Prof. Dr. Volker Linnemann
Dr. Sven Groppe
Universität zu Lübeck
Institut für Informationssysteme

Lübeck, den 10.7.2008

Vorlesung „Algorithmen und Datenstrukturen“

Sommersemester 2008

13. Übungsblatt

1. Erstes Beispiel mit JavaCC (5 Punkte)

Bitte befolgen Sie die unten stehenden Anweisungen, um einen ersten Parser mit Hilfe des Compiler-Compilers JavaCC [Co06] zu erzeugen und zum Laufen zu bringen. Das erste JavaCC Beispiel implementiert einen Scanner und einen Parser für die Grammatik G_{ADD} der Präsenzaufgabe zu diesem Übungsblatt.

1. Laden Sie *Tutorial1.zip* von der Webseite herunter und entpacken Sie dieses Zip-Archiv.
2. Starten Sie eine Kommandozeile.
3. Wechseln Sie in das Verzeichnis "`exercise1`" der entpackten Dateien mit Hilfe des Befehls "`cd exercise1`" in der Kommandozeile.
4. Tippen Sie "`java -classpath ..\lib\javacc.jar javacc Calculator1.jj`" in die Kommandozeile. `Calculator1.jj` beinhaltet die Grammatik für den javacc Compiler-Compiler (für die lexikalische und Syntaktische Analyse). Weiterer Java Sourcecode wird generiert, der einige Konstanten (`Calculator1Constants.java`), den Scanner (`Calculator1TokenManager.java`) und den Parser (`Calculator1.java`) enthält.
5. Kompilieren Sie alle generierten Java Quelldateien mittels "`javac *.java`".
6. Lassen Sie den generierten Parser mittels "java Calculator1" laufen.
7. Nun probieren Sie einige Ausdrücke wie "`2+4+(8-1)`", "`2+5--1`", "`2+7+(8-(-1)+78)`", "`7+(`", "`a+d`", ... Bitte tippen Sie nur einen Ausdruck pro Zeile ein.

Schauen Sie in die JavaCC Grammatik "`Calculator1.jj`" und die generierten Quelltexts für die Konstanten (`Calculator1Constants.java`), den Scanner (`Calculator1TokenManager.java`) und den Parser (`Calculator1.java`). Erweitern Sie die Grammatik "`Calculator1.jj`" um Multiplikation- ("`*`"), Modular- ("`%`") und Division- ("`/`") Operationen, in dem Sie ein neues Nichtterminal `product` in der JavaCC Datei "`Calculator1.jj`" einführen. Dabei sollen die Multiplikation- ("`*`"), Modular- ("`%`") und Division- ("`/`") Operationen höhere Präzedenz als die Additions- ("+") und Subtraktions- ("−") Operationen haben, d.h. `5+3*2` ist mit `5+(3*2)` identisch und nicht mit `(5+3)*2`.

Literatur

[Co06] CollabNet, javacc Project home, https://javacc.dev.java.net/, 2006.

2. Ermittlung des Resultates eines Arithmetischen Ausdruckes (5 Punkte)

Wechseln Sie in das Verzeichnis "`exercise2`" mittels "`cd ..\exercise2`" in der Kommandozeile.

"`Calculator2.jj`" beinhaltet eine javacc Grammatik, welche mit Java Code annotiert ist, so dass der generierte Parser das Resultat von arithmetischen additiven Ausdrücken berechnet und ausgibt.

Erweitern Sie die Grammatik "`Calculator2.jj`" um die Berechnung und Ausgabe von Multiplikation- ("*"), Modular- ("%") und Division- ("/") Operationen. Die gleichen Bemerkungen zur Präzedenz der Operationen wie unter Aufgabe 1 gelten auch hier.

3. Generation des Syntaxbaumes (1 Punkt)

Wechseln Sie in das Verzeichnis "`exercise3`" mittels "`cd ..\exercise3`" in der Kommandozeile.

Während einfache Berechnungen “inline” wie in Aufgabe 2 beschrieben in JavaCC Grammatiken integriert warden können, müssen komplexere Berechnungen und Übersetzungen oftmals auf den Syntaxbaum oder eine Variante des Syntaxbaumes durchgeführt werden.

