

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 №2**

**«**Перевод исходной программы в обратную польскую запись**»**

по курсу «Методы разработки трансляторов»

Отчет подготовили

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

Отчет принял

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

Краснодар, 2019

В ходе лабораторной работы требуется реализовать алгоритм Дейкстра для перевода программы на языке Basic в обратную польскую запись.

Ниже представлена таблица приоритетов операторов и операций.

|  |  |
| --- | --- |
| Приоритет | Операторы и операции |
| 1 | (, [, АЭМ, Ф, IF, WHILE |
| 2 | ), ], “,”, ;, THEN, ELSE |
| 3 | =, GOTO |
| 4 | || |
| 5 | && |
| 6 | ! |
| 7 | <, >, <=, >=, ==, != |
| 8 | +, - |
| 9 | \*, / |
| 10 | ^ |
| 11 | RETURN |
| 12 | : |
| 13 | НФ, КФ |

Если рассматриваемый символ является операцией, то её приоритет сравнивается с приоритетом верхнего символа стека. Если стек пуст, операция заносится в стек. Если в стеке верхний элемент имеет более высокий приоритет, то рассматриваемая операция проталкивается в стек. Если в стеке верхний элемент имеет более низкий или равный приоритет, то из стека в выходную строку выталкиваются все элементы (операции) до тех пор, пока не встретится операция с приоритетом выше, чем у рассматриваемой операции, после чего операция из входной строки проталкивается в стек. Если входной символ "(", то он всегда проталкивается в стек. Если входной символ ")", то он выталкивает из стека все символы операций до ближайшей "(". Сами скобки взаимно уничтожаются и в выходную строку не попадают. Для работы с массивами используется оператор АЭМ со счётчиком числа индексов массива. Оператор АЭМ так же имеет свой приоритет.

Для реализации перевода условных выражений используется следующий алгоритм. Условное выражение переводится как обычное. После него указывается имя метки и оператор УПЛ. После этого в ОПЗ переводятся операторы из блока THEN. Если есть блок операторов ELSE, то в конце блока THEN ставится безусловный переход на метку, с номером на один больше, чем у метки, которую указали перед УПЛ, а в начале ставим метку, которую указали перед УПЛ. После условного оператора ставим эту самую метку.

Для реализации циклов используется схожий подход, что и для условных выражений. Единственным отличием становится то, что в начало блока цикла ставится служебное обозначение «НЦ». Перед условием идёт первая метка, после условия условный переход по метке, идущей сразу после блока цикла. В конце блока цикла идёт безусловный переход по первой метке, стоящей перед условием.

Ниже приведён листинг программы:

#!/usr/bin/env python

# coding: utf-8

import re

import sys

import json

import subprocess

def get\_priority(operation):

priorities = [

['(', '[', 'АЭМ', 'Ф', 'IF', 'WHILE'],

[')', ']', ',', ';', 'THEN', 'ELSE'],

['=', 'goto'],

['||'],

['&&'],

['!'],

['<', '>', '<=', '>=', '==', '!='],

['+', '-'],

['\*', '/'],

['^'],

['RETURN'],

[':'],

['НФ', 'КФ', 'НП', 'КП']

]

for index, row in enumerate(priorities):

if operation in row:

return index

return False

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 get\_token\_type(token):

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

token\_name = get\_token\_name(token)

if token\_name in service\_words:

return 'service\_word'

if token\_name in operations:

return 'operation'

if token\_name in separators:

return 'separator'

if token[0] == 'C':

return constants[token[1]]['type']

if token[0] == 'I':

return identifiers[token[1]]['type']

return False

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

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

data = json.load(lab1\_file)

chain = data['chain']

tables = data['tables']

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

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

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

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

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

stack = []

result = []

AEM\_counter = None

F\_counter = None

M\_counter = 0

identifier\_context = False

if\_context = None

while\_context = None

while\_m = 0

type\_context = None

var\_pool = []

var\_type = None

func\_context = False

arg\_counter = 0

dim\_counter = 0

token\_number = 0

for token in chain:

token\_number += 1

token\_name = get\_token\_name(token)

token\_type = get\_token\_type(token)

previous\_stack = list(stack)

if token\_type != 'separator' or token\_name in ['(', ')', '[', ']', ',', ':', ';']:

if token\_type == 'identifier':

identifier\_context = True

if token\_name == 'return':

stack.append('RETURN')

continue

# Var and func declaring

if token\_name in ['void', 'int', 'float', 'char']:

type\_context = 'type'

var\_type = token\_name

var\_pool = []

if func\_context:

arg\_counter = 1 if arg\_counter == 0 else arg\_counter

if type\_context in ['type', 'type ids'] and token\_type == 'identifier':

type\_context = 'type ids'

var\_pool.append(token\_name)

# also look at the ';' processing below

if token\_name == '}' and not if\_context and not while\_context:

result.append('КФ')

func\_context = False

arg\_counter = 0

# WHILE processing

if token\_name == 'while':

stack.append('WHILE')

while\_context = 'while'

