

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 №1**

**«**Построение лексического анализатора**»**

по курсу «Методы разработки трансляторов»

Отчет подготовила

студентка 36 группы:

А. С. Радченко

Отчет принял

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

Краснодар, 2019

Цель лабораторной работы: построить лексический анализатор для языка программирования Basic.

Построим таблицы соответствия для служебных слов, операций и разделителей.

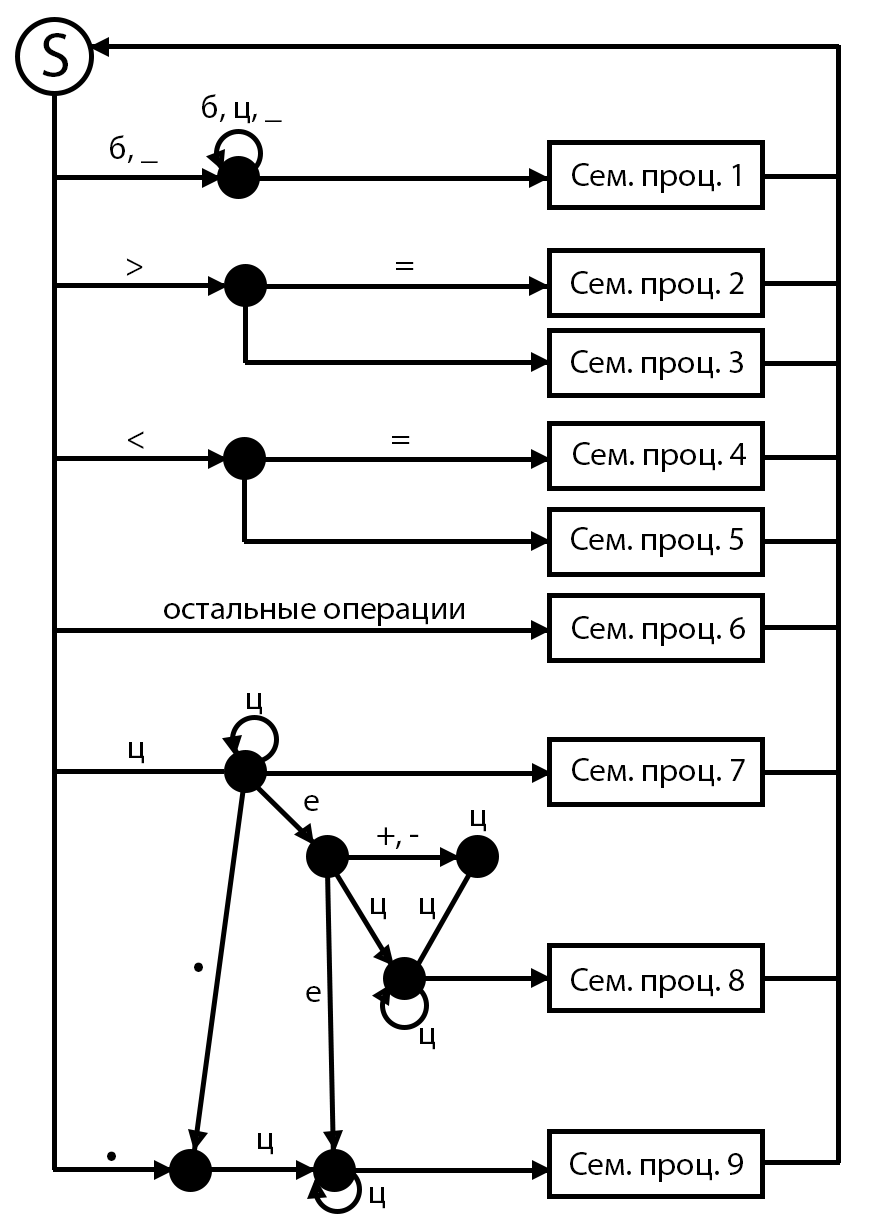
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| W0 | W1 | W2 | W3 | W4 | W5 | W6 | W7 | W8 | W9 |
| = | STOP | END | DIM | GOTO | GOSUB | IF | THEN | ELSE | END IF |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
| ( | ) | [ | ] | “ | : | , | ␣ |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| O0 | O1 | O2 | O3 | O4 | O5 | O6 | O7 | O8 | O9 | O10 |
| + | - | \* | / | ^ | < | > | = | <> | <= | >= |

Сначала алгоритм убирает номера строк, которые обязательны в синтаксисе языка Basic. После этого алгоритм очищает код от комментариев, лишних переносов строк, пробелов и табуляций.

