INSTITUTO FEDERAL DE MINAS GERAIS – IFMG - CAMPUS FORMIGA.
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ESPECIFICAÇÃO TÉCNICA DO COMPILADOR DA LINGUAGEM P

PEDRO DANIEL
LUCAS GABRIEL

FORMIGA
2019

1. INTRODUÇÃO

Um compilador é um programa que interpreta códigos que são escritos a partir de uma determinada linguagem de computador que são chamados de código fonte, criando uma serie de sequência que o mesmo necessita fazer para que o processo seja executado com sucesso.

O objetivo deste trabalho é o projeto e a implementação dos analisadores léxico, sintático, tabela de símbolos e tratamento de erros para a linguagem de programação P, gerada pela gramática dada. O trabalho foi implementado em Python 3.

O programa implementado recebe o arquivo a ser analisado como parâmetro na linha de comando e disponibilizar uma opção que permita gravar a tabela de símbolos em um arquivo texto.

1.1. Descrição do Projeto

Python é uma linguagem simples e fácil de trabalhar, por isso foi escolhida para o desenvolvimento deste compilador.

O projeto consiste da seguinte forma: dado como entrada um arquivo txt simbolizando um código na linguagem P, o programa deve ser capaz de analisar o mesmo e retornar a tabela de símbolos, ou retornar o erro no código se for o caso.

1.2. Informações extras sobre o Projeto

Como sabemos, o formalismo empregado pelo analisador léxico é o das Expressões Regulares. Tendo em vista isso, foi implementado uma função do analisador léxico que é retornar uma estrutura que contém 3 campos: token, lexema e tipo.

Sobre o analisador sintático, ele recebe uma cadeia de tokens do analisador léxico, também é projetado para emitir mensagens para quaisquer erros de sintaxe.

A junção dessas 2 etapas consiste neste projeto final, que recebe um arquivo escrito em linguagem hipotética (denominada de linguagem P) e compila o mesmo.

1.3. Execução do código

Para que o arquivo possa ser rodado, o seguinte comando deve ser executado:

>python <codigo.py> <arquivo_de_entrada.txt> -t <nome_arquivo_saida.txt>

Exemplo:

```
python AnalisadorSintatico.py exemplo2.txt -t saida1.txt
```

2. Especificação da linguagem P

2.1. Elementos Léxicos

O desenvolvimento do compilador foi feito através destes elementos léxicos.

a) Símbolos Especiais: PVIRG (;); DPONTOS (:); VIRG (,); ABREPAR((); FECHAPAR()); ABRECH({); FECHACH(});

b) Palavras reservadas: PROGRAMA, VARIAVEIS, INTEIRO, REAL, LOGICO, CARACTER, SE, SENAO, ENQUANTO, LEIA, ESCREVA, FALSO, VERDADEIRO.

c) Identificadores: identificadores válidos possuem, no máximo, 32 caracteres. Identificadores da linguagem de programação P começam com letra seguidos de quaisquer sequências de letras e/ou dígitos, identificadores são expressos pela seguinte expressão regular letra(letra|dígito) *. Identificadores são representados por TOKENS da classe ID. A linguagem de programação P não é case sensitive.

d) Operadores: OPREL (=, <, >, <=, >=, <); OPAD (+, -); OPMUL (*, /); OPNEG (!);

e) Números Inteiros e reais: constantes numéricas inteiras são formadas por sequências de dígitos, ou seja, são representadas pela seguinte expressão regular dígito(dígito)*. Constantes numéricas reais usam o '.' como separador e são representadas pela seguinte expressão regular dígito(dígito)*. (dígito)*. Constantes numéricas inteiras e reais são representadas por TOKENS da classe CTE. Dígitos pertencem ao intervalo [0-9];

f) Literais: cadeias de caracteres são formadas por quaisquer sequências de caracteres delimitados por aspas duplas. Cadeias de caracteres são representas por TOKENS da classe CADEIA. Letras pertencem ao intervalo [A-Z a-z];

g) Comentários: A linguagem de programação P admite comentários de linha e bloco, conforme especificados na linguagem de programação C.

h) Outras informações; os tipos básicos da linguagem de programação P são: INTEIRO, REAL, LOGICO, CARACTER. O comando de atribuição é definido pelos símbolos := e é representado por TOKENS da classe ATRIB;

