**Compiladores**

**Aula Prática 3**

***Objetivos***

* Inserir Semântica,
* Entender o que são gramáticas de atributos,
* Escrever o primeiro interpretador e
* Escrever o primeiro compilador

**Tarefa 1 – Criar um diretório de trabalho e copiar kit.**

Descompacte o arquivo lab3.zip. Nele, você tem os arquivos necessários para criar compiladores em COCO/R: Coco.jar, scanner.frame, parser.frame e Compile.java. Este kit tem seis sub-diretórios, cada um para uma das tarefas práticas a seguir (tarefas 2 a 5).

**Tarefa 2 – Semântica e Atributos**

Escreva uma gramática em COCO para substituir palavras em frases. As frases são dadas em linhas com a seguinte sintaxe: padrão a procurar (P), nova palavra (S), frase (F). P, S e F são separadas por um espaço, como no exemplo abaixo:

rosa margarida uma rosa eh uma rosa e nada mais

muito pouco Isso nao eh realmente muito provavel

Para o exemplo, a saída do seu compilador deve ser:

uma margarida eh uma margarida e nada mais

Isso nao eh realmente pouco provavel

Forneça soluções com e sem gramáticas de atributos.

**Tarefa 3 – Semântica aritmética**

Modifique a gramática abaixo para retornar o valor das expressões de entrada:

COMPILER Calc

CHARACTERS

digito = "0123456789".

TOKENS

numero = digito {digito}.

IGNORE '\n' + '\r' + '\t'

PRODUCTIONS

Calc = {"calc" Exp} EOF.

Exp = numero { "+" numero }.

END Calc.

Modifique a sua solução para suportar subtração. Use uma gramática de atributos.

**Tarefa 4 – Um Interpretador**

Modifique a gramática abaixo para retornar o valor das expressões. Use uma gramática de atributos. Não esqueça de lidar com divisão por zero.

COMPILER Calc

CHARACTERS

digito = "0123456789".

TOKENS

numero = digito {digito}.

IGNORE '\n' + '\r' + '\t'

PRODUCTIONS

Calc = Exp EOF .

Exp = ["-"] T { ( "+" T | "-" T ) } .

T = F { ( "\*" F | "/" F ) } .

F = numero | "(" Exp ")" | "abs" "(" Exp ")".

END Calc.

Teste sua gramática usando o seguinte código para Compile.java:

import java.io.\*;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Compile

{

public static void main(String argv[])

{

String str;

System.out.println("Calculadora (digite 's' para sair)");

System.out.println("Exemplos de expressoes: abs(-1), 2+5-3\*4, 3\*(1+4)");

while (true) {

try {

System.out.print("? ");

BufferedReader bufferRead = new

BufferedReader(new InputStreamReader(System.in));

str = bufferRead.readLine();

if (str.equalsIgnoreCase("s"))

System.exit(0);

InputStream stream = new

ByteArrayInputStream(str.getBytes("UTF-8"));

Scanner s = new Scanner(stream);

Parser p = new Parser(s);

p.Parse();

} catch(IOException e) {

e.printStackTrace();

}

}

}

}

Estenda a gramática para suportar função fatorial e números reais.

**Tarefa 5 – Um Compilador**

Escreveva uma gramática em COCO que traduza formulários dados como no formato abaixo:

|  |
| --- |
| #  # Exemplo de formulário sobre curso de esporte  # em escola do ensino medio  #  TEXTO "Qual o seu nome?"  ESCOLHAUMA "Qual o seu sexo?"  ("Feminino", "Masculino")  ESCOLHAVARIAS "Que esportes voce praticou esse ano?"  ("Futebol", "Voley", "Basquete", "Handebol", "Outros")  ESCOLHAUMA "O que voce achou do curso este ano?"  ("Fraco", "Regular", "Bom", "Excelente")  TEXTO "O que voce sugere que seja diferente no proximo ano?" |

Para HTML:

|  |
| --- |
| <html>  <head><title>Formulario</title></head>  <body>  <form>  Qual o seu nome?<br>  <textarea name='Q0' cols='50' rows='3'></textarea><br>  <br>    Qual o seu sexo?<br>  <input type='radio' name='Q1' value='Feminino'>Feminino<br>  <input type='radio' name='Q1' value='Masculino'>Masculino<br>  <br>    Que esportes voce praticou esse ano?<br>  <input type='checkbox' name='Q2' value='Futebol'>Futebol<br>  <input type='checkbox' name='Q2' value='Voley'>Voley<br>  <input type='checkbox' name='Q2' value='Basquete'>Basquete<br>  <input type='checkbox' name='Q2' value='Handebol'>Handebol<br>  <input type='checkbox' name='Q2' value='Outros'>Outros<br>  <br>  O que voce achou do curso este ano?<br>  <input type='radio' name='Q3' value='Fraco'>Fraco<br>  <input type='radio' name='Q3' value='Regular'>Regular<br>  <input type='radio' name='Q3' value='Bom'>Bom<br>  <input type='radio' name='Q3' value='Excelente'>Excelente<br>  <br>  O que voce sugere que seja diferente no proximo ano?<br>  <textarea name='Q4' cols='50' rows='3'></textarea><br>  <br>  </form>  </body>  </html> |

