

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Кафедра вычислительных технологий**

**ОТЧЕТ**

**по практической работе №4**

**«**Построение синтаксического анализатора**»**

по курсу «Методы разработки трансляторов»

Отчет подготовили

студенты 36 группы: студенты 36 группы: А. С. Радченко и Е. В. Малышкин

Отчет принял

зав. каф., проф., док. тех. наук: Ю. М. Вишняков

Краснодар, 2019

Цель работы: разработка программы синтаксического анализатора исходного текста.

Постановка задачи

Программа получает на входе файл - результат лексического анализа и выполняет синтаксический анализ исходной программы. Результатом работы должно быть сообщение о корректности программы или сообщение об первой обнаруженной ошибке с указанием строки и конструкции языка, при разборе которых обнаружена ошибка.

Грамматика исходного языка

Начальный символ грамматики – нетерминал <программа>.

<программа> ::= [<program>]<текст>

<program> ::= program <идентификатор>;

<текст> ::= [обявл. перем.][{<функция>|<процедура>}] begin [тело] end.

<объявл. перем.> ::= var {<строка объявления>}

<строка объявления> ::= <идентификатор>[, <идентификатор>]: <тип>;

<тип> ::= <тип переменной> | array [<интервал] of <тип переменной>

<тип переменной> ::= integer | real | string

<интервал> ::= <число>..<число>[, {число>..<число>}]

<функция> ::= function <идентификатор>[(<строка объявления>)]:<тип переменной>; <объявл. перем.> begin <тело> end;

<процедура> ::= procedure <идентификатор>[(строка объявления)]; <объявл. перем.> begin <тело> end;

<тело> ::= {оператор}

<оператор> ::= <идентификатор> := <выражение>; | <условный оператор> | goto <идентификатор>; | <идентификатор>:

<выражение> ::= <терм>{(+ | -)<терм>}

<терм> ::= <множитель> { (\* | / | ^)<множитель>}

<множитель> ::= <аргумент> | (<выражение>)

<агрумент> ::= <идентификатор> | <константа>

<буква> ::= A | B | C | .. | Y| Z

<цифра> ::= 0 | 1 | 2 | .. | 8 | 9

<условие> ::= <выражение><неравенство><выражение>

<неравенство> ::= < | > | <= | >= | <>

<константа> ::= <число>| <строка>

<число> ::= [-]<цифра>{<цифра}[.[E][-]{<цифра>}]

<строка> ::= ‘ [{<буква><число>}] ’

Описание алгоритма разбора

Программа сожержит в себе функции двух типов: проверки грамматик и служебные функции. Служебные функции позволяют считать следующий токен, вернуть считанный токен и считать предыдущий, а так же вызвать функцию, которая сохраняет сообщение об ошибке.

Функции проверки грамматик построены схожим между собой образом. Они используют функцию чтения следующего токена, а затем при помощи оператора if проверяют, ожидали ли мы встретить считанный токен. Если токен был ожидаем, то функция считывает следующий токен и проверяет его таким же образом. Так повторяется, пока функция не произведёт все операции требующиеся для проверки грамматики. Если прокерки завершились, а ошибки так и не выявлено, функция возвращает пустую строку. В противном случае, функция фозвращает строку с описанием ошибки и вызывает функцию error, которая запишет описание ошибки отдельно. Такие функции способны вызывать другие функции проверки грамматик (вложенные грамматики). В самом начале запускается функция wrapper, которая проверяет грамматику, содержащую в себе описание всей программы. По итогу выполнения этой функции мы либо получим пустую строку - знак успшеной проверки, либо строку с описание ошибки.

Листинг

import sys

import json

import subprocess

def get\_token\_name(token):

global service\_words, operations, separators, constants, identifiers

codes = ['W', 'O', 'R', 'C', 'I']

tables = [service\_words, operations, separators, constants, identifiers]

if token[0] in ['C', 'I']:

return tables[codes.index(token[0])][token[1]]['value']

return tables[codes.index(token[0])][token[1]]

def scan():

global token\_index, nxtsymb, cursymb

token\_index += 1

if token\_index < len(tokens\_chain):

cursymb = tokens\_chain[token\_index-1]

nxtsymb = tokens\_chain[token\_index]

def antiscan():

