FCW 1x Übung zu Formale Sprachen, Compiler- und Werkzeugbau 1 — WS 2016/17, Übung 6

Abgabetermin: in der KW 5 (beim Übungsleiter persönlich)

☐ **Gr. 1,** Dr. H. Dobler **Name** _____ **Aufwand in h** _____
☐ **Gr. 2,** Dr. G. Kronberger

Punkte _____ **Übungsleiter** _____

Weihnachten ist zwar schon vorbei, Sie haben aber trotzdem noch einen Wunsch frei: Sie dürfen sich eine der beiden Aufgaben A oder B (auf der nächsten Seite) aussuchen ;-)

A. MiniCpp, neue Anweisung: *switch* mit *case*(s) (12 + 12 Punkte)

MiniCpp bietet ja schon einiges, aber bei den Anweisungen (bisher gibt es zur Fallunterscheidung nur die binäre Verzweigung mit *if* und optionalem *else*) fehlt die Mehrfachverzweigung. Sie beschließen daher, die Sprache MiniCpp und damit den MiniCpp-Compiler so zu erweitern, dass auch die aus C++ bekannte *switch*-Anweisung mit beliebig vielen *case*-Zweigen und optionalem *default*-Zweig zur Verfügung steht. Gehen Sie bei der Realisierung dieses ambitionierten Vorhabens in zwei Schritten vor:

A.1. Änderungen/Erweiterungen am Frontend und in der internen Repräsentation

Erweitern Sie die Sprache (in der attributierten Grammatik *MiniCpp.atg*)

- um die drei neuen Schlüsselwörter *switch, case* und *default*, sowie
- um neue Grammatikregeln, welche die Syntax der Mehrfachverzweigung nach folgendem Muster definiert:

```
switch ( SimpleExpr ) { // where SimpleExpr is of type int
  case X:  StatSeq1     // where X, Y, … are int literals (aka numbers)
  case Y:  StatSeq2     //   and a break terminates the switch stat
    …
  default: StatSeqN     // optional, needs no break
} // switch
```

Dazu sind auch Änderungen/Erweiterungen bei den semantischen Aktionen der ATG und in der internen Repräsentation (vor allem im Module *AST.cs*) notwendig.

A.2. Änderungen/Erweiterungen in der Codegenerierung (nach der letzten Übung)

Schreiben Sie C#-Codestücke mit Mehrfachverzweigungen, übersetzen Sie diese mit dem C#-Compiler *csc* und studieren Sie den erzeugten Code mittels *ILDAsm*, um die notwendigen CIL-Befehle für den Codegenerator des MiniCpp-Compilers (im Modul *GenCilAsText.cs*) herauszufinden und erweitern Sie dann den Codegenerator des MiniCpp-Compilers entsprechend Ihrer Erkenntnisse.
Wenn Sie dann noch Zeit und Lust haben könnten Sie auch den zweiten Codegenerator (in *GenCilByRefEmit.cs*) entsprechend erweitern.

B. MiniCpp, neuer Datentyp: *double* (12 + 12 Punkte)

MiniCpp bietet ja schon einiges, aber bei den Datentypen (bisher nur *void*, *int* und *bool*) fehlt einer für das „exakte Rechnen". Sie beschließen daher, die Sprache MiniCpp und damit den MiniCpp-Compiler so zu erweitern, dass auch der aus C++ bekannte Datentyp *double* zur Verfügung steht. Gehen Sie bei der Realisierung dieses ambitionierten Vorhabens in zwei Schritten vor:

B.1. Änderungen/Erweiterungen am Frontend und in der internen Repräsentation

Erweitern Sie die Sprache (in der attributierten Grammatik *MiniCpp.atg*) um

- das neue Schlüsselwort *double*,
- eine neue Terminalklasse *dblNumber* für *double*-Literale (nach dem Muster von C++, so sollen z. B. 3.14, 2.71e0 aber auch 1E3 als *double*-Literale möglich sein),
- den neuen Datentyp *double* (in der Regel *Type*),
- *double*-Konstanten (in der Regel *ConstDecl*) und
- skalare *double*-Variablen sowie -Parameter und dynamische *double*-Felder (in der Regel *VarDefOrFuncDef*).

Lassen Sie dann zumindest reine *double*-Zuweisungen und reine *double*-Arithmetik gemäß der folgenden Muster zu (*x* und *y* seien *double*-Variablen oder Elemente von *double*-Feldern):

- *x* = 3.14;
- *x* = *y*;
- ... *x* + *y* ... *x* - *y* ... *x* * *y* ... *x* / *y*.

Für StreberInnen: Lassen Sie auch gemischte *int/double*-Zuweisungen und gemischte *int/double*-Arithmetik zu (*i* sei eine Variable oder ein Element eines *int*-Felds):

- *x* = 1; // but not *i* := 3.14;
- *x* = *i*; // but not *i* = *x*;
- ... *x* + *i* ... *x* - *i* ... *x* * *i* ... *x* / y … sowie ... *i* + *y* ... *i* - *y* ... *i* * *y* ... *i* / *y* …

Ein/Ausgabe wäre natürlich auch nett:

- *cin* >> *x*;
- *cout* << *y*;

Dazu sind Änderungen/Erweiterungen bei den semantischen Aktionen der ATG und in der internen Repräsentation (Module *SymTab.cs* und *AST.cs*), für Ein/Ausgabe auch in *CIL.cs* bzw. *CIL.frm* notwendig.

B.2. Änderungen/Erweiterungen in der Codegenerierung (nach der letzten Übung)

Schreiben Sie C#-Codestücke, übersetzen Sie diese mit dem C#-Compiler *csc* und studieren Sie den erzeugten Code mittels *ILDAsm*, um die notwendigen CIL-Befehle für den Codegenerator des MiniCpp-Compilers (im Modul *GenCilAsText.cs*) herauszufinden. Erweitern Sie dann den Codegenerator des MiniCpp-Compilers entsprechend Ihrer Erkenntnisse.
Wenn Sie dann noch Zeit und Lust haben könnten Sie auch den zweiten Codegenerator (in *GenCilByRefEmit.cs*) entsprechend erweitern.