1.0 Documentação descritiva

1.1 Analisador léxico

1.1.1 Informações gerais

O analisador está descrito em linguagem python, inicialmente importa a biblioteca “RE” para auxiliar na manipulação de expressões regulares. O módulo RE oferece operações para o trabalho com regex (“concatenação”, regular expressions).

1.1.2 Código fonte do Analisador

TOKENS: Auxilia na definição ou padronização geral de ‘tokens’ que serão analisados, armazena as **palavras** reservadas da linguagem analisada

TYPE_VALUE: padronizar componentes de acordo com a composição da palavra analisada. ‘NUM’ atribuído de símbolos numéricos e ‘id’ armazenando strings iniciadas por valores alfabéticos podendo conter sufixo numérico.

RULES: estabelece as regras do analisador, ou seja, padroniza as regex que serão analisadas.

from token import Token:classe de onde são importados os métodos da análise,

def __init__(self, code, rules, tks_reserved):

def endCode(self): Boolean

Realiza a verificação se o processo de análise chegou ao fim, ou seja, se todo código foi devidamente percorrido.

def nextToken(self):Token

Este tem por finalidade consumir o arquivo sempre que chamado, validando cada token de acordo com as regras(RULES) pré-definidas.

Classe Token:

def __init__(self, tk_name, tk_type, line):
“construtor” da classe

Gera os métodos de acesso às variáveis “self, tk_name, tk_type, line” e retorna o toString da classe**.**

Analisador Sintático cuca

def __init__(self, lexer):
“construtor da classe”, recebe um objeto do tipo lexer e acessa o método .nextToken(), a leitura é feita ignorando “tab, espaços, newline, comentários"

 # consumes token if is valid
def expectedToken(self, tk):
Utilize o token caso o token seja válido. Caso a execução encontre um “;” ou um componente diferente, a função lança uma exceção

 # compare if token is valid
def compare(self, tk, currentToken):
verifica se o token que está sendo analisado é válido.

 # check if header passed is valid
def checkHead(self, element): Boolean
analisa a validade do cabeçalho passado! verifica se o token atual corresponde a uma regra válida do analisador ou corresponde a um token reservado da linguagem/ analisador.

 # method for consume of the tokens from lexer
def consumesToken(self, tk): Boolean
Verifica se o token corrente é correspondente a um token das regras do analisador.
atualiza o token corrente ignorando token do tipo comentário, quebra de linha, tabs e espaços.

 # add variables in cache
def addVar(self, v): Boolean
adiciona os tokens em um array temporário - cache.

 # search for declarations of variables

def searchVar(self, v): Dict/None.

pesquisa - variáveis/tokens - n array variável(no cache).

 # search for ocurrences of the IDENTIFIER

def checkOcurrences(self, v): Dict/None

recebe um array de regras do analisador como parâmetro e verifica se o token corresponde a uma delas.

 # consume TYPES OF VARIABLE CHAR, BOOLEAN, INTEGER

def simpleType(self):

consome os tokens correspondente aos tipos de variáveis "CHAR, BOOLEAN, INTEGER" da linguagem pascal.

 # consumes token types CONSTANT

def constant(self): consome os tokens correspondentes a constantes.

 # variable

def variable(self): Consome tokens correspondentes a declaração de variáveis.

 # rule of characters

def characterConstant(self): Consome tokens correspondentes a declaração de constantes de caracteres.

 # rule of integer constant

def integerConstant(self): consome os tokens correspondentes a declarações de constantes inteiras.

 # consume TYPES OF VARIABLE ARRAY

def typeVarArray(self): consome os tokens correspondentes às variáveis do tipo Array(Vetor).

 # declaration of the variable TYPE ARRAY

def indexRange(self): Consome os tokens correspondente ao tamanho do Array.

