

Name _____

Matr. Nr. _____

Übungsgruppe

Punkte _____ korr. _____

- 1 (Wöß) Do 10¹⁵ - 11⁴⁵
- 2 (Wöß) Do 12⁰⁰ - 13³⁰
- 3 (Rammerstorfer) Do 17¹⁵ - 18⁴⁵

Letzter Abgabetermin:

Donnerstag, 11.12. 2003, 8¹⁵ Uhr

Codeerzeugung – Teil 1

(24 Punkte)

Erweitern Sie Ihren Compiler um die Codeerzeugung gemäß der in den Unterlagen ausgegebenen Spezifikation der *MicroJava-VM*. Die dafür nötigen Klassen befinden sich im Package *ssw.mj.codegen* und haben folgende Schnittstelle (Angabe ohne Access Modifier).

```
class Item {
    static final int    // item kinds
        Con    = 0,    // Konstante
        Local  = 1,    // lokale Variable
        Static = 2,    // globale Variable
        Stack  = 3,    // Ausdruck auf dem Stack
        Fld    = 4,    // Feld einer inneren Klasse
        Elem   = 5,    // Arrayelement
        Meth   = 6;    // Methodenaufruf

    int kind;    // Con, Local, Static, Stack, Fld, Elem, Meth
    Struct type; // item type
    int adr;    // Con: Wert; Local, Static, Fld, Meth: Adresse; Cond: Operator
    Obj obj;    // Meth: method object from symboltable
}
```

```
class Code {...}
```

Die Angabedatei `UB-UE5-Angabe.zip` enthält bereits ein Gerüst für den Codegenerator (`Code.java`) und die Klasse `Item` (`Item.java`).

Zusätzlich befindet sich dort auch ein Decoder (`Decoder.java`), der Objektcode der *MicroJava-VM* in textueller Form ausgibt. Sie können ihn als Hilfsmittel verwenden. Der Aufruf ist wie folgt:

- um direkt den Codepuffer auszugeben: `Decoder.decode(buf, 0, codeLength);`
- um den Inhalt der `.obj`-Datei auszugeben:
 - von der Commandozeile: `> java ssw.mj.codegen.Decoder <Objectfile>`
 - im Programm: `Decoder.decodeFile(file);`

Hinweise:

- In diesem ersten Teil der Codegenerierung sollen Sie *nur* die Teile vorsehen, die im VO-Skriptum bis einschließlich Folie 6.26 (Zuweisungen) beschrieben werden, d.h., Sie brauchen noch *keinen Code für Sprünge und Methoden bzw. Methodenaufrufe* erzeugen.
- Daher dürfen Sie bei dieser Übung in den folgenden Parsermethoden die für die korrekte Codeerzeugung notwendigen *Änderungen* noch *weglassen*:
 - `MethodDecl`
 - `BlockStat`
 - `ActPars`
 - `Condition`
 - `CondTerm`
 - `CondFact`
 - bei `Factor` die Alternative **Methodenauf** (`sym == lpar nach Designator`) (hier soll vorläufig nur die Art des Items auf *Stack* gesetzt werden (`x.kind=Item.Stack`))
 - `Relop`
- Außerdem dürfen Sie auch die Codeerzeugung für Anweisungen, die Sprünge oder Methoden(aufrufe) benötigen noch unverändert lassen, d.h. keinen Code dafür erzeugen. (Das betrifft die folgenden Alternativen in der Methode `Statement`):
 - bei `Designator` (`case ident:`) die Alternative **Methodenauf** (`case lpar:`)
 - **if**-Anweisung (`case ifKW:`)
 - **while**-Anweisung (`case whileKW:`)
 - **do-while**-Anweisung (`case doKW:`)
 - **continue**-Anweisung (`case continueKW:`)
 - **return**-Anweisung (`case returnKW:`)
- Vergessen Sie nicht, wieder alle Kontext- und sonstigen Nebenbedingungen, die Sie nun prüfen können, auch tatsächlich zu prüfen und entsprechende Fehlermeldungen über `Parser.Errors.semError` (alles, was wir dem Compiler jetzt hinzufügen, ist Semantik) auszugeben.