global token\_index, nxtsymb, cursymb

token\_index -= 1

if token\_index >= 1:

cursymb = tokens\_chain[token\_index-1]

nxtsymb = tokens\_chain[token\_index]

def error(text):

global mistakes

if text[-8:] == 'expected':

if text == '";" expected' and (cursymb[2] != nxtsymb[2]):

mistakes.append(str(text) + ' in the end of line ' + str(cursymb[2]))

return

text += ' instead of "' + get\_token\_name(nxtsymb) + '"'

mistakes.append(str(text) + ' at line ' + str(nxtsymb[2]))

def check(response):

if response:

error(response)

return False

else:

return True

def wrapper():

scan()

mistake = False

while not mistake:

print('PERED', get\_token\_name(nxtsymb), nxtsymb)

mistake = operator()

if mistake:

return mistake

if cursymb[0] == 'I' and get\_token\_name(nxtsymb) == ':':

scan()

else:

print('POSLE:', get\_token\_name(nxtsymb), nxtsymb)

scan()

print('POSLE2:', get\_token\_name(nxtsymb), nxtsymb)

if get\_token\_name(nxtsymb) == ';':

scan()

if get\_token\_name(nxtsymb) == 'stop':

break

def operator():

# print(Fore.GREEN + 'operator' + Style.RESET\_ALL)

print(get\_token\_name(nxtsymb), nxtsymb)

if get\_token\_name(nxtsymb) == ';':

pass

elif nxtsymb[0] == 'I':

scan()

if get\_token\_name(nxtsymb) == ':':

return ''

if get\_token\_name(nxtsymb) in ['[', '(']:

scan()

if not check(expression()):

return 'invalid array arguments'

scan()

while get\_token\_name(nxtsymb) == ',':

scan()

if not check(expression()):

return 'invalid array arguments'

scan()

if get\_token\_name(nxtsymb) not in [']', ')']:

return '"]" expected'

scan()

if get\_token\_name(nxtsymb) == '=':

if not check(assignment()):

return 'invalid assignment declaration'

return ''

if get\_token\_name(nxtsymb) in ['(', '[']:

scan()

if not check(expression()):

return 'invalid argument'

scan()

while get\_token\_name(nxtsymb) == ',':

scan()

if not check(expression()):

return 'invalid argument'

scan()

if get\_token\_name(nxtsymb) not in [')', ']']:

return 'expected ")"'

return ''

return 'unexpected "'+get\_token\_name(nxtsymb)+'"'

elif get\_token\_name(nxtsymb) == 'if':

scan()

if not check(condition()):

return 'condition expected'

scan()

if get\_token\_name(nxtsymb) != 'then':

return '"then" expected'

scan()

if not check(operators()):

return 'operators expected'

if get\_token\_name(nxtsymb) == 'end if':

return ''

elif get\_token\_name(nxtsymb) == 'else':

scan()

if not check(operators()):

return 'operators expected'

if get\_token\_name(nxtsymb) == 'end if':

return ''

else:

return '"end if" expected'

else:

return '"else" or "end if" expected'

return ''

elif get\_token\_name(nxtsymb) == 'goto':

scan()

if nxtsymb[0] != 'I':

return 'identifier expected'

return ''

elif get\_token\_name(nxtsymb) == 'while':

scan()

if not check(condition()):

return 'condition expected'

scan()

if get\_token\_name(nxtsymb) != 'do':

return '"do" expected'

scan()

if not check(operators()):

return 'operators expected'

if get\_token\_name(nxtsymb) != 'end while':

return '"end while" expected'

return ''

elif get\_token\_name(nxtsymb) == 'dim':

first\_flag = True

while get\_token\_name(nxtsymb) == ',' or first\_flag:

first\_flag = False

scan()

if nxtsymb[0] != 'I':

return 'identifier expected'

scan()

if get\_token\_name(nxtsymb) not in ['(', '[']:

return '"(" expected'

scan()

if nxtsymb[0] not in ['C', 'I']:

return 'length of array expected'

scan()

if get\_token\_name(nxtsymb) not in [')', ']']:

return '")" expected'

scan()

antiscan()

elif get\_token\_name(nxtsymb) == 'sub':

scan()

if nxtsymb[0] != 'I':

return 'function name expected'

