Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчёт**

**по лабораторной работе №2**

**Дисциплина: Методы разработки трансляторов**

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

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К. А. Корнилов

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Математическое и программное обеспечение

компьютерных технологий

Преподаватель

д-р техн. наук, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю. М. Вишняков

**СОДЕРЖАНИЕ**

[1 Вариант задания 3](#_Toc103025612)

[2 Понятие обратной польской записи 3](#_Toc103025613)

[3 Алгоритм Дейкстры 4](#_Toc103025614)

[4 Перевод операторов цикла в ОПЗ 7](#_Toc103025615)

[5 Результаты экспериментов 9](#_Toc103025616)

[Приложение А Листинг программы и комментарии к нему 13](#_Toc103025617)

**1 Вариант задания**

Вариант задания представляет собой пару: входной язык и выходной язык (таблица 1).

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| № варианта | Входной язык | Выходной язык |
| 47 | PHP | R |

Разработать программу для перевода закодированного текста исходной программы в обратную польскую запись.

Программа получает на входе файл – результат лексического анализа и строит обратную польскую запись исходной программы.

Отчет по работе должен содержать полное описание алгоритма Дейкстры: таблицу приоритетов операторов и операций, а также алгоритм работы со стеком. Листинг программы и комментарии к нему, пример.

**2 Понятие обратной польской записи**

Обратная польская запись (ОПЗ) – представляет собой одну из форм записи выражений и операторов, отличительной особенностью которой является расположение аргументов (операндов) перед операцией (оператором).

Например, выражение, записанное в обычной скобочной записи,

(a+d)/c+b\*(e+d),

в ОПЗ имеет следующее представление:

ad+c/bed+\*+.

Обратная польская запись получила широкое распространение благодаря своему основному преимуществу ОПЗ может быть вычислена за один просмотр цепочки слева направо, который часто называют проходом.

**3 Алгоритм Дейкстры**

Исследованию формальных способов преобразования арифметических и логических выражений в ОПЗ посвящены многочисленные исследования, однако в практике системного программирования наибольшее распространение получили способы преобразования на основе алгоритма Дейкстры.

Суть алгоритма Дейкстры можно представить следующим рисунком:

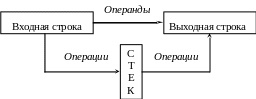


Рисунок 1 – Суть алгоритма Дейкстры

Из этого рисунка следует, что на вход алгоритма посимвольно поступает исходное выражение. Операнды исходного выражения пропускаются на выход и формируют так же посимвольно выходную строку. Операции обрабатываются по определенным правилам на основе стека.

Для реализации такой обработки известное в системном программировании понятие стека используется также в алгоритме Дейкстры для размещения в нем операций. При этом предварительно каждой операции приписывается свой приоритет на основе таблицы приоритетов, которая приведена ниже (таблица 2).

Таблица 2 – Таблица приоритетов

|  |  |
| --- | --- |
| Входной элемент | Приоритет |
| (  if  while  [  АЭМ  Ф  { | 0 |
| )  ,  ;  do  else  ] | 1 |
| = | 2 |
| Or  Xor | 3 |
| and | 4 |
| ! | 5 |

Продолжение таблицы 2

|  |  |
| --- | --- |
| Входной элемент | Приоритет |
| <  <=  !=  =  <>  >  >= | 6 |
| ^  &  |  ~  <<  >> | 7 |
| +  -  . | 8 |
| \*  /  % | 9 |
| \*\* | 10 |
| $ | 11 |
| }  function  return  echo | 12 |

**4 Перевод операторов цикла в ОПЗ**

Обработка оператора цикла с предусловием WHILE выражение DO оператор;:

1. Символ WHILE из входной строки заносится в стек. В стеке к символу WHILE добавляется рабочая метка Mi и после этого в выходную строку записывается часть Mi:.
2. Символ DO выталкивает в выходную строку все операции из стека до ближайшего WHILE Mi. В стеке к WHILE Mi добавляется рабочая метка Mi+1 и после этого в выходную строку записывается часть Mi+1 УПЛ.
3. Символ ‘;’ указывает на конец оператора цикла с предусловием и выталкивает из стека все символы до ближайшего WHILE Mi Mi+1, при этом сам WHILE уничтожается, а в выходную строку помещается Mi БП Mi+1:.

Обработка оператора цикла с постусловием while оператор do выражение; можно заменить последовательностью операторов

1. Символ ‘;’ указывает на конец оператора цикла с постусловием и выталкивают из стека все символы до ближайшего do Mi, при этом сам do уничтожается, а в выходную строку помещаются Mi+1 УПЛ Mi БП Mi+1:.

**5 Результаты экспериментов**

Примеры работы программы для программы для тестирования.

