Prof. Dr. Arnd Poetzsch-Heffter
Dr.-Ing. Jan Schäfer

Technische Universität Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

# Übungsblatt 11: Übersetzer und sprachverarbeitende Werkzeuge (SS 2011)

Hand Out: 6. Juli 2011
Hand In: 13. Juli 2011

## Aufgabe 1 Prüfungsfragen zu den Vorlesungen

Überlegen Sie sich mindestens zwei Prüfungsfragen und tragen Sie diese unter `http://bit.ly/kJORSi` ein.

## Aufgabe 2 Erweiterung von MiniJava um objektorientierte Konstrukte

Erweitern Sie Ihren MiniJava-Dialekt um Unterstützung für Objektorientierung, d.h. um Referenztypen, nicht-statische Member von Klassen und "extends"-Klauseln. Konstrukten können nicht selbst definiert werden. Es gibt immer nur den implizit definierten parameterlosen Konstruktor.

Das Objektlayout soll Folie 20 in Kapitel 3.2 entsprechen.

a) Erweitern Sie Ihren Scanner um die Schlüsselwörter **`new`**, **`null`** und **`extends`**.

b) Erweitern Sie Ihren Parser um folgende Produktionen:

```
Primary ::=
   ...
  | NEW IDENTIFIER IdentifierRest
  | NULL ;
ClassDeclaration ::=
   ...
  | CLASS:c IDENTIFIER:id EXTENDS IDENTIFIER ClassBody:body ;
Type ::= ...
   | IDENTIFIER ;
```

c) Erweitern Sie Ihren AST um folgende Deklarationen und bauen Sie diese entsprechend im Parser auf.

```
Expression = ...
        | Null ( SourcePosition sourcePos )
        | New ( SourcePosition sourcePos, Identifier ident )

Type = ...
    | Identifier

IdentifierOpt = IdentifierNone ()
          | Identifier

ClassDecl ( SourcePosition sourcePos, Identifier ident, IdentifierOpt superClass,
         ClassBodyDecls body )
```

d) Erweitern Sie die Namens- und Typanalyse um die neuen Konstrukte. Nun sind auch nicht-statische Methoden und Felder in Klassen erlaubt. Beachten Sie, dass Sie in statischen Methoden nur statische Methoden und Felder verwenden können.

Nehmen Sie die Existenz einer Klasse `Object` ohne Methoden und Felder an, die Supertyp aller Klassen ist. **`null`** ist zuweisungskompatibel zu jedem Referenztyp.

e) Erweitern Sie den Codegenerator um folgende Punkte. Auf eine vollständige Garbage-Collection können Sie jedoch verzichten.

**Speicherverwaltung** Realisieren Sie eine Speicherverwaltung, die den Heap in einen benutzten und einen unbenutzten Teil aufteilt. Die Grenze zwischen den beiden Teilen wird in einer globalen Variable gespeichert und bei jeder Speicherallokation entsprechend verschoben.

Erzeugen Sie hierfür Code bei jedem New-Expression und initialisieren sie den Speicher des Objekts.

**Methodentabellen** Generieren Sie ein `.data`-Segment in dem für jede Klasse eine Methodentabelle abgelegt ist.

**Methoden** Passen Sie Ihre Codegenerierung auf nicht-statische Methoden an. Führen Sie den impliziten `this`-Parameter ein.

**Methodenaufrufe** Erzeugen Sie Code für Methodenaufrufe. Denken Sie daran, den impliziten Parameter zu übergeben.

**Attributzugriffe** Passen Sie den Code für Variablenzugriffe so an, dass Sie auch auf Objektattribute zugreifen können.

## Aufgabe 3 Sethi–Ullman Algorithmus

a) Überlegen Sie sich wie der Sethi-Ullman-Algorithmus für MIPS aussehen muss und beschreiben Sie die Unterschiede zu dem Algorithmus aus der Vorlesung.

b) (Optional) Implementieren Sie den Sethi-Ullman-Algorithmus in Ihrem Kompiler.