AULAS TEÓRICO-PRÁTICAS DE COMPILADORES

2º semestre de 2002/2003

FICHA Nº 7 (3 horas)

Utilização do JavaCC [1] para gerar parsers. Esta aula tem como objectivo o início da aprendizagem do gerador de analisadores sintácticos JavaCC. Esta tarefa requer a aplicação dos conhecimentos até agora adquiridos na disciplina, principlamente no que respeita aos conceitos sobre análise sintáctica e sobre analisadores sintácticos descendentes.

O JavaCC utiliza um ficheiro com extensão .jj onde se encontra descrita a gramática para o parser e gera uma classe Java com o nome do parser e outras classes Java de suporte[1].

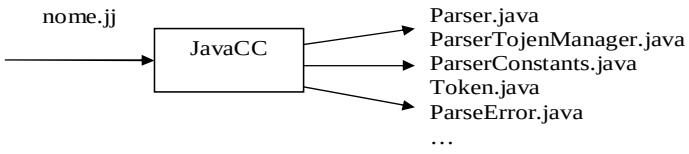

Figura 1. Entradas e saídas do JavaCC supondo que foi especificado "Parser" como nome do parser no ficheiro nome.jj.

1 Exemplo

Suponhamos que desejamos implementar um analisador sintáctico que reconheca expressões aritméticas com apenas um número inteiro positivo ou com uma adição ou uma subtração de dois números inteiros positivos (por exemplo: 2+3, 1-4, 5, etc.). De seguida apresenta-se uma gramática em EBNF com a especificação do símbolo terminal utilizando uma expressão regular:

INTEGER = [0-9]+

Aritm → Integer [("+" | "-") Integer]

Para implementar um parser que implemente esta gramática utilizando o JavaCC temos de criar um ficheiro com extensão .jj (vamos chamar-lhe: Exemplo.jj).

O ficheiro é constituído por:

1. Lista de opções (opcional): é onde pode ser definido o nível de *lookahead*, por exemplo.
2. Unidade de compilação Java (PARSER_BEGIN(nome) ... PARSER_END(nome))
3. Lista de produções da gramática (as produções aceitam os símbolos +, *, e ? com o mesmo significado, aquando da utilização em expressões regulares)

Por exemplo para a gramática anterior, poderemos criar o ficheiro Exemplo.jj seguinte:

[1] À medida que forem percebendo o JavaCC vão também tomando conhecimento do significado das classes de suporte.

```
PARSER_BEGIN(Exemplo)

    // código Java que invoca o parser
    public class Exemplo {

        public static void main(String args[]) throws ParseException {
            // criação do objecto utilizando o constructor com argumento para
            // ler do standard input (teclado)
            Exemplo parser = new Exemplo(System.in);
            Exemplo.Aritm();
        }
    }


PARSER_END(Exemplo)

// símbolos que não devem ser considerados na análise
SKIP :
{
    " " | "\t" | "\r"
}

// definição dos tokens (símbolos terminais)
TOKEN :
{
    < INTEGER : (["0" - "9"])+ >
    | < EOL : "\n" >
}

// definição da produção
void Aritm() : {}
{
    <INTEGER> ( ("+" | "-") <INTEGER> )? (<EOF> | <EOL>)
}
```

Em seguida deve fazer-se:

```
Javacc Exemplo.jj
```

Compilar o código Java gerado:

```
Javac *.java
```

Executar o analisador sintáctico:

```
Java Exemplo
```

Poderemos associar às funções referentes aos símbolos não-terminais pedaços de código Java. Por exemplo, as modificações apresentadas em seguida permitem escrever no ecrã mensagens a indicar os números que são lidos pelo parser:

```
void Aritm : {Token t1, t2;}
{
    t1=<INTEGER> {
        System.out.println("Integer = "+t1.image);
    }
    ( ("+" | "-") t2=<INTEGER> {
```

```
        System.out.println("Integer = "+t2.image);
    }
    )? (<EOF> | <EOL>)
}
```

Por cada símbolo terminal INTEGER, foi inserida uma linha de código Java que imprime no ecrã o valor do token lido (o método `image` da classe `Token` retorna uma String representativa do valor do token[2])

Insira estas modificações no ficheiro Exemplo.jj e volte a repetir o processo até à execução do parser. Verifique o que acontece.

Como poderíamos adicionar também a escrita do sinal (+ ou -) lido?

2 Exercícios

1. Considere a ficha Nº 5. No exercício pretendia-se implementar um reconhecedor de números inteiros e de números reais introduzidos via teclado.

    As expressões regulares que definem os números são:

    Inteiro → [0-9]+

    Real → [0-9]*.[0-9]+

Agora, pretende-se implementar o reconhecedor utilizando o gerador de analisadores léxicos e sintácticos: JavaCC.

Antes de iniciar as implementações referentes aos exercícios apresentados de seguida, deve ler com atenção o tutorial [1]. Tenha atenção aos símbolos que devem ser utilizados nas produções da gramática no JavaCC. Note que o JavaCC requer a definição de gramáticas não-ambíguas de modo a poder fornecer apenas uma árvore sintáctica concreta para uma dada frase.

2. Implemente utilizando o JavaCC um analisador sintáctico que permita reconhecer expressões aritméticas com as operações *, +, e -. O analisador deve tambem aceitar a utilização de parentesis;

3. Com base no analisador anterior, adicione o código Java necessário para implementar uma calculadora de expressões aceites pela gramática.

