Compiladores

Trabalho Prático T2

17 de abril de 2016

1 Objetivo

O objetivo deste trabalho é a criação de um analisador sintático (*parser*) como segundo componente do *front-end* de um compilador.

1.1 A LP de entrada

A linguagem de entrada padrão que o seu compilador deve aceitar é o G-Portugol (veja o manual da linguagem no AVA). Você é livre para escolher uma outra linguagem de entrada (e.g., Pascal, etc) mas nesse caso converse com o professor antes de começar para ver se a LP que você escolheu é razoável para este projeto.
Obs.: Toda a documentação do professor vai assumir que a linguagem de entrada é o G-Portugol.

2 Descrição do Problema

Você deverá implementar um *parser* para programas escritos em G-Portugol e deverá OBRIGATORIAMENTE integrar o seu *scanner* (desenvolvido no trabalho T1) ao seu *parser*.

Observações MUITO importantes:

- Para realizar a integração *scanner* + *parser* é bastante provável que você tenha de modificar alguma parte do código que foi entregue em T1. Isso não é um problema, na verdade é encorajado. O objetivo final aqui é ter um bloco funcional de código que realiza a análise léxica e sintática.
- Você deve implementar a gramática do G-Portugol descrita no manual da linguagem. A descrição já está quase no formato usual de entrada das ferramentas de criação de *parsers*, o ponto chave é garantir que a gramática implementada esteja livre de ambiguidades (sem mensagens de erro ou *warnings* ao compilar).
- O trabalho é composto de duas partes:
  1. Criação da gramática da G-Portugol no `bison` ou ANTLR.
  2. Emissão de código no formato `.dot` (detalhes adiante) para visualização da árvore de derivação (*parse tree*).

2.1 Testando o trabalho

O trabalho será testado da seguinte maneira para um trabalho gerado com o `flex`:

```
./trab2 < entrada.gpt
```

onde `entrada.gpt` é o arquivo com o programa na linguagem G-Portugol que deve ser analisado.

Caso o seu trabalho tenha sido feito no ANTLR a execução deve ser:

```
java Trab2 entrada.gpt
```

Exemplo de um arquivo de entrada (`teste.gpt`):

```
algoritmo teste;
inicio
    x := 42 + 3;
    f := 0 <= 1;
fim
```

IMPORTANTE: note que somente do ponto de vista sintático o programa acima está correto, embora ele não seja um programa G-Portugol válido pois está faltando a declaração das variáveis. Esse problema só será tratado no próximo trabalho, quando for feita a análise semântica.

Executando `./trab2 < teste.gpt`, teremos como resposta no terminal:

