeutlichung einen beliebigen Parsebaum.

Geben Sie eine AG an, die:

1. Nur synthetisierte Attribute enthält.
2. Nur ererbte Attribute enthält.
3. Ein Attribut enthält das sowohl synthetisiert als auch ererbt wird.
4. Stets mit einem Baumdurchlauf berechenbar ist.
5. Mehr als einen Baumdurchlauf für ihre Berechnung benötigen.
6. Nur für manche Bäume berechenbar ist. Es sollte sowohl berechenbare als auch nicht berechenbar Bäume geben, in denen alle Grammatikproduktionen vorkommen.

Aufgabe 2: *Attributierte Grammatiken und* LL

2.1 *Bedingungen*

Welche Bedingungen müssen gelten damit eine attributierte Grammatik während des LL parsens berechnet werden kann?

2.2 *Auswertung der Attribute beim rekursiven Abstieg*

Gegeben sei folgende attributierte Grammatik:

	Produktion	**Attributierungsregeln**
1)	$E \rightarrow T\ E'$	$E.val = T.val + E'.val$
		$E'.constants = E.constants$
		$T.constants = E.constants$
2)	$E' \rightarrow +\ T\ E'_1$	$E'.val = T.val + E'_1.val$
		$T.constants = E.constants$
		$E'_1.constants = E.constants$
3)	$E' \rightarrow \varepsilon$	$E'.val = 0$
4)	$T \rightarrow F\ T'$	$T.val = F.val * T'.val$
		$T'.constants = T.constants$
		$F.constants = T.constants$
5)	$T' \rightarrow *\ F\ T'_1$	$T'.val = F.val * T'_1.val$
		$F.constants = T'.constants$
		$T'_1.constants = T'.constants$
6)	$T' \rightarrow \varepsilon$	$T'.val = 1$
7)	$F \rightarrow (E)$	$F.val = E.val$
		$E.constants = F.constants$
8)	$F \rightarrow$ **digit**	$F.val =$ **digit**$.lexval$
9)	$F \rightarrow$ **constant**	$F.val = F.constants.getConstantValue($**constant**$.symbol)$

Ergänzen Sie den folgenden rekursiven Abstieg um die Auswertung der Attributierungsregeln.

```
void E() {
    T();
    E'();
}

void E'() {
    switch(t) {
    case '+':
        t = nextSymbol();
        T();
        E'();
        break;
    case ')':
    case EOF:
        break;
    default: Fehler();
    }
}

void T() {
    F();
    T'();
}

void T'() {
    switch(t) {
    case '*':
        t = nextSymbol();
        F();
        T'();
        break;
    case '+':
    case EOF:
        break;
    default: Fehler();
    }
}

void F() {
    switch(t) {
    case '(':
        t = nextSymbol();
        E();
        if (t == ')')
            t = nextSymbol();
        else
            Fehler();
        break;
    case digit:
        break;
    case constant:
        break;
    default: Fehler();
    }
}
```

Aufgabe 3: *Namensanalyse*

```
class Base {
    public int X , Y;
    public int f () {
        return X1 ;
    }
}
class Derived extends Base2 {
    public class Inner extends Base {
        public void func() {
            f3 ();
            X4 = 20;
            Y5 = 10;
        }
        private int Y ;
    }
    public int f () {
        return X6 ;
    }
    public void call_f(Base b ) {
        b7 . f8 ();
        ((Derived) b). f9 ();
    }
    public int X , Y;
}
```

Abbildung 1: Beispiel 1 (Java Code)

```
struct x { int x ; };
x1 k ;
int x ;
void f(void) { x2 = 20; k3 . x4 = 42; }
```

Abbildung 2: Beispiel 2 (C++ Code)

Die meisten Programmiersprachen besitzen geschachtelte Namensräume. Oft finden sich auch voneinander unabhängige Namensräume für verschiedene Programmierkonstrukte. Betrachten sie die Abbildungen 1, 2 und 3.

- Auf welche Definitionen beziehen sich die markierten (Bezeichner mit Subskript x_1) Referenzen? (Die Subskripte selbst sind natürlich nicht Teil des Programms, sondern hier nur zur Verdeutlichung der Aufgabenstellung).
- Beschreiben Sie wo in den Beispiele neue Namensraumschachteln entstehen.
- Gibt es unabhängig Namensräume?

```
extern "C" int rand(void);

int main(void) {
    int foo;

 foo :
    if (rand()) {
        float foo ;

        for (float foo = 0; foo < 42; ++ foo_1 ) {
            if (rand() < 13)
                goto foo_2 ;
        }
         foo_3 = 42;
    }
}
```

Abbildung 3: Beispiel 3 (C++ Code)