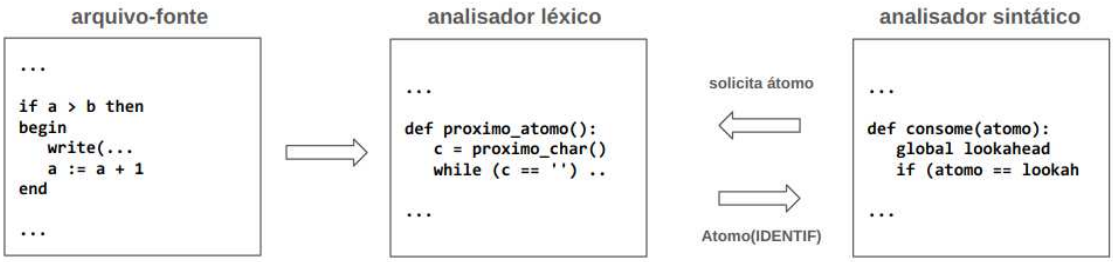
Faculdade Impacta de Tecnologia
Compiladores

Compilador fase 1: Análise léxica e sintática

O objetivo desse trabalho é implementar um Compilador com as fases de análise léxica e sintática para uma linguagem baseada na linguagem Pascal, denominada PascalLite. O Compilador para PascalLite restringe a linguagem Pascal para ter apenas tipos inteiros (integer) e lógicos (boolean), comandos condicionais (if) e repetição (while). Não implementaremos a declaração e chamadas de funções nessa linguagem, a exceção se faz as funções de entrada (read) e saída (write) implementadas de forma modificada.

Na implementação do Compilador o analisador léxico deve atender as necessidades do analisador sintático. A interação entre o analisador léxico e o analisador sintático se dará por meio da função consome() (do analisador sintático) que realizará chamadas à função obter_atomo() (do analisador léxico).



Gramática da linguagem PascalLite

A sintaxe da linguagem PascalLite 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. A notação $\{\alpha\}$ denotará a repetição da cadeia α zero, uma ou mais vezes (α^*) e a construção $[\beta]$ é equivalente a $\beta|\varepsilon$, ou seja, indica que a cadeia β é opcional.

<programa> ::= **program identificador [(** <lista de identificadores> **)] ;** <bloco>**.**

<bloco> ::= [<declarações de variáveis>] <comando composto>

<declarações de variáveis> ::= **var** <declaração> {**;** <declaração> }**;**

<declaração> ::= <lista de identificadores> **:** <tipo>

<lista de identificadores> ::= **identificador** { **,** **identificador** }

<tipo> ::= **integer** | **boolean**

<comando composto> ::= **begin** <comando> { **;** <comando> } **end**

<comando> ::=
<atribuicao> |
<comando de entrada> |
<comando de saída> |
<comando if> |
<comando while> |
<comando composto>

<atribuição> ::= **identificador** := <expressao> ;

<comando if> ::= **if** <expressao> **then** <comando>
[**else** <comando>]

<comando while> ::= **while** <expressao> **do** <comando>

<comando de entrada> ::= **read (** <lista de identificadores> **)**

<comando de saida> ::= **write (** <expressao> { **,** <expressao> } **)**

<expressao> ::= <expressao simples> [<operador relacional> <expressao simples>]

<operador relacional> ::= < | <= | = | <> | > | >=

<operador de adição> ::= + | − | **or**

<expressao simples> ::= [+ | −] <termo> { <operador de adição> <termo> }

<termo> ::= <fator> { <operador de multiplicação> <fator> }

<operador de multiplicação> ::= ∗ | / | **div** | **mod** | **and**

<fator> ::=
identificador |
numero |
(<expressao> **)** |
true |
false |
not <fator>

1 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**: 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). E ainda, há um outro tipo de comentário de blocos começando por `{` e terminado com `}`. 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.

- **Palavras reservadas**: As palavras reservadas na linguagem **PascalLite** são **begin**, **boolean**, **div**, **do**, **else**, **end**, **false**, **if**, **integer**, **mod**, **program**, **read**, **then**, **true**, **not**, **var**, **while**, **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.

