Letzter Abgabetermin: Mittwoch, 06.12.2017, 1800 Uhr

Symbolliste und Fehlerbehandlung (24 Punkte)

1. Symbolliste (14 Punkte)

Erweitern Sie Ihren Parser um eine Symbolliste. Die dafür notwendigen Klassen *TabImpl* im befindet sich im Package *ssw.mj.impl.* Die Klassen *Scope*, *Obj* und *Struct* im Package *ssw.mj.symtab*.

Die Klassen *Obj* und *Scope* sind bereits vollständig implementiert. Die fehlenden Methoden der Klasse *Struct* (Implementierung in *StructImpl*) werden erst in der nächsten Übung benötigt.

Vervollständigen Sie die Klasse *TabImpl*, zur Verwaltung der Symbolliste: Der Konstruktor initialisiert die Symbolliste, d.h. er baut das Universum auf. Dazu trägt er alle vor deklarierten Funktionen, Typen und Konstanten in die Symbolliste ein.

Die Methode *openScope()* legt einen neuen Gültigkeitsbereich (*curScope*) an und erhöht den aktuellen *level*; *closeScope()* entfernt den aktuellen *curScope* und vermindert den aktuellen *level*.

Die Methode *insert()* erzeugt ein Symbollistenobjekt (Klasse *Obj*), setzt seine Attribute und fügt es im *curScope* in die Symbolliste ein. Wenn dort bereits ein Eintrag mit dem gleichen Namen vorhanden ist, muss der semantische Fehler *DECL_NAME* ausgegeben werden. Wird die maximale Anzahl von Variablen oder Feldern überschritten, muss die Fehlermeldung *TOO_MANY_LOCALS*, *TOO_MANY_GLOBALS* bzw. *TOO_MANY_FIELDS* ausgegeben werden. Da in der Methode *insert()* nicht zwischen lokalen Variablen und Feldern unterschieden werden kann muss diese Überprüfung muss im Parser eingebaut werden. Die Anzahl der lokalen Variablen soll nach *VarDecl* in *MethodDecl* überprüft werden.

MethodDecl, die innerhalb einer Klasse (*ClassDecl*) vorkommen, sind sogenannte Klassenmethoden. Sie haben einen impliziten ersten Parameter *this* von dem Typ, der durch die aktuelle *ClassDecl* Deklaration beschrieben wird. Dazu wird der Methode *MethodDecl* ein *Obj clazz* parameter übergeben. Im Fall von Programmmethoden (nicht in *ClassDecl*) ist der Parameter *null*.

Klassenmethoden teilen sich einen Sichtbarkeitsbereich mit den Feldern der *ClassDecl*. Zusätzlich zur Symboltabelle müssen Klassenmethoden in der *Struct*, die den (Klassen-) Typ beschreibt, registriert werden. Die Methode *addMethod(Obj)* in der Klasse *Struct* ist für diesen Zweck schon fertig implementiert.

Klassenmethoden können nur über ein explizit angegebenes Klassenobjekt aufgerufen werden zum Beispiel *o.foo()*, aber auch in einer Klassenmethodendeklaration mit *this.foo()*.

Die Methoden *find()* und *findMember()* suchen Symbollisteneinträge. *find()* sucht nach einem Namen beginnend im aktuellen bis zum äußersten Gültigkeitsbereich, wird der Name nicht gefunden muss der semantische Fehler *NOT_FOUND* ausgegeben werden. *findMember()* sucht nach einem Namen (Feld oder Klassenmethode) in einer Klasse, deren *Struct* in der Schnittstelle mitgegeben wird. Wird der Name nicht gefunden muss der semantische Fehler *NO_FIELD* ausgegeben werden.

2. Fehlerbehandlung (10 Punkte)

Erweitern Sie Ihren Parser um eine Fehlerbehandlung nach der Methode der *speziellen Fangsymbole*. Fügen Sie dazu folgende 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 können Variablen-Deklarationen innerhalb von Klassen und Methoden ignorieren.
2. Wenn bei aufeinanderfolgenden Methoden ein Fehler auftritt soll nach der fehlerhaften Methode neu aufgesetzt werden. Benutzen sie *void* und *ident* als Fangsymbole, aber *ident* nur wenn es sich um einen Typen handelt.
3. Wenn bei einer Reihe von aufeinander folgenden Statements ein Fehler auftritt, so soll beim nächsten Statement (nach dem fehlerhaften) wiederaufgesetzt werden.

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

Bedenken Sie, dass sich *ident* nur bedingt als Fangsymbol eignet. Verwenden Sie semantische Informationen bei Deklarationen, um nur dann wieder aufzusetzen, wenn es sich um eine Typbezeichnung handelt. Beim Wiederaufsatz für Statements sind *ident* und *lbrace* ungeeignet und müssen daher aus den Fangsymbolen entfernt werden.

Unterdrücken Sie Folgefehlermeldungen im Parser. Eine Fehlermeldung darf nur dann ausgegeben werden, wenn seit dem letzten Fehler mindestens drei Tokens korrekt verarbeitet wurden, d.h. drei *scan()*s ohne Fehler durchgeführt wurden.

Abgabe und Hinweise

Die Abgabe der Übungen muss elektronisch erfolgen. Geben Sie folgende Dateien ab:

> Elektronisch in das Repository: Alle Quellcode-Dateien, die zum Ausführen des Compilers benötigt werden (Packages *ssw.mj*, *ssw.mj.codegen*, *ssw.mj.impl* und *ssw.mj.symtab*), also auch alle Klassen der Angabe. Die Verzeichnis-Struktur muss erhalten bleiben.
>
> svn://ssw.jku.at/2017W/UB/k<MatrNr>/branches/UE4

JUnit Testfälle: ScannerTest, ParserTest, SymbolTableTest