Затем алгоритм пытается найти в начале кода токен из таблиц выше. Если попытка заканчивается провалом, то дальше строка обрабатывается автоматом. Его схема изображена на следующей странице. После того, как алгоритм вычленяет из кода первый встретившийся токен, токен убирается из кода программы. Вышеописанные действия повторяются для оставшейся части кода, пока такая часть есть.



Листинг программы:

import re

import pprint

import json

service\_words = ['stop', 'end', 'dim', 'goto', 'gosub', 'if', 'then', 'else', 'end if']

operations = ['+', '-', '\*', '/', '^', '<', '>', '=', '<>', '<=', '>=']

separators = [' ', ',', ':', '(', ')', '[', ']', '\'', '"']

# In[205]:

def filter\_program(text):

formatted\_text = []

text = re.sub(r'[$%&!#]', '', text)

lines = text.split('\n')

for index, line in enumerate(lines):

splitted\_text = line.split('\'')

even\_flag = False

for span in splitted\_text:

if not even\_flag:

formatted\_span = re.sub(r'[\t\n]+', ' ', span)

formatted\_span = re.sub(r' +', ' ', formatted\_span)

formatted\_span = re.sub(r'^[0-9]+ ', '', formatted\_span)

formatted\_span = re.sub(r'REM.\*', '', formatted\_span)

formatted\_text.append({

'text': formatted\_span.lower(),

'type': 'code',

'line': index + 1

})

else:

formatted\_text.append({

'text': span,

'type': 'string',

'line': index + 1

})

even\_flag = not even\_flag

if not even\_flag:

print(Fore.RED + 'Unexpected end of line. There is unclosed apostrophe!' + Style.RESET\_ALL)

return None

return formatted\_text

class Analyzer:

state = 'S'

string = ''

collecting\_string = ''

def \_\_init\_\_(self, string):

self.string = string

def reset(self):

self.collecting\_string = ''

self.state = 'S'

def unexpected(self, symbol):

self.state = 'error'

print(Back.RED, Fore.WHITE,'ERRORE!', Style.RESET\_ALL, 'Unexpected symbol \"' + symbol + '\"')

return {

'kind': 'error',

'token': symbol,

'residue': ''

}

def symbol\_return(self, symbol):

self.string = symbol + self.string

if symbol != '':

self.collecting\_string = self.collecting\_string[:-1]

def collect\_next(self):

# print('String: "', end='')

# print(Fore.BLUE + self.string + Style.RESET\_ALL, end='')

# print('"')

try:

symbol = self.string[0]

self.string = self.string[1:]

except:

symbol = ''

self.collecting\_string += symbol

if self.state == 'S':

if symbol.isalpha() or symbol == '\_':

self.state = 'letter\_at\_first'

elif symbol == '<':

self.state = '<\_at\_first'

elif symbol == '>':

self.state = '>\_at\_first'

elif symbol in operations:

return {

'kind': 'operation',

'token': symbol,

'residue': self.string

}

elif symbol.isdigit():

self.state = 'digit\_at\_first'

elif symbol == '.':

self.state = '.\_at\_first'

return self.collect\_next()

if self.state in ['number -> .. -> number']:

if symbol.isdigit():

return self.collect\_next()

elif symbol in operations + ['']:

self.symbol\_return(symbol)

return {

'kind': 'integer\_interval',

'token': self.collecting\_string,

'residue': self.string

}

else:

return self.unexpected(symbol)

if self.state in ['number -> ..']:

if symbol.isdigit():

self.state = 'number -> .. -> number'

return self.collect\_next()

else:

return self.unexpected(symbol)

if self.state == 'digit\_at\_first':

if symbol.isdigit():

return self.collect\_next()

elif symbol == '.':

self.state = 'number -> .'

return self.collect\_next()

elif symbol == 'e':

self.state = 'number -> e'

return self.collect\_next()

elif symbol in operations + ['']:

self.symbol\_return(symbol)

return {

'kind': 'integer',

'token': self.collecting\_string,

'residue': self.string

}

else:

return self.unexpected(symbol)

if self.state == 'number -> .':

if symbol == '.':

self.symbol\_return(symbol)

self.symbol\_return(symbol)

return {

'kind': 'integer',

'token': self.collecting\_string,

'residue': self.string

}

if self.state == '.\_at\_first':

if symbol == '.':

return {

'kind': 'separator',

'token': self.collecting\_string,

'residue': self.string

}

if self.state in ['.\_at\_first', 'number -> .']:

if symbol.isdigit():

return self.collect\_next()

elif symbol == 'e':

self.state = 'number -> e'

return self.collect\_next()

elif symbol in operations + ['']:

self.symbol\_return(symbol)

return {

'kind': 'real',

'token': self.collecting\_string,

'residue': self.string

}

elif symbol == '.':

self.state = 'number -> ..'

