Trabalho 1 – Parser Descendente Recursivo

Linguagens Formais e Autômatos – Bach. Sistemas de Informação

Prof. Dr. Jefferson O. Andrade

Instituto Federal do Espírito Santo (Ifes) – Campus Serra

2019/1

1 Introdução

Este trabalho consiste na implementação de um parser descendente recursivo para uma Linguagem Livre de Contexto, que chamaremos de *MEL*.[1]

O trabalho pode ser desenvolvido em qualquer linguagem escolhida pelo estudante, desde que esta linguagem esteja disponível de modo livre para o sistema operacional Ubuntu Linux 18.04.

2 Enunciado

A linguagem em questão é a linguagem gerada pela gramática dada abaixo:[2]

⟨*expr*⟩ ::= ⟨*term*⟩ ((‘`+`’ | ‘`-`’) ⟨*term*⟩)*

⟨*term*⟩ ::= ⟨*factor*⟩ ((‘`*`’ | ‘`/`’ | ‘`//`’ | ‘`%`’) ⟨*factor*⟩)*

⟨*factor*⟩ ::= ⟨*base*⟩ (‘`^`’ ⟨*factor*⟩)?

⟨*base*⟩ ::= (‘`+`’ | ‘`-`’) ⟨*base*⟩
| ⟨*number*⟩
| ‘`(`’ ⟨*expr*⟩ ‘`)`’

⟨*number*⟩ ::= ⟨*digit*⟩+ ‘`.`’? ⟨*digit*⟩* ((‘`E`’ | ‘`e`’) (‘`+`’ | ‘`-`’)? ⟨*digit*⟩+)?

⟨*digit*⟩ ::= ‘`0`’ | ‘`1`’ | ‘`2`’ | ‘`3`’ | ‘`4`’ | ‘`5`’ | ‘`6`’ | ‘`7`’ | ‘`8`’ | ‘`9`’

Na gramática acima, o símbolo terminal ‘`//`’ representa a operação de divisão inteira, ou o quociente da divisão euclidiana, e o símbolo ‘`%`’ representa o resto da divisão.

O *parser descendente recursivo* derivado da gramática apresentada acima deve ser implementado como um módulo (classe, função, etc.) independente, e deve criar um programa que leia uma expressão da entrada padrão, faça o *parsing* da expressão lida, e imprima o resultado.

Por exemplo, supondo que o nome dado ao programa seja `trab1`, e que ele tenha sido implementado corretamente, o quadro abaixo mostra um exemplo de seção de uso do programa.

[1] Micro Expression Language.
[2] Em notação EBNF.

```
1 $ ./trab1
2 10 * 5 + 100 / 10 - 5 + 7 % 2
3 56
```

No exemplo acima, a linha 1 faz a chamada do programa; a linha 2 contém a expressão aritmética que foi entrada pelo usuário; e a linha 3 contém a resposta que foi gerada pelo programa.

3 Instruções

Algumas definições que devem ser consideradas no desenvolvimento do programa:

- O programa/*parser* deve aceitar (e ignorar) espaços em branco entre símbolos terminais.
- Caso a expressão apresente dois ou mais operadores diferentes, mas de mesma precedência, no mesmo "nível" – como pode ocorrer nas regras para $\langle expr \rangle$ e para $\langle term \rangle$ – deve-se utilizar associatividade à esquerda para resolver a avaliação da expressão. Observe que esta situação não se aplica à regra para $\langle factor \rangle$ que envolve exponenciação.
- O trabalho é de realização individual. Os estudantes podem debater ideias entre si, mas a elaboração do código deve ser um trabalho original de cada estudante.
- O trabalho deve ser entregue **exclusivamente** através do sistema AVA do Ifes na forma de um arquivo compactado em formato ZIP ou 7z na tarefa corresponde indicada na página da disciplina.

  ATENÇÃO: Apenas os formatos de compactação ZIP e 7z serão aceitos. Trabalhos entregue em outros formado de compactação tais como RAR ou LHA serão **anulados**.
- O arquivo compactado entregue deve conter um diretório (pasta) com todo o **código fonte** desenvolvido para o trabalho. A estrutura do deve ser a seguinte:

  ```
  hw01
  ├── Readme.md
  └── source
      ├── build file
      └── ...
  ```

  O arquivo `Readme.md` é um arquivo em formato Markdown contendo ao menos: (a) o nome do autor do trabalho; (b) a descrição da linguagem e ambiente de programação utilizados na elaboração do trabalho; (c) a descrição geral dos arquivos contidos no trabalho; (d) a forma de construção (*build*); (e) e o nome do programas desenvolvidos.

  O diretório `source` deve conter todo o código fonte propriamente dito. A organização interna deste diretório é livre, entretanto, deve haver algum arquivo de *build*, tal como `Makefile`, `build.xml`, script de shell, etc. de acordo com a ferramenta de *build* usada pelo autor. O arquivo de *build* e como ativá-lo deve estar descrito no arquivo *Readme.md*.
- A data de entrega está definida na atividade criada no AVA.