**Compiladores**

**Aula Prática 1**

***Objetivos***

* Se familiarizar com aplicações simples do Coco/R e sua sintaxe e
* Escrever gramáticas que descrevem características simples de linguagens.
* Como checar gramáticas em Coco/R e compilar fontes para elas.

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

Descompacte o arquivo lab1.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 quatro sub-diretórios, cada um para uma das tarefas práticas a seguir (tarefas 2 a 5).

**Tarefa 2 - Calculadora**

Entre no diretório tarefa1. Observe a gramática dada em sala de aula:

calc.atg.

Compile a gramática dada e a teste usando o arquivo de entrada exps1.txt.

java –jar ../Coco.jar –frames .. calc.atg

javac \*.java

java Compile exps1.txt

Modifique a gramática dada para suportar:

1. calc2.atg: subtração, divisão e parênteses. Test com exps2.txt;
2. calc3.atg: números reais do tipo **3.4**, **3.** e **.4**. Test com exps3.txt;
3. calc4.atg: funções abs() e fatorial. Test com exps4.txt;

**Tarefa 3 – Família**

Uma família pode ser descrita em um formato como no exemplo abaixo:

Sobrenome: Silva

Pais: Pedro Henrique, Eva

Filhos: Paulo David, Carlos Márcio, Helen, Ana Maria

Avós: João Lima Silva (falecido) [ pai de Pedro Henrique ], Ana Silva, João Freire, Maria Cristina Freire [ mãe de Eva ]

Netos: Ian Fábio [ filho de Paulo David ]

Outros: 1 carro pequeno, 1 casa, 2 gatos

Escreva uma gramática que descreva famílias como esta. Note que alguns dos membros podem ter falecido. Assuma que as várias seções podem vir em qualquer ordem. Suporte diferentes tipos de nomes tais como O'Toole, McGregor ou Ribeiro-Neto.

**Tarefa 4 – Música**

Uma letra de música (com suas notas) pode ser descrita em um formato como no exemplo abaixo:

C

Atirei o pau no gato-to

Db G

Mas o gato-to

C C

Não morreu-reu-reu

F F#

Dona Chica-ca

C

Admirou-se-se

G

Do berro

C

Do berro que o gato deu: Miau!

Uma canção pode ser formada por qualquer número de linhas e cada linha é escrita em duas partes, (a) uma parte para as notas musicais (C, C# ou Db, D, D# ou Eb, E, F, F# ou Gb, G, G# ou Ab, A, A# ou Bb, B) e é terminada por um fim de linha; e (a) uma para as palavras que podem ter qualquer número de letras e que terminam com fim de linha.

**Tarefa 5 – Expandido nossa primeira linguagem**

Dada a gramática da linguagem SIMPLE vista em sala de aula, elimine a ambiguidade na descrição de ident.

Depois, expanda a gramática para suportar:

5.1) Novos operadores e números

* Operador %.
* Números não podem iniciar com 0, a não ser que o número seja o próprio 0.
* Suporte a números em forma hexadecimal (0x21ef, 0XfffF).
* Tipo booleano com constantes true e false;

5.2) Novas instruções

* Suporte a instrução do-while com break e continue.
* Instruções de incremento e decremento (não como expressões!).
* Laço for como em Python (for i in 1..10 ...).
* Cláusula elif (equivale a else if).

Nota: Nunca se apresse em escrever a primeira gramática que atende o exemplo. Pense sobre os tipos de exemplos que poderiam ser atendidos e como você poderia escrever regras mais gerais.

Dica: Estes exercícios não são sobre semântica. Portanto, não se preocupe com isso agora.

**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 esquerda, como abaixo:

expression = term | expression "," term.

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

expression = term {"," term}.

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