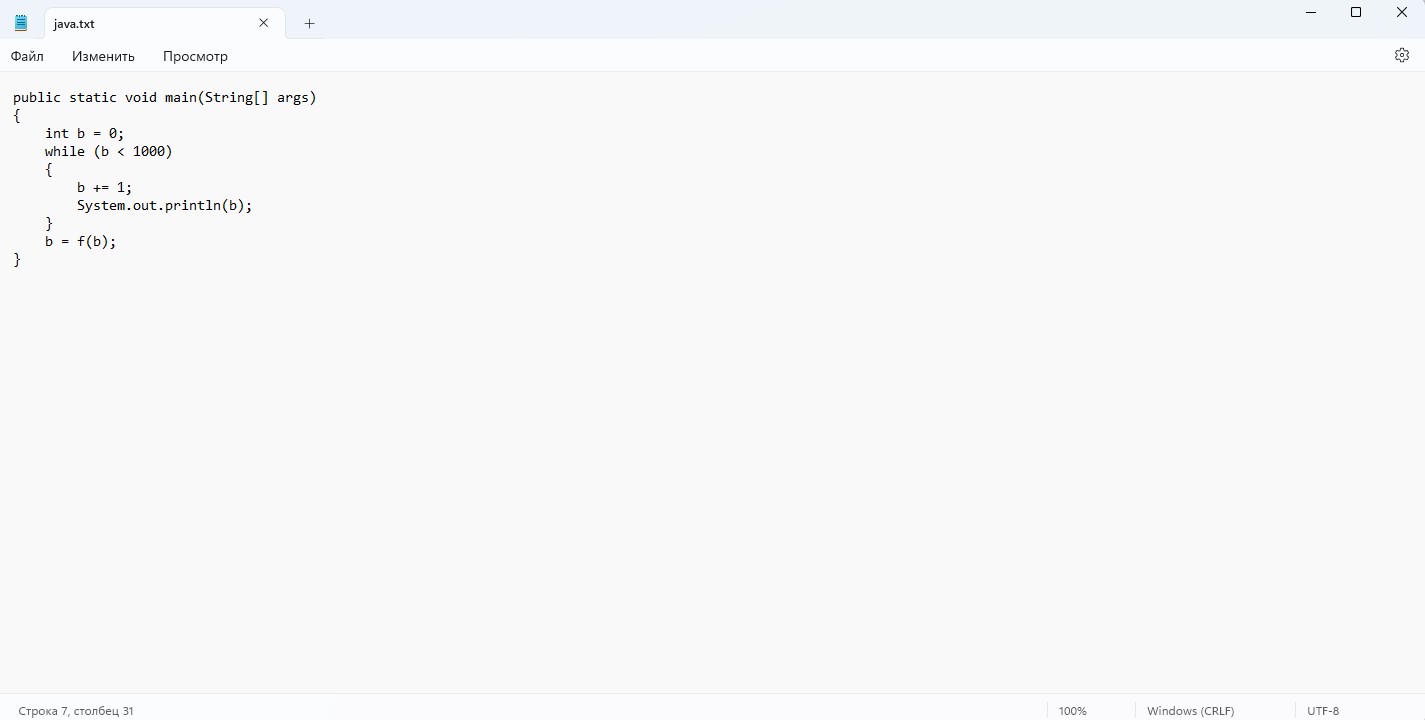


Рисунок 2 – Скриншот файла №1, содержащего текст на входном языке программирования