2.2. Elementos de Sintaxe

Os aspectos sintáticos da linguagem são definidos pela gramática livre de contexto G1 a seguir:

G1 = {{A, PROG, DECLS, C-COMP, LIST-DECLS, DECL-TIPO, D, LIST-ID, E, TIPO, LISTACOMANDOS, G, COMANDOS, IF, WHILE, READ, ATRIB, WRITE, EXPR, H, LIST-W, L, ELEM-W, SIMPLES, P, R, TERMO, S, FAT} {programa, id, variáveis, inteiro, real, logico, caracter, abrepar, fechapar, se, abrech, fechach, senao, enquanto, leia, atrib, escreva, cadeia, cte, verdadeiro, falso, oprel, opad, opmul, opneg, pvirg, virg, dpontos}, P, PROG}

<A*> ->	<PROG> 'fim-de-arquivo';
<PROG> ->	'programa' 'id' 'pvirg' <DECLS> <C-COMP>;
<DECLS> ->	& | 'variáveis' <LISTDECLS>;
<LISTDECLS> ->	<DECL-TIPO> <D>;
<D> ->	& | <LISTDECLS>;
<DECL-TIPO> ->	<LIST-ID> 'dpontos' <TIPO> 'pvirg';
<LIST-ID> ->	'id' <E>;
<E> ->	& | 'virg' <LIST-ID>;
<TIPO> ->	'inteiro' | 'real' | 'logico' | 'caracter';
<C-COMP> ->	'abrech' <LISTA-COMANDOS> 'fechach';
<LISTA-COMANDOS> ->	<COMANDOS> <G>;
<G> ->	& | <LISTA-COMANDOS>;
<COMANDOS> ->	<IF> | <WHILE> | <READ> | <WRITE> | <ATRIB>;
<IF> ->	'se' 'abrepar' <EXPR> 'fechapar' <C-COMP> <H>;
<H> ->	& | 'senao' <C-COMP>;
<WHILE> ->	'enquanto' 'abrepar' <EXPR> 'fechapar' <C-COMP>;
<READ> ->	'leia' 'abrepar' <LIST-ID> 'fechapar' 'pvirg';
<ATRIB> ->	'id' 'atrib' <EXPR> 'pvirg';
<WRITE> ->	'escreva' 'abrepar' <LIST-W> 'fechapar' 'pvirg';
<LIST-W> ->	<ELEM-W> <L>;
<L> ->	& | 'virg' <LIST-W>;
<ELEM-W> ->	<EXPR> | 'cadeia';
<EXPR> ->	<SIMPLES> <P>;
<P> ->	& | 'oprel' <SIMPLES>;
<SIMPLES> ->	<TERMO> <R>;
<R> ->	& | 'opad' <SIMPLES>;
<TERMO> ->	<FAT> <S>;
<S> ->	& | 'opmul' <TERMO>;
<FAT> ->	'id' | 'cte' | 'abrepar' <EXPR> 'fechapar' | 'verdadeiro' | 'falso' | 'opneg' <FAT>;

2.3. First And Follow Sets

Variable	First Set	Follow Set
<A>	programa	$
<PROG>	programa	fim-de-arquivo
<DECLS>	λ variáveis	abrech
<LISTDECLS>	id	abrech
<D>	λ id	abrech
<DECL-TIPO>	id	abrech id
<LIST-ID>	id	dpontos fechapar
<E>	λ virg	dpontos fechapar
<TIPO>	caracter inteiro logico real	pvirg
<C-COMP>	abrech	enquanto escreva fechach fim-de-arquivo id leia se senao
<LISTA-COMANDOS>	enquanto escreva id leia se	fechach
<G>	λ enquanto escreva id leia se	fechach
<COMANDOS>	enquanto escreva id leia se	enquanto escreva fechach id leia se
<IF>	se	enquanto escreva fechach id leia se
<H>	λ senao	enquanto escreva fechach id leia se
<WHILE>	enquanto	enquanto escreva fechach id leia se
<READ>	leia	enquanto escreva fechach id leia se
<ATRIB>	id	enquanto escreva fechach id leia se
<WRITE>	escreva	enquanto escreva fechach id leia se
<LIST-W>	abrepar cadeia cte falso id opneg verdadeiro	fechapar
<L>	λ virg	fechapar
<ELEM-W>	abrepar cadeia cte falso id opneg verdadeiro	fechapar virg
<EXPR>	abrepar cte falso id opneg verdadeiro	fechapar pvirg virg
<P>	λ oprel	fechapar pvirg virg
<SIMPLES>	abrepar cte falso id opneg verdadeiro	fechapar oprel pvirg virg
<R>	λ opad	fechapar oprel pvirg virg
<TERMO>	abrepar cte falso id opneg verdadeiro	fechapar opad oprel pvirg virg
<S>	λ opmul	fechapar opad oprel pvirg virg
<FAT>	abrepar cte falso id opneg verdadeiro	opad opmul

