予習課題

今回の授業では、LL(k)構文解析器や、その生成系を正しく理解して使用することを目標とする。LL(k)構文解析器は LR(k)構文解析器や LALR(k)構文解析器と異なり、構文解析表を(ギリギリ)人間が作成できるという特徴がある。このため、パーサ・ジェネレータなどを利用して構文解析器を作成した際にも、その生成されたプログラムを推測しやすく、また追跡しやすいという利点がある。そのため、LL(k)パーサ・ジェネレータは構文解析器の仕組みを知らない人でも作りやすい。

LL(k)構文解析器で解析可能な文法を LL(k)文法とよぶ。これは、通常の BNF でよく見かける文法の種類、文脈自由文法(Context-Free Grammar, CFG)のサブセットである。このサブセットとして利用できなくなった個所に強い特徴があり、この特徴を正しく理解していないと構文解析器を正しく作成できない場合がある。

この課題では、LL(k)構文解析器の特徴をいくつか取り込んだ問題を用意した。とくに LL(k)構文解析器の作成方法を現時点で語るつもりは全くないので、力技で解いていただければと思う。ただし、以下の利用を禁止する。

- パーサ・ジェネレータなどのツール(JavaCC, ANTLR, CUPS など)
- パーサ・コンビネータなどのライブラリ(JParsec など)

なお、別に LR 構文解析系の手法を元に課題を解いてもよいが、解析表が大変なことになるのでお勧めしない。

準備

Google グループより「parserstudy-01-eclipse-project.zip」をダウンロードして、Eclipse 上にプロジェクトとしてインポートする。その後、コマンドコンソールからそのプロジェクトをインポートしたディレクトリ上で、下記のコマンドを実行する。

```
mvn eclipse:eclipse
```

これで、同プロジェクトに「Refresh」を実行することで、Java のプロジェクトとして正しく認識されるはずである。

つぎに、下記のクラスを実行する。

```
jp.undo.vtable.parser.prep.ExampleParser
```

これは、以降の解きかたの指針を示すサンプルプログラムである。すべてのプログラムはテキストを解析し、それに対応する抽象構文木を生成することを目的としている。サンプルのように、テキストを字句解析(Lexical Analysis)する部分や、抽象構文木のモデルオブジェクトは既に用意してあるため、それらを利用すること。それぞれは、以下のパッケージに格納されている。

- 字句解析器

- jp.undo.vtable.parser.lang.arithmetic.scanner.TokenScanner

- 抽象構文木のモデル
  - jp.undo.vtable.parser.lang.arithmetic.model.*

また、以降の課題ではそれぞれに単体テストを用意しており、それらが全て成功することで課題を解けたものとみなすことにする。

課題 0

jp.undo.vtable.parser.prep.Q0Parser クラス内の parse()メソッドの内容を、同メソッドの Javadoc に説明される動作と同等の動作を行うようにせよ。また、単体テスト jp.undo.vtable.parser.prep.Q0ParserTest が全て成功することを確認すること。

なお、Q0Parser は次のような構文規則と構文アクションを持ち、Expression から開始する言語である。

```
  Expression   ::=   Value$a '+' Value$b              ; new Add(a, b)
       Value   ::=   NUMBER$t                         ; new Value(t.image)
```

課題 1

jp.undo.vtable.parser.prep.Q1Parser クラス内の parse()メソッドの内容を、同メソッドの Javadoc に説明される動作と同等の動作を行うようにせよ。また、単体テスト jp.undo.vtable.parser.prep.Q1ParserTest が全て成功することを確認すること。

なお、Q1Parser は次のような構文規則と構文アクションを持ち、Expression から開始する言語である。

```
  Expression   ::=   '+' Expression$a Expression$b    ; new Add(a, b)
                |    '-' Expression$a Expression$b    ; new Subtract(a, b)
                |    Value$v                          ; v
       Value   ::=   NUMBER$t                         ; new Value(t.image)
```

課題 2

jp.undo.vtable.parser.prep.Q2Parser クラス内の parse()メソッドの内容を、同メソッドの Javadoc に説明される動作と同等の動作を行うようにせよ。また、単体テスト jp.undo.vtable.parser.prep.Q2ParserTest が全て成功することを確認すること。

なお、Q2Parser は次のような構文規則と構文アクションを持ち、Expression から開始する言語である。

```
  Expression   ::=   '-' Value$a                       ; new Minus(a)
                |    '-' Value$a Value$b               ; new Subtract(a, b)
```

```
                |    Value$v                          ; v
      Value     ::=  NUMBER$t                        ; new Value(t.image)
```

課題 3

jp.undo.vtable.parser.prep.Q3Parser クラス内の parse()メソッドの内容を、同メソッドの Javadoc に説明される動作と同等の動作を行うようにせよ。また、単体テスト jp.undo.vtable.parser.prep.Q3ParserTest が全て成功することを確認すること。

なお、Q3Parser は次のような構文規則と構文アクションを持ち、Expression から開始する言語である。

```
  Expression    ::=  Value$a '+' Value$b             ; new Add(a, b)
                |    Value$a '-' Value$b             ; new Subtract(a, b)
                |    Value$a '*' Value$b             ; new Multiply(a, b)
                |    Value$a '/' Value$b             ; new Divide(a, b)
                |    Value$v                         ; v
      Value     ::=  NUMBER$t                        ; new Value(t.image)
```

課題 4

jp.undo.vtable.parser.prep.Q4Parser クラス内の parse()メソッドの内容を、同メソッドの Javadoc に説明される動作と同等の動作を行うようにせよ。また、単体テスト jp.undo.vtable.parser.prep.Q4ParserTest が全て成功することを確認すること。

なお、Q4Parser は次のような構文規則と構文アクションを持ち、Expression から開始する言語である。

```
  Expression    ::=  AddExpr$e                       ; e
     AddExpr    ::=  AddExpr$a '+' Value$b           ; new Add(a, b)
                |    Value$a                         ; a
      Value     ::=  NUMBER$t                        ; new Value(t.image)
```

課題 5

jp.undo.vtable.parser.prep.Q5Parser クラス内の parse()メソッドの内容を、同メソッドの Javadoc に説明される動作と同等の動作を行うようにせよ。また、単体テスト jp.undo.vtable.parser.prep.Q5ParserTest が全て成功することを確認すること。

なお、Q5Parser は次のような構文規則と構文アクションを持ち、Expression から開始する言語である。

```
  Expression  ::=  AddExpr$e                        ; e
     AddExpr  ::=  AddExpr$a '+' MultExpr$b        ; new Add(a, b)
              |    AddExpr$a '-' MultExpr$b        ; new Subtract(a, b)
              |    MultExpr$a                      ; a
    MultExpr  ::=  MultExpr$a '*' UnaryExpr$b      ; new Multiply(a, b)
              |    MultExpr$a '/' UnaryExpr$b      ; new Divide(a, b)
              |    UnaryExpr$a                     ; a
   UnaryExpr  ::=  '+' UnaryExpr$e                 ; new Plus(e)
              |    '-' UnaryExpr$e                 ; new Minus(e)
              |    PrimaryExpr$e                   ; e
 PrimaryExpr  ::=  '(' Expression$e ')'           ; new Parenthesized(e)
              |    Value$v                        ; v
       Value  ::=  NUMBER$t                       ; new Value(t.image)
```