Hochschule Bochum
Bochum University
of Applied Sciences

Institut für Informatik
Prof. Dr.-Ing. Wolf Ritschel

Praktikum Theoretische Informatik

In der Vorlesung haben Sie die Konstruktion eines Parsers, der nach dem Prinzip des rekursiven Abstiegs arbeitet mit Hilfe einer rechtrekursiven kontextfreien Grammatik kennengelernt.

In der Vorlesung wurde u.a. das folgende Beispiel für eine solche Grammatik vorgestellt:

num → digit | digit num

digit → '1' | '2' | '3' | '4' | '5' |'6' | '7' | '8' | '9' | '0‘

Als Moodle finden Sie das Java Listing eines Parsers, der die Syntax dieser Datei überprüft. Der Parser gibt den zugehörigen Syntaxbaum, ggf. auch Syntaxfehler auf der Konsole aus.

Aufgabenstellung:

a) Schreiben Sie in Anlehnung an das Beispiel mit Hilfe der Methode des Rekursiven Abstiegs in Java einen Parser für die nachfolgend angegebene Grammatik für arithmetische Ausdrücke:

expression → term rightExpression

rightExpression → '+' term rightExpression

rightExpression → ‘-' term rightExpression

rightExpression → ε

term → operator rightTerm

rightTerm → '⋆' operator rightTerm

rightTerm → ‘/' operator rightTerm

rightTerm → ε

operator → '(' expression ')' | num

num → digit | digit num

digit → '1' | '2' | '3' | '4' | '5' |'6' | '7' | '8' | '9' | '0’

b) Sorgen Sie dafür, dass Ihr Parser den Syntaxbaum und wenn nötig möglichst hilfreiche Syntaxfehler ausgibt. Insbesondere auf nicht korrekte Klammerung soll durch eine entsprechende Fehlermeldung hingewiesen werden.

Beispiel einer mögliche Ausgabe des Parsers für den Ausdruck **3*(5+2**

```
expression->
    term
    Operator->
      num->
        digit->
          match: 3
    rightTerm->
      match: *
      Operator->
        match: (
        expression->
            term
            Operator->
              num->
                digit->
                  match: 5
            rightTerm->
              Epsilon
          rightExpression->
            match: +
              term
              Operator->
                num->
                  digit->
                    match: 2
              rightTerm->
                Epsilon
            rightExpression->
              Epsilon
Syntax Fehler beim 7. Zeichen: EOF
Geschlossene Klammer erwartet
Fehler im Ausdruck
```

c) Bereiten Sie zu einem arithmetischen Testausdruck, der mindestens eine Multiplikation bzw. Division, eine Addition bzw. Subtraktion sowie einen geklammerten Ausdruck enthält, einen von Hand erstellten Syntaxbaum vor, um zu testen, ob Ihr Parser korrekt arbeitet.

d) Überlegen Sie sich, warum man im Parser für arithmetische Ausdrücke nicht die folgende, in der Vorlesung vorgestellte Implementierung der Regeln **num → digit | num digit** verwenden kann:

```
// num -> num digit | digit
static boolean num(){
   char [] digitSet = {'1','2','3','4','5','6','7','8','9','0'};
   char [] eofSet ={EOF};
   if (lookAhead(digitSet))
      return num()&& digit();      //num-> num digit
   else if (lookAhead(eofSet))
      return digit();              //num->digit
   else{
      syntaxError("Ziffer erwartet");    //Syntaxfehler
      return false;
   }
}
```

Bemerkung zum Ablauf des Praktikums:

Bereiten Sie schon zum Praktikumstermin ein entsprechendes Java Programm und Ihre Testdaten vor.

Ihre Lösung können Sie zum Erlangen des Testates bei Herrn Dipl-Ing. Stefan Rohkämper, MSc. (Raum C4-21) präsentieren.

Die Programmierung in Aufgabeteil a) und b) kann in Gruppen zu maximal drei Personen bearbeitet werden, wenn jedes Gruppenmitglied mindestens 3 Regeln der Grammatik selber programmiert und das durch entsprechende Antworten auf Fragen belegen kann.

Kennzeichnen Sie bitte im Kommentar Ihres Programms, welche Java Methode von welchem Gruppenmitglied programmiert wurde.

Aufgabenteil c) und d) müssen von jedem Praktikumsteilnehmer einzeln vorbereitet werden, d.h. also, dass eine 3er Gruppe auch jeweils 3 Syntaxbäume als Testdaten haben muss.