Compilador fase 1: Análise léxica e sintática

O objetivo desse trabalho é implementar as fases de análise léxica e sintática de um compilador para uma linguagem baseada na **linguagem Pascal,** denominada **Pascal+-**. O compilador para **Pascal+-** restringe a **linguagem Pascal** para ter apenas tipos **inteiros (`integer`)** e **lógicos** (**`boolean`**), comandos condicionais (**`if`**) e repetição (**`for`**) e não implementa a declaração e chamadas de funções, a exceção se faz para as funções de entrada (**`read`**) e saída (**`write`**).

Na implementação do compilador o analisador léxico deve atender as demandas do analisador sintático. A interação entre o analisador léxico e o analisador sintático se dá por meio da função **`consome()`** (**analisador sintático**) que realizará chamadas à função **`obter_atomo()`** (**analisador léxico**). O **nome do arquivo -fonte é informado por linha de comando** ao compilador.

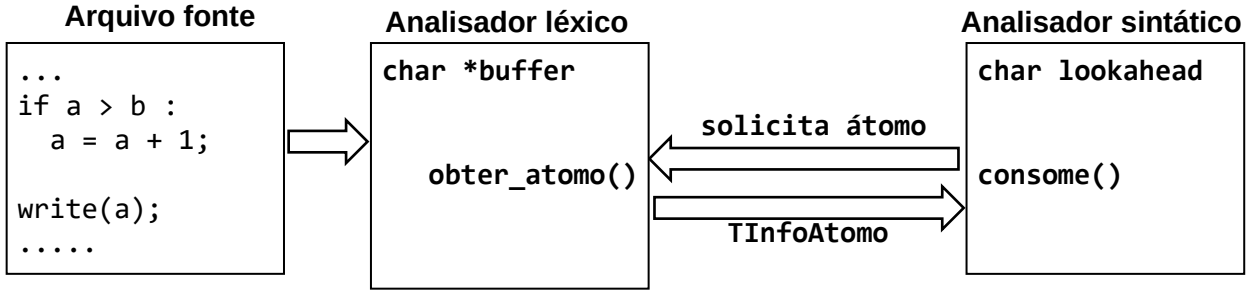

Figura 1: Interação entre Analisador Léxico e Sintático

A seguir são apresentadas a gramática da linguagem **Pascal+**, que deve ser seguida rigorosamente, e as especificações léxicas da linguagem, onde são definidos os átomos da linguagem.

Gramática da linguagem Pascal+-

A **sintaxe** da linguagem **Pascal+-** está descrita na notação **EBNF**, os `<não-terminais>` da gramática são nomes entre parênteses angulares **<** e **>** e os símbolos **`terminais`** (átomos do analisador léxico) estão em **`negrito`** ou entre aspas (Ex: **“;”**). A notação **{ α }** denotará a repetição da cadeia α zero, uma ou mais vezes **(α^*)** e a construção **[β]** é equivalente a **β|λ**, ou seja, indica que a cadeia **β** é opcional.

```
<programa> ::= program identificador “;” <bloco> “.”

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

<declaracao_de_variaveis> ::= {<tipo> <lista_variavel> “;”}

<tipo> ::= integer | boolean

<lista_variavel> ::= 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 <expressao>

<comando_condicional> ::= if <expressao> “:”
                          <comando> [elif <comando>]

<comando_repeticao> ::= for identificador of <expressão> to <expressão> “:” <comando>
```

```
<comando_entrada> ::= read “(“ <lista_variavel> “)”

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

<expressao> ::= <expressao_logica> { or <expressao_logica> }

<expressao_logica>::= <expressao_relacional> { and <expressao_relacional> }

<expressão_relacional> ::= <expressao_simples> [<op_relacional> <expressao_simples>]

<op_relacional> ::= “<” | “<=” | “=” | “/=” | “>” | “>=”

<expressao_simples> ::= <termo> { (“+” | “–” ) <termo> }

<termo> ::= <fator> { ( “*” | “/” ) <fator> }

<fator> ::= identificador |
            numero        |
            true          |
            false         |
            not <fator>   |
            “(“ <expressao> “)”
```

Especificação Léxica

- **Caracteres Delimitadores**: Os caracteres delimitadores: espaços em branco, quebra de linhas, tabulação e retorno de carro (**‘ ’**, **‘\n’**, **‘\t’**, **‘\r’**) deverão ser eliminados (ignorados) pelo analisador léxico, mas o controle de linha (contagem de linha) deverá ser mantido.

- **Comentários**: Existem dois tipos de comentário, um começando com **‘#’** e indo até o final da linha (1 linha) com o finalizador do comentário o caractere **‘\n’**. O outro começando com **“{-”** e terminando com **“-}”** (várias linhas), nesse comentário é importante que a contagem de linha seja mantida, além disso os comentários são repassados para o analisador sintático para serem reportados e descartados.