Die Generation des Syntaxbaumes bkommt man bei JavaCC fast geschenkt, wenn man den Präprozessor JJTree verwendet. Bitte kompilieren Sie "`Calculator3.jjt`" mittels "`java –classpath ..\lib\javacc.jar jjtree Calculator3.jjt`" in der Kommandozeile, welches "`Calculator3.jj`". Kompilieren Sie nun wie gewohnt "`Calculator3.jj`".

Erweitern Sie die Grammatik "`Calculator3.jjt`" entsprechend um die Behandlung von Multiplikation- ("*"), Modular- ("%") und Division- ("/") Operationen. Die gleichen Bemerkungen zur Präzedenz der Operationen wie unter Aufgabe 1 gelten auch hier.

4. Generation des Abstrakten Syntaxbaumes (5 Punkte)

Wechseln Sie in das Verzeichnis "`exercise4`" mittels "`cd ..\exercise4`" in der Kommandozeile.

Der generierte Syntaxbaum der JJTree Grammatik "`Calculator3.jjt`" speichert zu viele redundante Informationen. Können Sie Beispiele redundanter Informationen geben?

Redundante Informationen in Syntaxbäumen machen diesbezügliche Algorithmen unnötig komplex. Daher unterstützt JJTree die manuelle Manipulation zum Vermeiden von redundanten Informationen, zum Beispiel bei sogenannten Kettenproduktionen. Im Vergleich zum Syntaxbaum wird dieser Baum ohne redundante Informationen Abstrakter Syntaxbaum genannt. Kompilieren Sie das "`Calculator4.jjt`" JJTree Grammatik Datei und lassen Sie den generierten Parser mit ein paar Beispielen laufen.

Erweitern Sie die Grammatik "`Calculator4.jjt`" entsprechend um die Behandlung von Multiplikation- ("*"), Modular- ("%") und Division- ("/") Operationen. Die gleichen Bemerkungen zur Präzedenz der Operationen wie unter Aufgabe 1 gelten auch hier.

Bemerkung: Wir stellen den Quelltext "`ASTIntLit.java`" wegen der folgenden Gründe zur Verfügung:

- Die generierte Klasse "`ASTIntLit.java`" des JJTree Präprozessors würde keine Methoden zum Abspeichern weiterer Informationen im Knoten des Abstrakten Syntaxbaumes beinhalten.
- Wir wollen jedoch einen ganzzahligen Wert im entsprechenden Knoten des Abstrakten Syntaxbaumes abspeichern mittels explizitem Java-Code in "`Calculator4.jjt`".
- Bereits bestehende Java Dateien werden durch den JJTree Präprozessor **nicht** ersetzt durch generierten Quelltext.

5. Berechnung des Resultates von Arithmetischen Ausdrücken durch Verwendung des Abstrakten Syntaxbaumes (10 Punkte)

In den folgenden beiden Teilaufgaben werden wir zwei verschiedene Ansätze für Berechnungen und Übersetzungen basierend auf den Abstrakten Syntaxbaum kennen lernen. Ein Ansatz verwendet eine recursive Methode, während der andere Ansatz die Visitor-Schnittstelle des JJTree Präprozessors verwendet.

a) Besuchen des Abstrakten Syntaxbaumes mittels einer rekursiven Methode

Wechseln Sie in das Verzeichnis "`exercise5a`" mittels "`cd ..\exercise5a`" in der Kommandozeile.

Kompilieren Sie die "`Calculator5a.jjt`" JJTree Grammatik Datei und lassen Sie den generierten Parser mit einigen Beispielen laufen. Ähnlich zur Aufgabe 2 berechnet der generierte Parser den Wert eines arithmetischen Ausdruckes. Wir verwenden diesmal den Abstrakten Syntaxbaum und eine rekursive Methode "`public int recursiveMethod(Node n) throws Exception`", welche zuerst die Auswertung auf den Kindsknoten durchführt, um anschließend den Operator des aktuellen Knotens im Abstrakten Syntaxbaum anzuwenden.

Erweitern Sie die Grammatik "Calculator5a.jjt" um die entsprechende Berechnung und Ausgabe von Multiplikation- ("*"), Modular- ("%") und Division- ("/") Operationen. Die gleichen Bemerkungen zur Präzedenz der Operationen wie unter Aufgabe 1 gelten auch hier. Erweitern Sie weiterhin die rekursive Methode `public int recursiveMethod(Node n) throws Exception`, so dass die neuen Nichtterminale bei der Berechnung des Ergebnisses berücksichtigt werden.

b) Besuchen des Abstrakten Syntaxbaumes mittels Verwendung der Visitor Schnittstelle

Wechseln Sie in das Verzeichnis "exercise5b" mittels "`cd ..\exercise5b`" in der Kommandozeile.