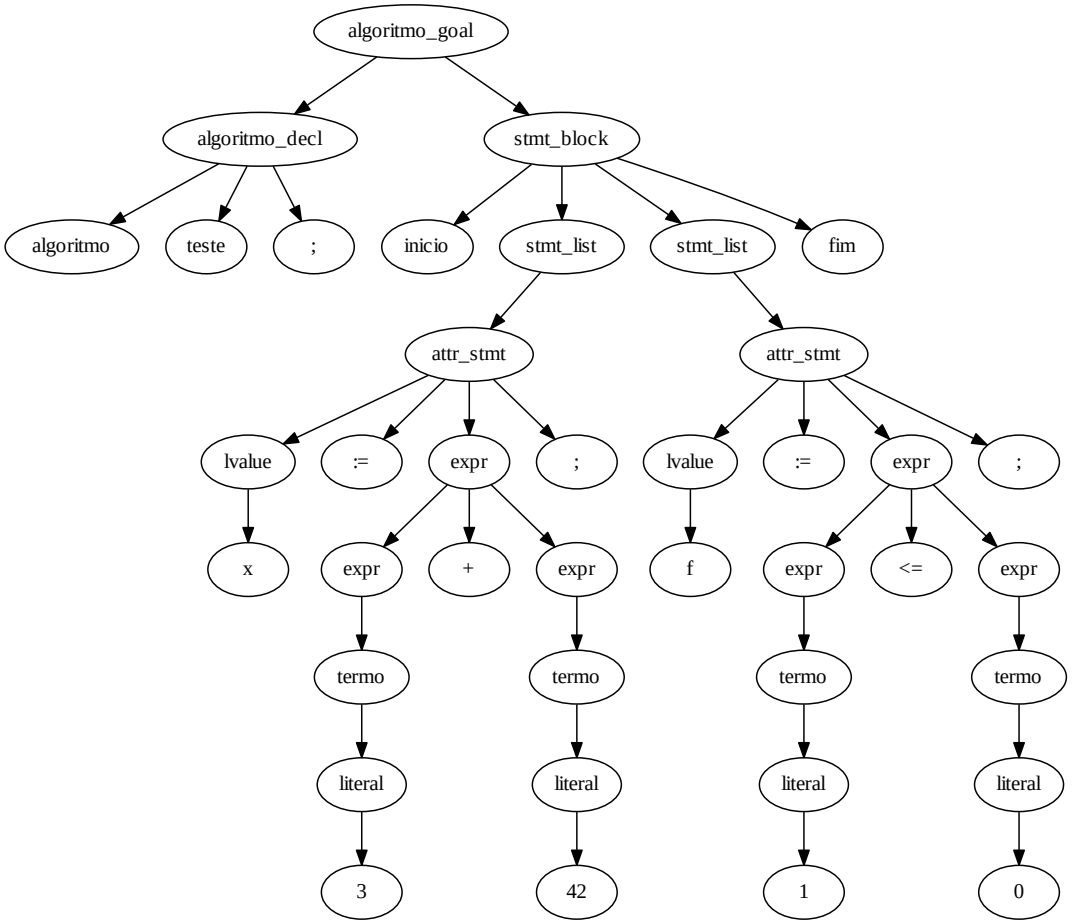
```
digraph {
graph [ordering="out"];
node0[label="algoritmo_decl"];
node1[label="algoritmo"];
node0 -> node1;
node2[label="teste"];
node0 -> node2;
node3[label=";"];
node0 -> node3;
node4[label="lvalue"];
node5[label="x"];
node4 -> node5;
node6[label="literal"];
node7[label="42"];
node6 -> node7;
node8[label="termo"];
node8 -> node6;
node9[label="expr"];
node9 -> node8;
node10[label="literal"];
node11[label="3"];
node10 -> node11;
node12[label="termo"];
node12 -> node10;
node13[label="expr"];
node13 -> node12;
node14[label="expr"];
node14 -> node13;
node15[label="+"];
node14 -> node15;
node14 -> node9;
node16[label="attr_stmt"];
node16 -> node4;
node17[label=":="];
node16 -> node17;
node16 -> node14;
node18[label=";"];
node16 -> node18;
node19[label="stmt_list"];
node19 -> node16;
node20[label="lvalue"];
node21[label="f"];
node20 -> node21;
node22[label="literal"];
node23[label="0"];
```

```
node22 -> node23;
node24[label="termo"];
node24 -> node22;
node25[label="expr"];
node25 -> node24;
node26[label="literal"];
node27[label="1"];
node26 -> node27;
node28[label="termo"];
node28 -> node26;
node29[label="expr"];
node29 -> node28;
node30[label="expr"];
node30 -> node29;
node31[label="<="];
node30 -> node31;
node30 -> node25;
node32[label="attr_stmt"];
node32 -> node20;
node33[label=":="];
node32 -> node33;
node32 -> node30;
node34[label=";"];
node32 -> node34;
node35[label="stmt_list"];
node35 -> node32;
node36[label="stmt_block"];
node37[label="inicio"];
node36 -> node37;
node36 -> node19;
node36 -> node35;
node38[label="fim"];
node36 -> node38;
node39[label="algoritmo_goal"];
node39 -> node0;
node39 -> node36;
}
```