- **Identificadores**: Os identificadores podem iniciar, opcionalmente, por caractere underline (`'_'`) seguido de letra minúscula ou maiúscula, seguido de zero ou mais letras minúsculas e/ou maiúsculas, dígitos ou caractere `'_'`, limitados a 20 caracteres. Caso seja encontrado um identificador com mais de 20 caracteres deve ser retornado **ERRO** pelo analisador léxico. A seguir a definição regular para **identificadores**.

  $letra \rightarrow a \mid b \mid \ldots \mid z \mid A \mid B \mid \ldots \mid Z \mid \_$

  $digito \rightarrow 0 \mid 1 \mid \ldots \mid 9$

  $identificador \rightarrow letra\ (letra \mid digito\ )^*$

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

- Números: No compilador teremos apenas números inteiros, com a seguinte definição regular abaixo:

  $digito \rightarrow 0 \mid 1 \mid \ldots \mid 9$

  $numero \rightarrow digito+$

2 Execução do Compilador

No Compilador quando for detectado um erro sintático ou léxico, o analisador 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 a linha do erro e qual era o átomo esperado e qual foi o átomo encontrado pelo Compilador, por exemplo para o programa exemplo1:

Entrada no compilador:

```
01.   program exemplo1;
02.   var num: integer;
03.   begin
04.      num := 10
05.      write(num);
06.   end.
```

Saída:

```
Linha: 1 - atomo: PROGRAM              lexema: program
Linha: 1 - atomo: IDENTIF              lexema: ex01
Linha: 1 - atomo: PONTO_VIRG           lexema: ;
Linha: 2 - atomo: VAR                  lexema: var
Linha: 2 - atomo: IDENTIF              lexema: num
Linha: 2 - atomo: DOIS_PONTOS          lexema: :
Linha: 2 - atomo: INTEGER              lexema: integer
Linha: 2 - atomo: PONTO_VIRG           lexema: ;
Linha: 3 - atomo: BEGIN                lexema: begin
Linha: 4 - atomo: IDENTIF              lexema: num
Linha: 4 - atomo: ATRIB                lexema: :=
Linha: 4 - atomo: NUM                  lexema: 10              valor: 10
Erro sintático: Esperado [END   ] encontrado [WRITE   ] na linha 1
```

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

```
01. (*programa le dois números
02. inteiros e encontra o maior*)
03.
04. program ex02;
05. var num1, num2: integer;
06.    _maior : integer;
07. begin
08.    read(num1, num2);
09.    if num1 > num2 then
10.       _maior := n1
11.    else
12.       _maior := n2;
13.    write(_maior)
14. end.
```

O para cada átomo reconhecido o compilador imprime as seguintes informações baseado nas informações contidas na estrutura **Atomo**, e ao final informa que a análise terminou com sucesso:

Saída:

```
Linha: 4 - atomo: PROGRAM              lexema: program
Linha: 4 - atomo: IDENTIF              lexema: ex02
Linha: 4 - atomo: PONTO_VIRG           lexema: ;
Linha: 5 - atomo: VAR                  lexema: var
Linha: 5 - atomo: IDENTIF              lexema: num1
Linha: 5 - atomo: VIRGULA              lexema: ,
Linha: 5 - atomo: IDENTIF              lexema: num2
Linha: 5 - atomo: DOIS_PONTOS          lexema: :
Linha: 5 - atomo: INTEGER              lexema: integer
Linha: 5 - atomo: PONTO_VIRG           lexema: ;
Linha: 6 - atomo: IDENTIF              lexema: _maior
Linha: 6 - atomo: DOIS_PONTOS          lexema: :
Linha: 6 - atomo: INTEGER              lexema: integer
Linha: 6 - atomo: PONTO_VIRG           lexema: ;
Linha: 7 - atomo: BEGIN                lexema: begin
Linha: 8 - atomo: READ                 lexema: read
...
14 linhas analisadas, programa sintaticamente correto.
```

Observações importantes: O programa deve estar bem documentado e pode ser feito em grupo de até 5 alunos, não esqueçam de colocar o nome dos integrantes do grupo no arquivo fonte do trabalho.

O trabalho será avaliado de acordo com os seguintes critérios:

1. Funcionamento do programa.
2. O trabalho deve ser implementado utilizando a linguagem python.
3. O quão fiel é o programa quanto à descrição do enunciado, principalmente ao formato do arquivo de entrada;
4. Clareza e organização, programas com código confuso (linhas longas, variáveis com nomes não-significativos, ...) e desorganizado (sem comentários, ...) também serão penalizados.