3. Desenvolvimento

3.1. Analisador Léxico

O analisador Léxico foi o primeiro a ser implementado, com os tokens definidos por uma tupla, contendo um identificador (numero) e uma string (o token).

É definido pela classe Token seu tipo (TipoToken), seu lexema (o que foi lido) e a linha em que foi lido, então entra de fato no analisador léxico.

No analisador léxico o arquivo é aberto, então lido, caractere por caractere até eles formarem um token, isso se repete até o arquivo ter sido completamente lido. Nessa parte são tratados dígitos, letras, caracteres especiais, comentários ou inválidos. O léxico retorna todas as informações do arquivo que o sintático necessita.

Funcionamento geral;

- Ler o código fonte;
- Agrupar caracteres em itens léxicos (TOKENS)
- Ignorar espaços brancos e comentários;
- Detectar e diagnosticar erros léxicos

class TipoToken		Classe que define os tokens
class Token		Classe que define as características do token
class Léxico		Parte principal do analisador léxico
	def __init__(self, nomeArquivo)	Função que recebe o nome do arquivo
	def abreArquivo(self)	Função que abre o arquivo
	def fechaArquivo(self)	Função que fecha o arquivo
	def getChar(self)	Função que ler o arquivo, caracter por caracter
	def getToken(self)	Função que fica lendo caracteres até eles formarem um token

3.2. Analisador sintático

O analisador sintático depende do analisador léxico para funcionar, pois este recebe os tokens do mesmo. O sintático passa por todo o arquivo, verificando a corretude do mesmo, e retornando mensagens de erro quando é o caso.

Ele verifica se o objeto de análise atual é uma palavra reservada, um ID ou um token normal, Verifica se o token lido é igual ao esperado, Consome o token atual, e chama o próximo

Funcionamento geral;

- Agrupar TOKENS em estruturas sintáticas (expressões, comandos, declarações, etc.)
- Verificar se a sintaxe da linguagem na qual o programa foi escrito está sendo respeitada.
- Detectar e diagnosticar erros sintáticos

class Sintático	Classe principal do analisador sintático
def interprete(self, nomeArquivo)	Inicia o analisador sintático
def atualIgual(self, token)	Verifica se o token lido é igual ao esperado
def consome(self, token)	Consome o token atual, e chama o próximo
def A(self)	Não terminal Inicial
def PROG(self)	Não terminal "Main"
def DECLS(self)	Não terminal das declarações
def LIST_DECLS(self)	Não terminal responsável pela estrutura das variáveis
def D(self)	Terminal que verifica se há mais outra variável
def DECL_TIPO(self)	Não terminal responsável pela estrutura das variáveis
def LIST_ID(self)	Não terminal responsável por ler um identificador
def E(self)	Não terminal responsável por verificar se os ID acabaram, ou se mais um foi declarado
def TIPO(self)	Não terminal responsável por verificar a declaração da variável
def C_COMP(self)	Não terminal que verifica a estrutura dos comandos
def LISTA_COMANDOS(self)	Não terminal que verifica se um comando foi declarado
def G(self)	Não terminal que verifica se mais algum comando foi declarado
def COMANDOS(self)	Não terminal que verifica qual comando foi declarado
def IF(self)	Não terminal responsável pela estrutura do tipo SE
def H(self)	Não terminal que verifica se o SENAO foi declarado
def WHILE(self)	Não terminal responsável pela declaração ENQUANTO
def READ(self)	Não terminal responsável pela declaração LEIA
def ATRIB(self)	Não terminal responsável pela atribuição
def WRITE(self)	Não terminal responsável pela declaração ESCRITA
def LIST_W(self)	Não terminal responsável pela estrutura da escrita
def L(self)	Não terminal que verifica se mais alguma cadeia necessita ser escrita
def ELEM_W(self)	Não terminal que verifica o que será escrito
def EXPR(self)	Não terminal que lê uma serie de números e operações
def P(self)	Não terminal que verifica se houve uma declaração do tipo relacional
def SIMPLES(self)	Não terminal que lê uma série de números e operações
def R(self)	Não terminal que verifica se houve uma declaração de adição/subtração
def TERMO(self)	Não terminal que lê uma série de números e operações
def S(self)	Não terminal que verifica se houve uma declaração de multiplicação/divisão
def FAT(self)	Não terminal que lê os números e operações

