Analisador Léxico

O objetivo do trabalho é construir um Analisador Léxico para realizar a análise léxica de um programa escrito uma linguagem baseada no Pascal (**PascalLite**), o Analisador Léxico deve ler um o programa fonte em um arquivo texto e identificar os átomos (tokens) da linguagem.

No compilador, a análise léxica é realiza como uma subrotina do Analisador Sintático, sendo assim, para que seja possível a reutilização do Analisador Léxico pelo Analisador Sintático, deverá ser implementada uma rotina responsável na pela análise léxica no caso a função **`obter_atomo()`**, posteriormente na fase de Análise Sintática está rotina será a interface entre as duas análises, abaixo a declaração da função:

```
TInfoAtomo obter_atomo(void);
```

Para cada átomo encontrado, a rotina **`obter_atomo()`** retorna uma estrutura com informações referentes ao átomo contendo além do átomo encontrado, a linha que gerou o átomo e o seu atributo (caso seja necessário). Além disso o Analisador Léxico faz um controle das linhas do programa fonte e a eliminação dos delimitadores (espaços em branco, tabulação, nova linha e retorno de carro). Considere abaixo um programa escrito em **PascalLite**:

```
 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.
```

O para cada átomo reconhecido o Analisador Léxico imprime as seguintes informações baseado nas informações contidas na estrutura **`TInfoAtomo`**:

```
linha  1:comentario
linha  3:program
linha  3:identificador – atributo:exemplo
linha  3:ponto e virgula
linha  4:number
linha  4:identificador – atributo:num
linha  4:virgula
..............
```

Caso ocorra um erro no reconhecimento de um dos átomos do programa fonte analisado, o seu analisador léxico deve parar a execução com uma mensagem informando a linha onde ocorreu o **ERRO**.

Nas seções a seguir são apresentadas as definições regulares para os átomos que a rotina **obter_atomo()** deve identificar.

1. **Definição regular para átomo IDENTIFICADOR**

   LETRA → a|b|...|z|A|B|...|Z
   DIGITO → 0|1|...|9
   IDENTIFICADOR → LETRA(LETRA|DIGITO|_)*

   As palavras reservadas em uma linguagem de programação não podem ser utilizadas como identificadores, para simplificar a rotina de análise léxica, o reconhecimento das palavras reservadas deve ser realizada na rotina que reconhece identificadores.

   Quando um **IDENTIFICADOR** for reconhecido, este deve ser buscado na **lista de palavras reservadas**; se for encontrado, **deve ser retornado o átomo associado a palavra reservada encontrada**; caso contrário o átomo **IDENTIFICADOR deve ser retornado com o seu atributo** e como atributo do **IDENTIFICADOR** a sequência de caracteres que gerou o átomo (lexema).

   Importante:
   O tamanho do identificador deve ser limitado em **15 caracteres**, caso seja encontrado um identificador maior que o limite o **obter_atomo()** o Analisador Léxico deve retornar **ERRO**. Além disso, conforme a definição regular acima, um identificador pode ter, depois da primeira **LETRA**, o caractere *underline* "_".

   A seguir é apresentada a lista dos átomos que devem ser retornados quando uma das palavras reservadas for reconhecida pela rotina de análise léxica.

Átomos Retornados	Descrição
AND	E lógico
BEGIN	Início de um comando composto
BOOLEAN	Tipo booleano
CHAR	Tipo de dado para definir um caractere
DO	Usado no bloco de estrutura de repetição
ELSE	Caso contrário da instrução de seleção
END	Fim de um bloco ou do programa
FALSE	Constante booleana para o valor falso
IF	Determina uma estrutura de condicional
MOD	Operador de resto
NOT	Negação lógica
NUMBER	Tipo de dado para números inteiros e reais
OR	OU lógico
PROGRAM	Início do programa
READ	Define uma função para entrada (leitura)
SET	Define início do comando de atribuição
THEN	Usado no bloco de estrutura condicional
TO	Parte do Comando de atribuição
TRUE	Constante booleana para o valor verdadeiro
WHILE	Determina um laço com condição no início
WRITE	Define uma função para saída (impressão)

O analisador léxico **não é sensível ao caso**, ou seja, as Palavras Reservadas e os **IDENTIFICADORES** podem ser informados com caracteres maiúsculos ou minúsculos, sendo assim, para o lexema "**ProGraM**" ou "**PROGRAM**" temos uma palavra reservada e o átomo retornado é **PROGRAM** sem atributos.

2. **Definição dos átomos simples**

ABRE_PAR → (
FECHA_PAR →)
PONTO → .
PONTO_VIRGULA → ;
VIRGULA → ,

Toda vez que for reconhecido um dos símbolos a rotina **obter_atomo()** retorna o átomo correspondente a definição regular do símbolo**. Esses átomos não possuem atributo**.

3. **Operadores Aritméticos**

SUBTRACAO → -
ADICAO → +
DIVISAO → /
MULTIPLICACAO → *

Toda vez que for reconhecido um dos símbolos acima a rotina **obter_atomo()** retorna o átomo correspondente ao operador aritmético**. Esses átomos não possuem atributo**.

4. **Operadores Relacionais**

ME → <
MEI → <=
IG → =
DI → /=
MA → >
MAI → >=

Toda vez que for reconhecido um dos símbolos acima a rotina **obter_atomo()** retorna o átomo **OP_RELACIONAL** e como atributo desse átomo uma constante (**enumeração**) representando o operador relacional.

5. **Comentários na Linguagem**

NOVA_LINHA → caracter de nova linha(=13) ou retorno de carro(=10)
ASCII → qualquer caractere ASCII – NOVA_LINHA

COMENTARIO_VARIAS_LINHAS → {- (ASCII | NOVA_LINHA)* -}
COMENTARIO_UMA_LINHA → //ASCII*

COMENTARIO → **COMENTARIO_VARIAS_LINHAS** | **COMENTARIO_UMA_LINHA**

Para o comentário da linguagem é retornado o átomo **COMENTARIO**, importante é que o controle de linha é mantido dentro do comentário. Para o **COMENTARIO_VARIAS_LINHAS** verifique se o comentário é fechado com o caractere **'-}'** caso não seja o programa deverá retornar o átomo **ERRO**, e é claro, o átomo **COMENTARIO não possui atributo**.

6. Constantes numéricas

6.1. Constantes com valores inteiros e reais

EXPONENCIAL → e(+|-|λ)DIGITO^{+}

NUMERO → DIGITOS^{+}(EXPONENCIAL|λ)

A rotina **obter_atomo()** retorna o átomo **NUMERO** e o **seu valor numérico como atributo**. Lembrando que o símbolo λ representa o símbolo vazio (palavra vazia), assim os símbolos + ou - são opcionais. Podemos ter como exemplo de **NUMERO** o lexema **"10"**, **"10e2"**, **"10e-1"** ou **"22e+1"**.

6.2. Constante caractere

CARACTERE → 'qualquer caractere ASCII'

A rotina **obter_atomo()** retorna o átomo **CARACTERE**, ou seja, qualquer caractere da tabela ASCII entre apóstrofos ('), para esse átomo é retorno como **atributo** o caractere que gerou o átomo.

7. Erros

Para qualquer outro caractere estranho, que não faça parte do alfabeto do Analisador léxico, ou seja, caracteres que não foram usados nas definições regulares e não seja delimitadores de texto (espaços em branco, tabulação, nova linha, retorno de carro e finalizador de string) a rotina **obter_atomo()** deverá devolver o átomo **ERRO** e o Analisador Léxico será finalizado.

8. Orientações para entrega do projeto

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**.