M\_counter += 1

while\_m = M\_counter

result.append('НЦ M' + str(while\_m) + ':')

if token\_name == 'do' and while\_context == 'while':

while stack and stack[-1] != 'WHILE':

result.append(stack.pop())

while\_context = 'while do'

M\_counter += 1

result.append('M' + str(M\_counter) + ' УПЛ')

if token\_name == 'end while' and while\_context == 'while do':

while stack and stack[-1] != 'WHILE':

result.append(stack.pop())

if stack and stack[-1] == 'WHILE':

stack.pop()

result.append('КЦ M' + str(while\_m) + ' БП M' + str(M\_counter) + ':')

while\_context = None

# IF processing

if token\_name == 'if':

stack.append('IF')

if\_context = 'if'

if token\_name == 'then' and if\_context == 'if':

if\_context = 'if then'

while stack and stack[-1] != 'IF':

result.append(stack.pop())

M\_counter += 1

result.append('M' + str(M\_counter) + ' УПЛ')

if token\_name == 'else' and if\_context == 'if then':

if\_context = 'if then else'

while stack and stack[-1] != 'IF':

result.append(stack.pop())

M\_counter += 1

result.append('M' + str(M\_counter) + ' БП M' + str(M\_counter-1) + ':')

if token\_name == 'end if':

if if\_context == 'if then':

while stack and stack[-1] != 'IF':

result.append(stack.pop())

if stack and stack[-1] == 'IF':

stack.pop()

result.append('M' + str(M\_counter) + ':')

if\_context = None

if if\_context == 'if then else':

while stack and stack[-1] != 'IF':

result.append(stack.pop())

if stack and stack[-1] == 'IF':

stack.pop()

result.append('M' + str(M\_counter) + ':')

if\_context = None

# DIM processing

if token\_name == 'dim':

# result.append('НА')

dim\_counter = 1

# SUB processing

if token\_name == 'sub':

result.append('НП')

func\_context = 'sub'

if token\_name == 'end sub':

while stack and stack[-1] != 'КП':

result.append(stack.pop())

result.append('КП')

# Ordinary expression processing

if token\_type in ['identifier', 'integer', 'string', 'real']:

result.append(token\_name)

if token\_type in ['operation'] or token\_name in ['=', 'goto', ':']:

token\_name = 'БП' if token\_name == 'goto' else token\_name

if not stack:

stack.append(token\_name)

else:

while stack and get\_priority(stack[-1]) > get\_priority(token\_name):

result.append(stack.pop())

else:

stack.append(token\_name)

# Brackets processing

if token\_name == '(' and not identifier\_context:

stack.append('(')

if token\_name == ')' and not (F\_counter):

while stack[-1] != '(':

# print(stack)

result.append(stack.pop())

stack.pop()

if stack and stack[-1] == 'WHILE':

while\_context = 'while ()'

M\_counter += 1

result.append('M' + str(M\_counter) + ' УПЛ')

if type\_context == 'type ids':

result.append(str(len(var\_pool)) + ' ' + var\_type)

# Arrays processing

if token\_name == '[':

stack.append('АЭМ')

AEM\_counter = 2

if token\_name == ',' and AEM\_counter:

while stack[-1] != 'АЭМ':

result.append(stack.pop())

AEM\_counter += 1

if token\_name == ']':

while stack[-1] != 'АЭМ':

result.append(stack.pop())

result.append(str(AEM\_counter) + ' ' + stack.pop())

AEM\_counter = None

# Functions processing

if token\_name == '(' and (identifier\_context or func\_context):

stack.append('Ф')

F\_counter = 1

if type\_context:

func\_context = True

arg\_counter = 0

result.append('1 ' + str(var\_type))

type\_context = None

if token\_name == ',' and F\_counter:

while stack[-1] != 'Ф':

result.append(stack.pop())

F\_counter += 1

if token\_name == ',' and dim\_counter:

dim\_counter += 1

if token\_name == ')' and F\_counter:

F\_counter += 1

if type\_context == 'type ids':

result.append(str(len(var\_pool)) + ' ' + var\_type)

type\_context = None

while stack[-1] != 'Ф':

result.append(stack.pop())

result.append(str(F\_counter) + ' ' + stack.pop())

F\_counter = None

func\_context = 'sub args'

# ";" processing

if token\_name == ';' or token\_number == len(chain):

if dim\_counter:

result.append(str(dim\_counter) + ' НА')

dim\_counter = 0

if func\_context == 'sub':

result.append('1 Ф')

func\_context = None

while stack and stack[-1] not in ['IF', 'WHILE', 'КП'] and stack[-1] != 'Ф':

result.append(stack.pop())

if type\_context == 'type ids':

result.append(str(len(var\_pool)) + ' '+ var\_type)

type\_context = None

if token\_type != 'identifier':

identifier\_context = False

previous\_stack = list(stack)

while stack:

result.append(stack.pop())

string = ''

for i in result:

string += ' ' + i

data = {

'rpn': result,

'tables': tables

}

with open('./lab2.json', 'w') as outfile:

json.dump(data, outfile)

Примеры