3.3. Exemplo de arquivo de entrada

```
1.  //Este programa tem um comentário de bloco aberto na linha 13 e não
2.  //fechado e identificadores inválidos nas linhas 11 e 12
3.  PROGRAMA exemplo2;
4.  VARIAVEIS x,y: INTEIRO;
5.          c:  CARACTER;
6.          r:  REAL;
7.          b:  LOGICO;
8.  {
9.     ESCREVA("Digite um número:");
10.    LEIA(x);
11.    y:=çx;
12.    #b:=VERDADEIRO;
13.    ENQUANTO(b)  /* isto eh apenas um exemplo que contém erros
14.    {
15.       SE(y>10)
16.
17.          ESCREVA(y);
18.       }
19.       SENAO
20.       {
21.          y:=y+3;
22.          b:=FALSO;
23.       }
24.    }
25. }
```

3.4. Saída

```
(venv) C:\Users\pedro\Downloads\Aula\6º Semestre\Compiladores\compiladores1-trabalho1-2019-2\Compilador>python AnalisadorSintatico.py exemplo2.tx
t -t saida1.txt
ERRO DE SINTAXE [linha 11]: foi recebido "caracter(es) invalido(s)"
ERRO DE SINTAXE [linha 12]: foi recebido "<#>"
ERRO DE SINTAXE [linha 13]: foi recebido "comentario nao fechado"
----- FIM - DA - ANALISE -----

(venv) C:\Users\pedro\Downloads\Aula\6º Semestre\Compiladores\compiladores1-trabalho1-2019-2\Compilador>
```

```
(venv) C:\Users\pedro\Downloads\Aula\6º Semestre\Compiladores\compiladores1-trabalho1-2019-2\Compilador>python AnalisadorSintatico.py exemplo1.txt
----- FIM - DA - ANALISE -----

(venv) C:\Users\pedro\Downloads\Aula\6º Semestre\Compiladores\compiladores1-trabalho1-2019-2\Compilador>python AnalisadorSintatico.py exemplo4.txt
ERRO DE SINTAXE [linha 3]: Variaveis nao podem começar com numeros.
ERRO DE SINTAXE [linha 20]: era esperado "{"
ERRO DE SINTAXE [linha 20]: foi recebido "y"
----- FIM - DA - ANALISE -----

(venv) C:\Users\pedro\Downloads\Aula\6º Semestre\Compiladores\compiladores1-trabalho1-2019-2\Compilador>python AnalisadorSintatico.py exemplo5.txt
ERRO DE SINTAXE [linha 3]: Variaveis nao podem começar com numeros.
ERRO DE SINTAXE [linha 3]: foi recebido "comentario nao fechado"
----- FIM - DA - ANALISE -----

(venv) C:\Users\pedro\Downloads\Aula\6º Semestre\Compiladores\compiladores1-trabalho1-2019-2\Compilador>python AnalisadorSintatico.py exemplo6.txt
ERRO DE SINTAXE [linha 5]: foi recebido "b"
ERRO DE SINTAXE [linha 20]: foi recebido "b"
----- FIM - DA - ANALISE -----
```

3.5. Tabela de Símbolos

{'programa': (18, 'programa'), 'variáveis': (19, 'variáveis'), 'inteiro': (20, 'inteiro'), 'real': (21, 'real'), 'logico': (22, 'logico'), 'caracter': (23, 'caracter'), 'se': (24, 'se'), 'senao': (25, 'senao'), 'enquanto': (26, 'enquanto'), 'leia': (27, 'leia'), 'escreva': (28, 'escreva'), 'falso': (29, 'falso'), 'verdadeiro': (30, 'verdadeiro'), 'exemplo2': (18, 'programa'), ' "x" "y"': (20, 'inteiro'), ' "c"': (23, 'caracter'), ' "r"': (21, 'real'), ' "b"': (22, 'logico')}