A saída exibida acima corresponde uma representação textual da árvore de derivação escrita na linguagem `.dot` da ferramenta GraphViz. Se salvarmos a saída para um arquivo `tree.dot` e executarmos:

```
dot -Tpdf tree.dot -o tree.pdf
```

O resultado será um arquivo PDF com a figura abaixo:



Observações importantes sobre a figura acima:

- O seu *parser* não precisa emitir um código idêntico ao exibido aqui para o exemplo. O objetivo não é replicar o resultado do professor, é simplesmente tem uma forma simples de visualizar a árvore de derivação do programa de entrada. Embora os *parsers* construídos com o `bison` não construam de fato uma árvore de derivação, esse exercício é uma preparação interessante para o próximo trabalho de análise semântica.
- Você é livre para definir o nomes e rótulos dos nós da árvore de derivação e quais elementos você quer incluir na árvore. O seu *parser* deve minimamente construir árvores de derivação para as possíveis expressões da linguagem. O objetivo aqui é ter uma forma de enxergar o resultado final do trabalho do *parser* então não há uma única forma de fazer. No entanto, quanto mais detalhada (e precisa) a saída da visualização, melhor.

Ao submeter o seu trabalho para correção, além dos códigos-fonte envie um arquivo LEIAME com as instruções para compilar o seu trabalho. Idealmente, a sua submissão deve incluir também um arquivo `Makefile` que gera como executável para o seu *parser* um arquivo de nome `trab2`. *(Se você não sabe usar o `make` mas quer aprender, converse com o professor.)*

3 Regras para Desenvolvimento e Entrega do Trabalho

- **Data da Entrega**: O trabalho deve ser entregue até às 23:55 h do dia 08/05/2016 (Domingo). Não serão aceitos trabalhos após essa data.

- **Grupo**: O trabalho deve ser feito em grupos de 2 alunos. Informe seu grupo para o professor para que ele possa fazer o cadastro no AVA. IMPORTANTE: alunos sem cadastro no AVA ou que não tenham sido incluídos em algum grupo não conseguiram submeter o trabalho.

- **Linguagem de Programação e Ferramentas**: Para implementar o seu analisador sintático você pode escolher entre usar o `bison` ou o ANTLR.
  Obs. 1: Se você quiser você pode usar outras ferramentas à sua escolha, mas nesse caso você deverá ser capaz de resolver os eventuais problemas que surgirem por conta própria.
  Obs. 2: Se você quiser você pode usar outras ferramentas para visualização no lugar do GraphViz. Só não será aceito como resultado a saída padrão do ANTLR da árvore de derivação.

- **Como entregar**: Pela atividade criada no AVA. Envie um arquivo compactado com todo o seu trabalho. SOMENTE uma pessoa da dupla deve enviar o trabalho no AVA. **Coloque o nome das duas pessoas da dupla no nome do arquivo do trabalho.**

- **Recomendações**: Modularize o seu código adequadamente. Crie códigos claros e organizados. Utilize um estilo de programação consistente. Comente o seu código extensivamente. Não deixe para começar o trabalho na última hora.

4 Avaliação

- O trabalho vale 2.5 pontos na média parcial do semestre.

- A pontuação do trabalho fica dividida da seguinte forma:

  - 2.0 pontos para a correta implementação da gramática do G-Portugol. Isso quer dizer que o seu *parser* deve reconhecer qualquer programa que esteja dentro das regras sintáticas da linguagem e deve também rejeitar qualquer programa que não esteja dentro das regras.
  - 0.5 ponto para emissão de código para visualização da árvore de derivação.

- Trabalhos com erros de compilação receberão nota zero.

- Caso seja detectado plágio (entre alunos ou da internet), todos os envolvidos receberão nota zero.

- Serão levadas em conta, além da correção da saída do seu programa, a clareza e simplicidade de seu código.

- A critério do professor, poderão ser realizadas entrevistas com os alunos, sobre o conteúdo do trabalho entregue. Caso algum aluno seja convocado para uma entrevista, a nota do trabalho será dependente do desempenho na entrevista. (Vide item sobre plágio, acima.)