scan()

if get\_token\_name(nxtsymb) == '(':

first\_flag = True

while get\_token\_name(nxtsymb) == ',' or first\_flag:

first\_flag = False

scan()

if nxtsymb[0] != 'I':

return 'identifier expected'

scan()

if get\_token\_name(nxtsymb) != ')':

return '")" expected'

scan()

if not check(operators()):

return 'operators expected'

print('NU SHO', get\_token\_name(nxtsymb))

if get\_token\_name(nxtsymb) != 'end sub':

return '"end sub" expected'

return ''

elif get\_token\_name(nxtsymb) == 'return':

scan()

if not check(expression()):

return 'invalid expression'

return ''

else:

return 'operator expected'

return ''

def operators():

scan()

mistake = False

while not mistake:

# print('PERED:', get\_token\_name(nxtsymb), nxtsymb)

mistake = operator()

# print('OSHIBKA', mistake)

if mistake:

return ''

if cursymb[0] == 'I' and get\_token\_name(nxtsymb) == ':':

scan()

else:

scan()

# print('POSLE:', get\_token\_name(nxtsymb), nxtsymb)

if get\_token\_name(nxtsymb) == ';':

scan()

if get\_token\_name(nxtsymb) == 'stop':

return ''

def condition():

# print(Fore.GREEN + 'condition' + Style.RESET\_ALL)

if not check(expression()):

return 'expression expected'

scan()

if get\_token\_name(nxtsymb) not in ['=', '>', '<', '>=', '<=', '<>']:

return 'invalid comparasion operation'

scan()

if not check(expression()):

return 'expression expected'

return ''

def assignment():

# print(Fore.GREEN + 'assignment' + Style.RESET\_ALL)

if get\_token\_name(nxtsymb) != '=':

return '"=" expected'

scan()

if not check(expression()):

return 'invalid expression'

return ''

def expression():

# print(Fore.GREEN + 'expression' + Style.RESET\_ALL)

if not check(term()):

return 'term expected'

scan()

while get\_token\_name(nxtsymb) in ['+', '-']:

scan()

if not check(term()):

return 'term expected'

scan()

else:

antiscan()

pass

return ''

def term():

# print(Fore.GREEN + 'term' + Style.RESET\_ALL)

if not check(factor()):

return 'factor expected'

scan()

while get\_token\_name(nxtsymb) in ['\*', '/', '^']:

scan()

if not check(factor()):

return 'factor expected'

scan()

else:

antiscan()

pass

return ''

def factor():

# print(Fore.GREEN + 'factor' + Style.RESET\_ALL)

if get\_token\_name(nxtsymb) == '(':

scan()

if not check(expression()):

return 'expression expected'

scan()

if get\_token\_name(nxtsymb) != ')':

return '")" expected'

else:

if not check(argument()):

return 'constant or identifier expected'

return ''

def argument():

# print(Fore.GREEN + 'argument' + Style.RESET\_ALL)

if nxtsymb[0] not in ['I', 'C']:

return 'constant or identifier expected'

scan()

if get\_token\_name(nxtsymb) in ['[', '(']:

scan()

if not(check(expression())):

return 'invalid expression'

scan()

while get\_token\_name(nxtsymb) == ',':

scan()

if not(check(expression())):

return 'invalid expression'

scan()

if get\_token\_name(nxtsymb) not in [']', ')']:

return 'expected \']\' or \')\''

else:

antiscan()

return ''

subprocess.run(["python", "lab1.py"])

with open('./lab1.json') as lab1\_file:

data = json.load(lab1\_file)

tokens\_chain = data['chain']

service\_words = data['tables']['service\_words']

operations = data['tables']['operations']

separators = data['tables']['separators']

constants = data['tables']['constants']

identifiers = data['tables']['identifiers']

new\_tokens\_chain = []

for item in tokens\_chain:

if get\_token\_name(item) != ' ':

new\_tokens\_chain.append(item)

tokens\_chain = list(new\_tokens\_chain)

nxtsymb = []

cursymb = []

token\_index = -1

mistakes = []

def is\_mistakes():

check(wrapper())

if mistakes:

return mistakes[0]

return False

with open('./checked.txt', 'w') as outfile:

outfile.write(is\_mistakes())

Примеры