**Algumas consideração práticas sobre Coco/R**

*Convenções de nomes com JAVA e COCO*

* Não use nomes de diretórios com espaços.
* Use nomes curtos para símbolos (ex, CIf em lugar de ComandoIf);
* Lembre que o nome da gramática é o nome do arquivo e deve aparecer após COMPILER, como a primeira regras e depois de END;
* Gramáticas devem ser salvas em arquivos com extensão atg.
* Armazene todos os arquivos associados a uma gramática em um mesmo subdiretório

*Checagem de Erros*

Checagem de erros em COCO/R se dá em dois estágios. O primeiro está relacionado com erros de sintaxe – como esquecer o ponto depois da produção. Esses são fáceis de corrigir. O segundo corresponde a não terminais que não são definidos, ou não podem ser alcançados, ou são cíclicos, inúteis ou que não satisfazem as restrições LL(1). Nós só vamos nos concentrar em restrições LL1 mais tarde e, pra esse laboratório, o jeito é torcer pra elas não serem muito chatas. A forma mais comum de violar LL1 é ter alternativas para um não terminal que inicia com a mesma string. Assim, um parser LL1 não é capaz de decidir que alternativa seguir, como no exemplo:

assignment = variableName ":=" expression

| variableName index ":=" expression.

index = "[" subscript "]".

Ambas alternativas para assignment começam com variableName. Contudo, é fácil escrever produções que não tem esse problema:

assignment = variableName [ index ] ":=" expression.

index = "[" subscript "]".

Note também que qualquer expressão escrita usando recursão pela esqueda, como abaixo:

expression = term | expression "," term.

violam LL(1) e deveriam ser re-escritas como:

expression = term {"," term}.

Por enquanto, se você encontrar restrições deste tipo, as mostre ao professor que pode tentar ajuda-los em resolvê-las.

**Respostas**

*Semântica em vermelho, para facilitar a leitura*

**Tarefa 2**

*Solução 1, simples*

COMPILER Subs

CHARACTERS

letter = 'A'..'Z' + 'a'..'z'.

TOKENS

EOL = '\n'.

word = letter { letter }.

IGNORE '\t' + '\r'

PRODUCTIONS

Subs

= Line { Line }

.

Line (. String p = "", s = "", w = ""; .)

= word (. p = t.val; .)

word (. s = t.val; .)

word (. if (t.val.equals(p)) System.out.print(s);

else System.out.print(t.val); .)

{ word (. if (t.val.equals(p)) System.out.print(" " + s);

else System.out.print(" " + t.val); .)

}

EOL (. System.out.print("\n"); .)

.

END Subs.

*Solução 2, com uma função*

/\* gramatica substitution \*/

import java.io.\*;

COMPILER Subs1

public String substitution(String p, String s, String target)

{

if (target.equals(p))

return s;

else

return target;

}

CHARACTERS

letter = 'A'..'Z' + 'a'..'z'.

TOKENS

EOL = '\n'.

word = letter { letter }.

IGNORE '\t' + '\r'

PRODUCTIONS

Subs1

= Line { Line }

.

Line (. String p = "", s = ""; .)

= word (. p = t.val; .)

word (. s = t.val; .)

word (. System.out.print(substitution(p, s, t.val)); .)

{ word (. System.out.print(" " + substitution(p, s, t.val)); .)

}

EOL (. System.out.print("\n"); .)

.

END Subs1.

*Solução 3, solução com atributos*

/\* gramatica substitution \*/

import java.io.\*;

COMPILER Subs2

public String substitution(String p, String s, String target)

{

if (target.equals(p))

return s;

else

return target;

}

CHARACTERS

letter = 'A'..'Z' + 'a'..'z'.

TOKENS

EOL = '\n'.

word = letter { letter }.

IGNORE '\t' + '\r'

PRODUCTIONS

Subs2

= Line { Line }

.

Line (. String p = "", s = "", nova = ""; .)

= word (. p = t.val; .)

word (. s = t.val; .)