 # consumes TYPES OF VARIABLES declared

def consumesTypeVars(self):

executa uma ação de acordo com o tipo da variável - chama uma regra correspondente ao tipo

 # program

def program(self):

verifica se o código passado possui o escopo “program”. função principal para validar a sintaxe de um programa em “pascalzin”

 # block

def block(self):

regra que verifica o bloco de declaração (regras statement, variáveis e funções)

 # consumes block of declaration from variables

def variableDeclarationPart(self):

consome/executa/valida a regra do bloco de declaração de variáveis

 # valid declaration of the variables

def variableDeclaration(self, scope):

valida as variáveis declaradas

 # consumes block of declaration from procedures and functions

def procedureAndFunctionDeclarationPart(self):

consome/executa/valida a regra do bloco de declaração de procedientos

 # valid declaration of the procedures

def procedureDeclaration(self):

verifica e valida os procedimentos declarados

 # valid declaration of functions

def functionDeclaration(self):

verifica e valida as funções declaradas

 # consumes parameters passed in the function

def functionParam(self):

verifica e valida as regras dos parâmetros passados na função

 # consumes block statement of the program

def statementPart(self): verifica as regras do escopo principal do begin até o end

 # valid structure of the statement from application

def compoundStatement(self):

valida a regra a estrutura do escopo da aplicação

 # check statement block

def statement(self):

verifica os estados que determinado bloco pode assumir: ler, escrever, ou escopo begin end.

 # an simple structure [ASSIGNMENT OF VARIABLE, READ OR WRITE]

def simpleStatement(self):

statement com regra de tipo "simples", como ler, escrever, atribuição

 # an block structure [BEGIN, IF OR WHILE]

def structuredStatement(self):

identificar a existência ou não de componentes específicos das estruturas condicionais IF, repetição WHILE, e o início de uma nova statement begin .

 # rule of assignment of variables

def assignmentStatement(self):

valida a regra de atribuição de variáveis.

 # rule of statement READ

def readStatement(self):

valida as regras do boloco/ estado de leitura

 # rule of statement WRITE

def writeStatement(self):

valida as regras do boloco/ estado de escrita

 # rule of statement IF

def ifStatement(self):

valida as regras do bloco/ estado do escopo estrutura de condição if

 # rule of statement WHILE

def whileStatement(self):

validação das regras do boloco/estado da estrutura de repetição While

 # rule of EXPRESSION

def expression(self):

validação de regras de expressão matemática de forma genérica, pode haver uma operação simples ou usando operadores relacionais

 # rule simple expression

def simpleExpression(self):

validação de regras simples de operações básicas (soma, subtração)

 # rule for multiple expressions

def term(self):

Garante a recursividade da análise/consumo dos termos da expressão.

 # rule for Breakdown of mathematical expression

def factor(self):

analisa cada termo da expressão, e de acordo com o token corrente, executa uma regra correspondente. Se for identificador, executa regra do identificador...

 # consumes MINUS OR PLUS

def sign(self):

verifica se o termo da expressão é de soma ou subtração

 # consumes OPERATOR RELATIONAL

def relationalOperator(self):

validação das regras de operadores relacionais

 # consumes OPERATOR MINUS, PLUS or OR

def consumeAddingOperator(self):

validação das regras/cosumir operadores -,+ e or.

 # consumes OPERATOR TIMES, DIV or AND

def consumeMultiplyingOperator(self):

validação das regras/consumir das operações envolvendo multiplicação, divisão, "e".

ANALISADOR SEMÂNTICO

def checkToken(self, dic, token):

função auxiliar para verificar se token existe na tabela de símbolos

 # verify if variable is declared in scope actual

def checkVariableDeclaredID(self, tableSimb, token, scope):

função/regra responsável para verificação de variáveis declaradas no programa

def checkTypeVariable(self, tableSimb, variableAssign, token, scope):

regra para verificar se expressão contém os mesmo tipos de dados

 # return type the variable declared

def type(self, tableSimb, token, scope):

função auxiliar que retorna o tipo de dado se a variável tiver sido declarada

 # return type the return the function

def typeFunction(self, tableSimb, token):

função auxiliar que retorna o tipo de dado se a função tiver sido declarada

 # check if function is declared

def checkFunctionID(self, tableSimb, token, scope):

regra para verificar identificador de função

 # compare type of the variables

def compareType(self, termo1, termo2, type, token):

regra para comparar se dois termos/variáveis são do mesmo tipo de dado.