Analisador Sintático Descendente Recursivo

O objetivo desse trabalho é implementar um Analisador Sintático Descendente Recursivo (ASDR) para uma linguagem baseada no Pascal (**PascalLite**). O analisador léxico implementado anteriormente, deverá ser adaptado e estar funcionando corretamente para atender as necessidade do analisador sintático, a interação entre o analisador léxico e o analisador sintático se dará por meio da função `consome()` vista em sala, ou seja, somente a função **`consome()`** fará chamadas à função **`obter_atomo()`** do analisador léxico. Caso o aluno não tenha implementado o analisador léxico isso deverá ser feito em conjunto com o ASDR.

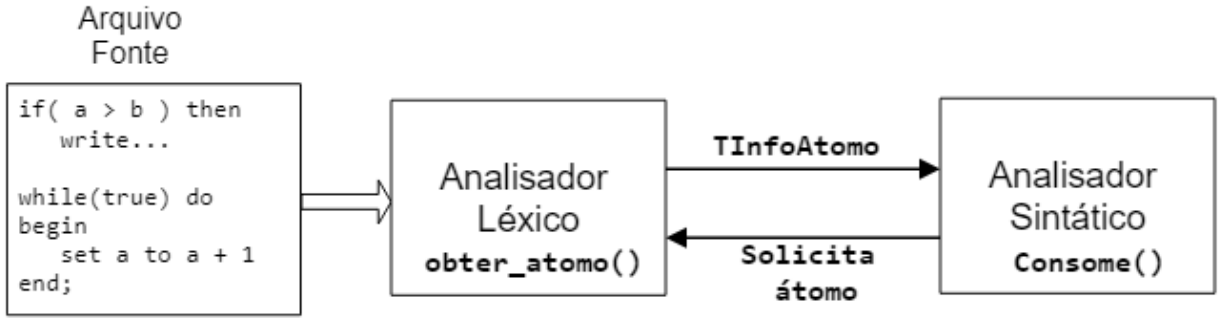

Figura 1: Interação entre Analisador Léxico e Sintático

Na implementação do ASDR quando for detectado um **erro sintático** ou **léxico**, o ASDR deve-se emitir uma mensagem de erro explicativa e terminar a execução do programa. A mensagem explicativa deve informar a linha do erro, o tipo do erro (léxico ou sintático) e caso seja um erro sintático, deve-se informar qual era o átomo esperado e qual foi encontrado pelo ASDR, por exemplo:

Saída do programa:

```
Program exemplo;
begin
    write(maior);
end.

Erro sintático na linha 4: esperado [comando] encontrado [END]
```

A seguir temos um programa em **PascalLite** que lê uma certa quantidade de números inteiros e encontra o maior número.

```
 1 {- programa le uma sequencia de numeros inteiros
 2 e encontra o maior -}
 3 Program exemplo;
 4   number num, maior, cont, qtd;
 5 begin
 6     read(qtd);
 7     set cont to 0;
 8     set maior to 0; // inicializa a variavel maior com 0
 9     while( cont < qtd ) do
10     begin
11       read(num);
12       if( num > maior ) then
13            set maior to num;
14
15       set cont to cont + 1
16     end;
17     write(maior) // imprime o maior valor
18 end.
```

A sintaxe da nossa versão do **PascalLite** será dada na notação BNF estendida, por conveniência, introduziremos mais uma notação **[α]** que é equivalente a **α|λ**, ou seja, indicará que a cadeia **α** é opcional. Os <não-terminais> da gramática são nomes entre parênteses angulares **<** e **>** e os símbolos **terminais** estão em **NEGRITO** (átomos do analisador léxico) ou entre aspas (Ex: **“;”**).

<programa>::= **PROGRAM IDENTIFICADOR** “;” <bloco> “.”

<bloco>::= <declaracao_de_variaveis> <comando_composto>

<declaracao_de_variaveis> ::= {<tipo> <variaveis> **“;”**}

<tipo> ::= **NUMBER** | **CHAR** | **BOOLEAN**

<variaveis> ::= **IDENTIFICADOR** {“,” **IDENTIFICADOR** }

<comando_composto> ::= **BEGIN** <comando> {“;”<comando>} **END**

```
<comando> ::= <comando_atribuicao>  |
              <comando_condicional> |
              <comando_repeticao>   |
              <comando_entrada>     |
              <comando_saida>       |
              <comando_composto>
```

<comando_atribuicao> ::= **SET IDENTIFICADOR TO** <expressão>

```
<comando_condicional> ::= IF “(” <expressao> “)” THEN
                                <comando> [ELSE <comando>]
```

<comando_repeticao> ::= **WHILE** “(” <expressao> “)” **DO** <comando>

<comando_entrada> ::= **READ** “(“ <variaveis> “)”

<comando_saida> ::= **WRITE** “(“ <expressao> { “,” <expressao> } “)”

<expressao> ::= <expressao_simples> [**OP_RELACIONAL** <expressao_simples>]

<expressao_simples> ::= [“+” | “–”] <termo> { <operador_simples> <termo> }

<operador_simples> ::= “+” | “–” | **OR**

<termo> ::= <fator> { <operador_termo> <fator> }

<operador_termo> ::= “*” | “/” | **MOD** | **AND**

```
<fator> ::= IDENTIFICADOR |
            NUMERO        |
            CARACTERE     |
            TRUE          |
            FALSE         |
            NOT <fator>   |
            “(” <expressao> “)”
```

O programa entregue será avaliado de acordo com os seguintes itens:

- Funcionamento do programa, ou seja, programas com erros de compilação e não executando receberão nota 0 (zero);
- O programa deve estar na linguagem **C** e testados no compilador do **CodeBlocks**, caso programa apresentarem *warning* ao serem compilados serão penalizados;
- Após a execução o programa deve finalizar com retorno igual a 0,
- O quão fiel é o programa quanto à descrição do enunciado;
- Clareza e organização, programas com código confuso (linhas longas, variáveis com nomes não-significativos, etc.) e desorganizado (sem indentação, sem comentários, etc.) também serão penalizados; e
- Este trabalho pode ser desenvolvido em grupos de até **2 alunos** e sigam as **Orientações para Desenvolvimento de Trabalhos Práticos** disponível no **Moodle**.