Kompilieren Sie die "Calculator5b.jjt" JJTree Grammatik Datei und lassen Sie den generierten Parser mit einigen Beispielen laufen. Ähnlich zur Aufgabe 2 und 5 b) berechnet der generierte Parser den Wert eines arithmetischen Ausdruckes. Wir verwenden diesmal die Visitor Schnittstelle von JJTree. JJTree verwendet eine Variante des VISITOR Patterns für allgemeines Besuchen des Abstrakten Syntaxbaumes. Dazu haben wir die Option "VISITOR" auf "true" zu setzen. Betrachten Sie dazu bitte "Calculator5b.jjt". Falls "VISITOR" auf "true" gesetzt ist, generiert JJTree

- das Interface "Calculator5bVisitor.java". Wir müssen dieses Interface durch eine Klasse (hier "Calculator5bVisitorImplementation") implementieren, um das VISITOR Pattern anzuwenden. Bitte schauen Sie sich "Calculator5bVisitor.java" und "Calculator5bVisitorImplementation.java" an.

- die Methoden

  ```
  /** Accept the visitor. **/
  public Object jjtAccept(Calculator5bVisitor visitor, Object data);
  ```

  im Interface "Node.java".

- die Methoden

  ```
  /** Accept the visitor. **/
  public Object jjtAccept(Calculator5bVisitor visitor, Object data)
  { return visitor.visit(this, data) }

  /** Accept the visitor. **/
  public Object childrenAccept( Calculator5bVisitor visitor,
                                Object data) {
        if (children != null) {
              for (int i=0; i < children.length; ++i) {
                    children[i].jjtAccept(visitor, data);}}
        return data;}
  ```

  in "SimpleNode.java".

Achten Sie darauf, dass alle nicht-generierten Klassen wie "ASTIntLit.java" zusätzlich die Methode

```
/** Accept the visitor. **/
public Object jjtAccept(Calculator5bVisitor visitor, Object data) {
      return visitor.visit(this, data)
}
```

beinhalten, ansonsten kann das VISITOR Pattern nicht verwendet werden.

Erweitern Sie die Grammatik "Calculator5b.jjt" um die entsprechende Berechnung und Ausgabe von Multiplikation- ("*"), Modular- ("%") und Division- ("/") Operationen. Die gleichen Bemerkungen zur Präzedenz der Operationen wie unter Aufgabe 1 gelten auch hier. Erweitern Sie weiterhin die Klasse `VisitorCalculator5bImplementation` in der Datei "VisitorCalculator5bImplementation.java", so dass die neuen Nichtterminale bei der Berechnung des Ergebnisses berücksichtigt werden.

Bemerkungen:

- Jede Seite soll oben rechts den Namen der Abgebenden und die Übungsgruppennummer (wichtig!) enthalten.
- Lösungen für die Übungsaufgaben sind (in der Regel) zu zweit abzugeben.
- Kommentieren Sie Ihre Lösungen! Besteht eine Lösung aus mehreren Zetteln, so sind diese zusammen zu heften. Bitte keine Hüllen, Mappen, o.ä..
- Bitte schicken Sie *Programmieraufgaben zusätzlich zur Abgabe auf Papier in elektronischer Form per Email* an Ihren jeweiligen Tutor.
- Kommentieren Sie ihren Quelltext bei Programmieraufgaben. Dabei sollen keine Trivialitäten kommentiert werden, also bitte keine Kommentare wie

  ```
  x=5; // Wir weisen nun der Variablen x den Wert 5 zu
  ```

  sondern sinnvolle Kommentare, die Ideen des Quelltextabschnittes beschreiben oder auf Unteraufgaben (z. B. a), b), …) hinweisen.
- **Hinreichende Bedingung für die Zulassung zur Klausur:** 50% der erreichbaren Punkte bei jedem Übungszettel (bis auf zwei) und einmaliges Vorrechnen in der Übung
- **Zertifikatskriterium:** Das Bestehen der Klausur am Ende des Semesters

Abgabetermin: Donnerstag, 17.7.2008, nach der Vorlesung