return self.collect\_next()

else:

return self.unexpected(symbol)

if self.state == 'number -> e':

if symbol in ['+', '-']:

self.state = 'number -> e -> +/-'

return self.collect\_next()

elif symbol.isdigit():

self.state = 'number -> e -> digit'

return self.collect\_next()

else:

return self.unexpected(symbol)

if self.state in ['number -> e -> +/-', 'number -> e -> digit']:

if symbol.isdigit():

return self.collect\_next()

elif symbol in operations + ['']:

self.symbol\_return(symbol)

return {

'kind': 'real',

'token': self.collecting\_string,

'residue': self.string

}

else:

return self.unexpected(symbol)

if self.state == '<\_at\_first':

if symbol in ['>', '=']:

return {

'kind': 'operation',

'token': self.collecting\_string,

'residue': self.string

}

else:

self.symbol\_return(symbol)

return {

'kind': 'operation',

'token': self.collecting\_string,

'residue': self.string

}

if self.state == '>\_at\_first':

if symbol == '=':

return {

'kind': 'operation',

'token': self.collecting\_string,

'residue': self.string

}

else:

self.symbol\_return(symbol)

return {

'kind': 'operation',

'token': self.collecting\_string,

'residue': self.string

}

if self.state == 'letter\_at\_first':

if symbol.isalpha() or symbol.isdigit() or symbol == '\_':

return self.collect\_next()

elif symbol in operations + ['']:

self.symbol\_return(symbol)

return {

'kind': 'identifier',

'token': self.collecting\_string,

'residue': self.string

}

else:

return self.unexpected(symbol)

return {

'kind': 'exeption',

'token': str(self.state),

'residue': self.string

}

def find\_in\_begin\_of(line):

global separators, operations, service\_words

for i in range(len(line), -1, -1):

if line[:i] in separators + operations + service\_words + constants + identifiers:

return [line[:i], line[i:]]

return False

def split\_by\_token(line):

found = find\_in\_begin\_of(line)

if found:

return found

# Not found a tabled token in begin of line... OKay! Search the right border of this identifier or constant!

for right\_border in range(len(line)+1):

if find\_in\_begin\_of(line[right\_border:]):

break

return [line[:right\_border], line[right\_border:]]

def get\_next\_token():

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

if not segments:

return False

while segments and not segments[0]['text'] and segments[0]['type'] == 'code':

segments.pop(0)

if not segments:

return False

line = segments[0]['line']

if segments[0]['type'] == 'code':

[token, segments[0]['text']] = split\_by\_token(segments[0]['text'])

if not segments[0]['text']:

segments.pop(0)

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

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

found = False

for index, table in enumerate(tables):

if token in table:

return [symbols[index], table.index(token), line]

found = True

break

if not found:

automat = Analyzer(token)

tail = token

while tail:

automat.reset()

analyzed = automat.collect\_next()

kind = analyzed['kind']

token\_name = analyzed['token']

tail = analyzed['residue']

if kind == 'identifier':

if {'type': 'identifier', 'value': token} not in identifiers:

identifiers.append({

'type': 'identifier',

'value': token

})

else:

return ['I', identifiers.index({'type': 'identifier', 'value': token})]

return ['I', len(identifiers) - 1, line]

if kind in ['string', 'integer', 'real']:

constants.append({

'type': kind,

'value': token

})

return ['C', len(constants) - 1, line]

else:

constants.append({

'type': 'string',

'value': segments[0]['text']

})

segments.pop(0)

return ['C', len(constants) - 1, line]

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]]

segments = filter\_program(open('./src/simple.bas').read())

constants = []

identifiers = []

chain = []

token = 'nu sho, poihali'

while token:

token = get\_next\_token()

if token:

chain.append(token)

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

data = {

'chain': chain,

'tables': {

'service\_words': service\_words,

'operations': operations,

'separators': separators,

'constants': constants,

'identifiers': identifiers

}

}

with open('./res/lab1.json', 'w') as outfile:

json.dump(data, outfile)

print(identifiers)

Примеры