3 Referências de Apoio

1. JavaCC: http://www.experimentalstuff.com/Technologies/JavaCC/index.html

2. Oliver Enseling, "Build your own languages with JavaCC", Copyright © 2003 JavaWorld.com, an IDG company, http://www.javaworld.com/javaworld/jw-12-2000/jw-1229-cooltools_p.html (cópia local em pdf: http://w3.ualg.pt/~jmcardo/ensino/PS2003/AulaTP7/jw-1229-cooltools.pdf)

[2] Outros métodos serão apresentados posteriormente.

4 Solução utilizando o JavaCC

1ª parte (escrita de cada um dos números lidos)

```
PARSER_BEGIN(Exemplo)


    // código Java que invoca o parser
    public class Exemplo {

        public static void main(String args[]) throws ParseException {
            // criação do objecto utilizando o constructor com argumento para
            // ler do standard input (teclado)
            Exemplo parser = new Exemplo(System.in);
            Exemplo.Aritm();
        }
}

PARSER_END(Exemplo)

// símbolos que não devem ser considerados na análise
SKIP :
{
    " " | "\t" | "\r"
}

// definição dos tokens (símbolos terminais)
TOKEN :
{
    < INTEGER : (["0" - "9"])+ >
    | < EOL : "\n" >
}

// definição da produção
void Aritm() : {Token t1, t2;}
{
    t1=<INTEGER> {
        System.out.println("1st Integer :"+t1.image);
    }
    ( ("+" | "-") t2=<INTEGER> {
        System.out.println("2nd Integer :"+t2.image);
    }
    )? (<EOF> | <EOL>)
}
```

1ª parte (escrita de cada um dos números lidos e do sinal)

```
PARSER_BEGIN(Exemplo)


    // código Java que invoca o parser
    public class Exemplo {

        public static void main(String args[]) throws ParseException {
```

```
            // criação do objecto utilizando o constructor com argumento para
            // ler do standard input (teclado)
            Exemplo parser = new Exemplo(System.in);
            Exemplo.Aritm();
        }
}

PARSER_END(Exemplo)

// símbolos que não devem ser considerados na análise
SKIP :
{
   " " | "\t" | "\r"
}

// definição dos tokens (símbolos terminais)
TOKEN :
{
   < INTEGER : (["0" - "9"])+ >
   | < EOL : "\n" >
   | < SIGN : ("-" | "+") >
}

// definição da produção
void Aritm() : {Token t1, t2, t3;}
{
   t1=<INTEGER> {
      System.out.println("1st Integer :"+t1.image);
   }
   ( t3=<SIGN> {  // outra forma t3=<("-" | "+")> s/ def. de um novo token
      System.out.println("signal :"+t3.image);
   }
   t2=<INTEGER> {
      System.out.println("2nd Integer :"+t2.image);
   }
   )? (<EOF> | <EOL>)
}
```

Exercício 2.1:

(a) Criar o ficheiro RecNumberJCC.jj

(b) Gerar o analisador: javacc RecNumberJCC.jj (é criado o programa do reconhecedor em Java: RecNumberJCC.java)

(c) Depois disto deve-se compilar a classe e temos os bytecodes Java do reconhecedor.

Ficheiro RecNumber.jj:

```
PARSER_BEGIN(RecNumberJCC)

public class RecNumberJCC {

  public static void main(String args[]) throws ParseException {
    RecNumberJCC parser = new RecNumberJCC(System.in);

    try {
```

```
        parser.Input();
    } catch (TokenMgrError x) {
        System.out.println("Unrecognized number");
    }
  }

}

PARSER_END(RecNumberJCC)

TOKEN :
{
        < EOL: "\n" >
        | < T_END: "\r" >
}

TOKEN :
{
        < T_INTEGER: ( ["0"-"9"] )+ >
        | < T_REAL: ( ["0"-"9"] )* "." ( ["0"-"9"] )+ >
}

void Input() :
{}
{
  <T_INTEGER> ( <EOL> | <T_END> | <EOF> )
        {
                System.out.println("Integer number");
        }
  | <T_REAL> ( <EOL> | <T_END> | <EOF> )
        {
                System.out.println("Real number");
        }
}
```

Ficheiro RecNumber.jj com analisador que mostra no ecrã o valor do número reconhecido:

```
PARSER_BEGIN(RecNumberJCC)

public class RecNumberJCC {

  public static void main(String args[]) throws ParseException {
    RecNumberJCC parser = new RecNumberJCC(System.in);

    try {
        parser.Input();
    } catch (TokenMgrError x) {
        System.out.println("Unrecognized number");
    }
  }

}

PARSER_END(RecNumberJCC)

TOKEN :
{
        < EOL: "\n" >
        | < T_END: "\r" >
}

TOKEN :
{
```

```
        < T_INTEGER: ( ["0"-"9"] )+ >
        | < T_REAL: ( ["0"-"9"] )* "." ( ["0"-"9"] )+ >
}

void Input() :
{Token x;}
{
  (x = < T_INTEGER>) ( <EOL> | <T_END> | <EOF> )
        {
                System.out.println("Integer number: "+Integer.valueOf(x.image));
        }
  | (x = <T_REAL>) ( <EOL> | <T_END> | <EOF> )
        {
                System.out.println("Real number: "+Double.valueOf(x.image));
        }
}
```