Frase<out nova, p, s>

(. System.out.println(nova); .)

EOL

.

Frase<out String nova, String p, String s>

= word (. nova = substitution(p, s, t.val); .)

{ word (. nova += " " + substitution(p, s, t.val); .)

}

.

END Subs2.

**Tarefa 3**

*Solução 1, sem atributos (neste exemplo, a expressão é re-escrita na saída)*

COMPILER Calc

CHARACTERS

digito = "0123456789".

TOKENS

numero = digito {digito}.

IGNORE '\n' + '\r' + '\t'

PRODUCTIONS

Calc = { "calc" Exp } EOF.

Exp (. int n; String s; .)

= numero (.

s = t.val;

n = Integer.parseInt(t.val);

.)

{"+" numero (.

s = s + "+" + t.val;

n += Integer.parseInt(t.val);

.)

} (. System.out.println(s + " = " + n); .)

.

END Calc.

*Solução 2, com atributos e subtração, sem re-escrita na saída da expressão*

COMPILER Calc

CHARACTERS

digito = "0123456789".

TOKENS

numero = digito {digito}.

IGNORE '\n' + '\r' + '\t'

PRODUCTIONS

Calc (. int n = 0; .)

= {"calc"

Exp<out n> (. System.out.println(n); .)

}

EOF

.

Exp<out int n>

= numero (. n = Integer.parseInt(t.val); .)

{( "+" numero (. n += Integer.parseInt(t.val); .)

| "-" numero (. n -= Integer.parseInt(t.val); .)

)

}

.

END Calc.

**Tarefa 4**

COMPILER Calc

CHARACTERS

digito = "0123456789".

TOKENS

numero = digito {digito}.

IGNORE '\n' + '\r' + '\t'

PRODUCTIONS

Calc (. int n; .)

= Exp<out n>

EOF (. System.out.println(" " + n); .)

.

Exp<out int v> (. int n; int sinal = 1; .)

= ["-" (. sinal = -1; .)

]

T<out n> (. v = sinal \* n; .)

{ ( "+" T<out n> (. v += n; .)

| "-" T<out n> (. v -= n; .)

)

}

.

T<out int v> (. int n; .)

= F<out n> (. v = n; .)

{ ( "\*" F<out n> (. v \*= n; .)

| "/" F<out n> (. if (n != 0)

v /= n;

else {

System.out.println(

" ERRO: divisao por zero.");

}

.)

)

}

.

F<out int v> (. int n; v = 0; .)

= numero (. v = Integer.parseInt(t.val); .)

| "("

Exp<out n> (. v = n; .)

")"

| "abs" "("

Exp<out n> (. v = n; if (n < 0) v \*= -1; .)

")"

.

END Calc.

**Tarefa 5**

*Solução incompleta apenas suportando Radio buttons. Consegue completar o código?*

import java.io.\*;

import java.util.ArrayList;

class HtmlOutput {

String conteudo;

int count;

public HtmlOutput()

{

count = 0;

conteudo = "<html>\n";

conteudo += "<head><title>Formulario</title></head>\n";

conteudo += "<body>\n<form>\n";

}

public void addRadio(String s, ArrayList al)

{

conteudo += s + "<br>\n";

for (Object val: al) {

conteudo += "<input type='radio' name='Q" + count + "' ";

conteudo += "value='" + val + "'>" + val + "<br>\n";

}

conteudo += "<br>\n\n";

count ++;

}

public void print()

{

conteudo += "</form>\n</body>\n";

conteudo += "</html>\n";

System.out.print(conteudo);

}

}

COMPILER Form

HtmlOutput html;

CHARACTERS

aspas = '"'.

semAspas = ANY - aspas.

TOKENS

string = aspas { semAspas } aspas.

COMMENTS FROM '#' TO '\n'

IGNORE '\n' + '\r' + '\t'

PRODUCTIONS

Form (. html = new HtmlOutput(); .)

= Questao

{ Questao

} (. html.print(); .)

.

Questao = QTexto | QUma | QVarias.

QTexto = "TEXTO" string.

QUma (. String s; ArrayList al; .)

= "ESCOLHAUMA"

Str<out s>

Opcoes<out al> (. html.addRadio(s, al); .)

.

QVarias (. String s; ArrayList al; .)

= "ESCOLHAVARIAS"

Str<out s>

Opcoes<out al>

.

Opcoes<out ArrayList v> (. String s; .)

= "(" Str<out s> (. v = new ArrayList();

v.add(s); .)

{ "," Str<out s> (. v.add(s); .)

} ")"

.

Str<out String s>

= string (. s = t.val.substring(1, t.val.length()-1); .)

.

END Form.