Institut für Systemsoftware
Prof. Dr. H. Mössenböck

Übungen zu Übersetzerbau WS 2005/2006

Übung 4

Zuname _____ Vorname _____ Matr.-Nr. _____

Übungsgruppe

- ❑ 1 (Kotzmann) Do 10^{15}-11^{45}
- ❑ 2 (Kotzmann) Do 13^{45}-15^{15}
- ❑ 3 (Wimmer) Do 10^{15}-11^{45}

Punkte _____ korr. _____

Letzter Abgabetermin
Mittwoch, 23.11.2005, 20^{15} Uhr

Symbolliste und Fehlerbehandlung

1. Symbolliste (14 Punkte)

Erweitern Sie Ihren Parser um eine Symbolliste. Die dafür notwendigen Klassen *Tab*, *Scope*, *Obj* und *Struct* befinden sich in einem eigenen Package *ssw.mj.symtab*.

Die Klassen *Obj* und *Scope* sind bereits vollständig implementiert. Implementieren Sie in der Klasse *Struct* noch die Methoden, um die in *MicroJava* benötigten Arten der Typ-Kompatibilität zu prüfen (siehe VO-Skriptum, Kapitel 5.3, Seite 9).

Vervollständigen Sie in der Klasse *Tab* sind statischen Methoden zur Verwaltung der Symbolliste: Die Methode *init()* initialisiert die Symbolliste und trägt alle vordeklarierten Namen (Funktionen, Typen und Konstanten) von *MicroJava* ein.

Die Methoden *openScope()* und *closeScope()* legen einen neuen *topScope* an bzw. entfernen den aktuellen *topScope* und erhöhen bzw. vermindern den aktuellen *level*.

Die Methode *insert()* erzeugt ein Symbollistenobjekt (Klasse *Obj*), setzt seine Attribute und fügt es im aktuellen Gültigkeitsbereich (*topScope*) in die Symbolliste ein. Wenn dort bereits ein Eintrag mit dem gleichen Namen vorhanden ist, soll ein semantischer Fehler (*DECL_NAME*) ausgegeben werden. Wird die maximale Anzahl von lokalen oder globalen Variablen überschritten, soll die Fehlermeldung *LOCALS* bzw. *GLOBALS* ausgegeben werden.

Die Methoden *find()* und *findField()* dienen dazu, später auf die Symbollisteneinträge zugreifen zu können (semantische Fehler *NOT_FOUND* bzw. *NO_FIELD*, wenn ein Name nicht gefunden wird). *find()* sucht nach einem Namen beginnend im aktuellen bis zum äußersten Gültigkeitsbereich. *findField()* sucht nach einem Namen in einer inneren Klasse, deren *Struct* in der Schnittstelle mitgegeben wird.

Erweitern Sie den Parser, so dass die vollständige Symboltabelle aufgebaut wird. Zugriffe auf die Symboltabelle beim Parsen von Statements (z.B. beim Zugriff auf Namen in der Grammatik-Regel *Designator*) sind bei dieser Übung noch *nicht* gefordert. Daher können im Parser auch noch keine semantischen Fehler (z.B. Verwendung von undeklarierten Variablen) ausgegeben werden.

2. Fehlerbehandlung (10 Punkte)

Erweitern Sie Ihren Parser derart, dass er die Analyse nicht beim ersten erkannten Fehler abbricht, sondern nach der Methode der *speziellen Fangsymbole* fortsetzt. Fügen Sie dazu zwei Synchronisationspunkte in Ihre Implementierung ein:

1. Wenn bei einer Reihe von aufeinander folgenden Deklarationen (*ConstDecl*, *VarDecl*, *ClassDecl*) ein Fehler auftritt, soll unmittelbar nach der fehlerhaften Deklaration wieder aufgesetzt werden. Beschränken Sie sich dabei nur auf globale Deklarationen, d.h. Sie müssen Variablen-Deklarationen innerhalb von Klassen oder Methoden nicht berücksichtigen.
2. Wenn bei einer Reihe von aufeinander folgenden Statements ein Fehler auftritt, so soll beim nächsten Statement (nach dem fehlerhaften) wieder aufgesetzt werden.

Suchen Sie in der MicroJava-Grammatik jene Stellen, an denen diese Synchronisationspunkte eingefügt werden müssen und implementieren Sie für den Wiederaufsatz die Methoden *recoverDecl()* und *recoverStat()*, die jeweils die Analyse nach einem Fehler in einer Deklaration oder einem Statement fortsetzen.

Bedenken Sie auch, dass sich *ident* nicht als Fangsymbol eignet, und verwenden Sie zusätzlich semantische Informationen, um bei bestimmten Namen doch wieder aufzusetzen (wenn es sich um Typbezeichnungen handelt).

Unterdrücken Sie irreführende Folgefehlermeldungen, indem Sie die Klasse *Error* erweitern: Eine Fehlermeldung soll nur dann ausgegeben werden, wenn seit dem letzten Fehler mindestens 3 Tokens korrekt verarbeitet wurden.

Abgabe

Die Abgabe der Übungen 2 – 6 muss auf Papier und elektronisch erfolgen. Geben Sie folgende Dateien ab:

- Ausgedruckt auf Papier: *Parser.java*, *Errors.java*, *Tab.java*, *Struct.java*.
- Elektronisch als ZIP-Datei: **Alle** Dateien, die zum **Ausführen** des Compilers benötigt werden (Packages *ssw.mj*, *ssw.mj.codegen und ssw.mj.symtab*), also auch alle Klassen der Angabe. Auf die Datei *messages.properties* nicht vergessen. Die Verzeichnis-Struktur muss in der ZIP-Datei erhalten bleiben.
- **Nicht abzugeben:** JUnit-Testfälle, *.class*-Dateien, Projekt-Dateien von IDEs.