- **Identificadores**: Os identificadores começam com uma letra em minúsculo, em seguida pode vir zero ou mais letras minúsculas, *underline* **‘_’** ou dígitos, limitados a 15 caracteres. Caso seja encontrado um identificador com mais de 15 caracteres deve ser retornado **ERRO** pelo analisador léxico. A seguir a definição regular para **identificadores**.

  letra → a|b|...|z|
  digito → 0|1|...|9
  identificador → letra(letra|digito|_)*

  Importante: Na saída do compilador, para átomo **identificador**, **deverá ser impresso o lexema que gerou o átomo**, ou seja, a sequência de caracteres reconhecida.

- **Palavras reservadas**: As palavras reservadas na linguagem **Pascal+-** são **lexemas em minúsculo**: **and**, **begin**, **boolean**, **elif**, **end**, **false**, **for**, **if**, **integer**, **not**, **of**, **or**, **program**, **read**, **set**, **to**, **true**, **write**

  Importante: Uma sugestão é que as palavras reservadas sejam reconhecidas na mesma função que reconhece os **identificadores** e deve ser retornado um **átomo específico para cada palavra reservada** reconhecida.

- **Números**: No compilador teremos somente números inteiros na **notação binária**, com seguinte definição regular abaixo:
  numero $\rightarrow$ 0b(0|1)$^+$

  Importante: Na saída do compilador, para átomo **numero,** deverá ser impresso o **valor numérico na notação decimal do atributo do átomo**, ou seja, o lexema que gerou o átomo convertido para notação decimal.

Execução do Compilador

O compilador deve ler o arquivo fonte, com o nome informado por linha de comando, e informar, na tela do computador, cada um dos átomos reconhecidos no arquivo. Caso seja detectado um **erro léxico ou sintático** o compilador 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 o **átomo encontrado** na análise, veja abaixo um exemplo de saída da execução do compilador para o programa **exemplo1**:

Arquivo fonte de entrada.

```
1  program exemplo1;
2  begin
3      write(maior
4  end.
```

Saída do compilador na tela

```
#  1:program
#  1:identificador | exemplo1
#  1:ponto_virgula
#  2:begin
#  3:write
#  3:identificador | maior
#  3:erro sintatico, esperado [)] encontrado [end]
```

A seguir temos um outro programa em **Pascal+-** que lê uma dois números e encontra o maior, o programa a seguir está correto (léxico e sintático).

Arquivo fonte de entrada.

```
 1  {-
 2  programa le dois numeros inteiros e encontra o maior
 3  -}
 4  program exemplo2;
 5    integer num_1, num_2;
 6    integer maior;
 7  begin
 8    read(num_1);
 9    read(num_2);
10    if num_1 > num_2:
11        maior = num_1
12    elif
13        maior = num_2;
14
15    write(maior) # imprime o maior valor
16  end.
```

Na saída da execução do compilador é impressa uma lista dos átomos reconhecidos e sua respectiva linha, e ao final compilador informa que a análise terminou com sucesso:

Saída do compilador na tela

```
#  1:comentario
#  4:program
#  4:identificador | exemplo2
#  4:ponto_virgula
#  5:integer
#  5:identificador | num_1
#  5:virgula
#  5:identificador | num_2
#  5:ponto_virgula
#  6:integer
#  6:identificador | maior
#  6:ponto_virgula
#  7:begin
..... e por ai vai.........
16 linhas analisadas, programa sintaticamente correto
```

Observações importantes:

O programa deve estar bem documentado e pode ser feito em grupo de até **2 alunos**, não esqueçam de colocar o **nome dos integrantes** do grupo no início do arquivo fonte do trabalho e sigam as **Orientações para Desenvolvimento de Trabalhos Práticos** disponível no **Moodle**.

O trabalho será avaliado de acordo com os seguintes critérios:

Funcionamento do programa:

- Caso o programa não compile ou não execute **será atribuída a nota `0` ao trabalho.**
- Caso programa apresentarem ***warning*** a ser compilado ou não finalize com retorno igual a `0`, será descontado `1.0` (um ponto) por ***warning*** relatado.
- O trabalho deve ser desenvolvido na **linguagem C** e será testado usando o compilador do **MinGW** com **VSCode**, para configurar sua máquina no Windows acesse: https://www.doug.dev.br/2022/Instalacoes-e-configuracoes-para-programar-em-C-usando-o-VS-Code/
- Compile seu programa com o seguinte comando abaixo, considere que o programa fonte do seu compilador seja `compilador.c`:
  `gcc -g -Og -Wall compilador.c -o compilador`

Atendimento a especificação do enunciado:

- O quão fiel é o programa quanto à descrição do enunciado, o seu programa deve seguir a gramática descrita acima e fazer leitura de **programa fonte** armazenado em **arquivo** com o nome informado **por linha de comando.**
- Clareza e organização, programas com código confuso (linhas longas, variáveis com nomes não-significativos, ...) e desorganizado (sem indentação, sem comentários, ...) também serão penalizados.
- Entrega de um arquivo **Readme.txt** explicando até a parte do trabalho que foi feito