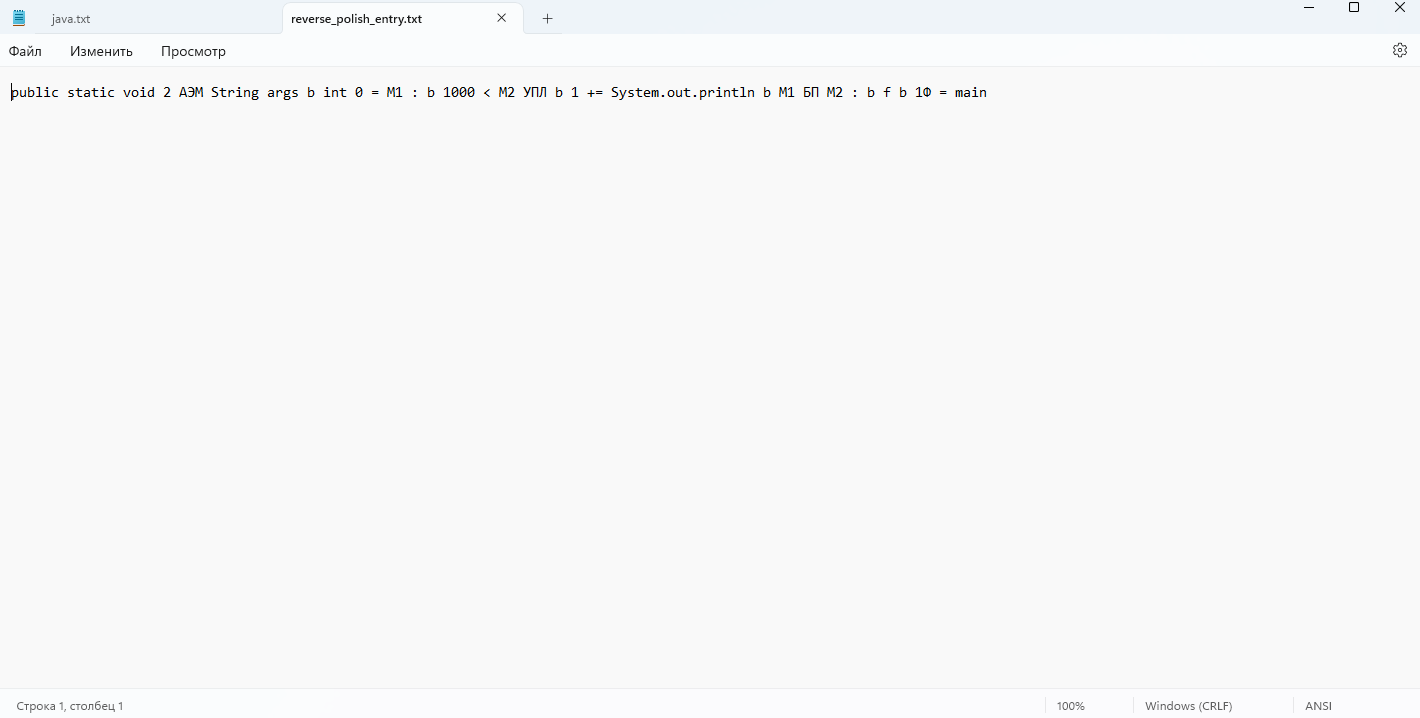


Рисунок 3 – Результат работы программы №1

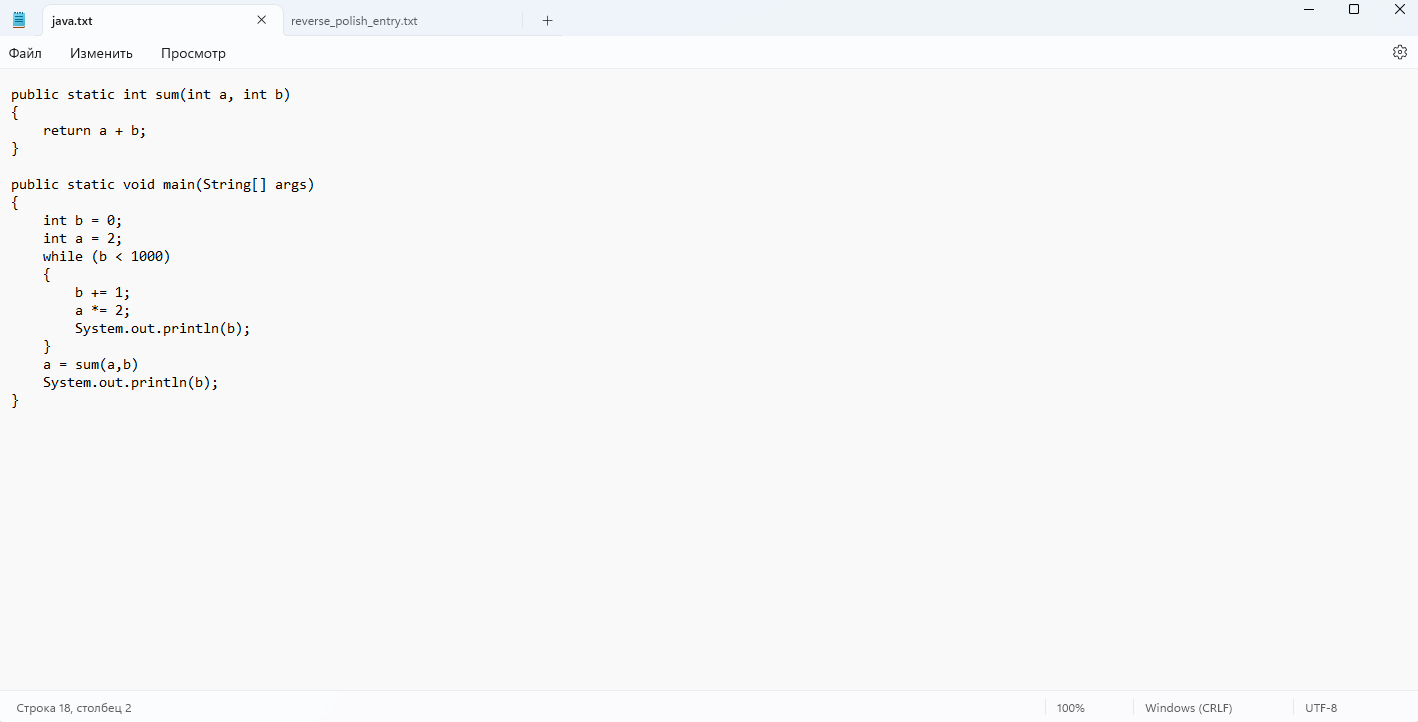


Рисунок 4 – Скриншот файла №2, содержащего текст на входном языке программирования

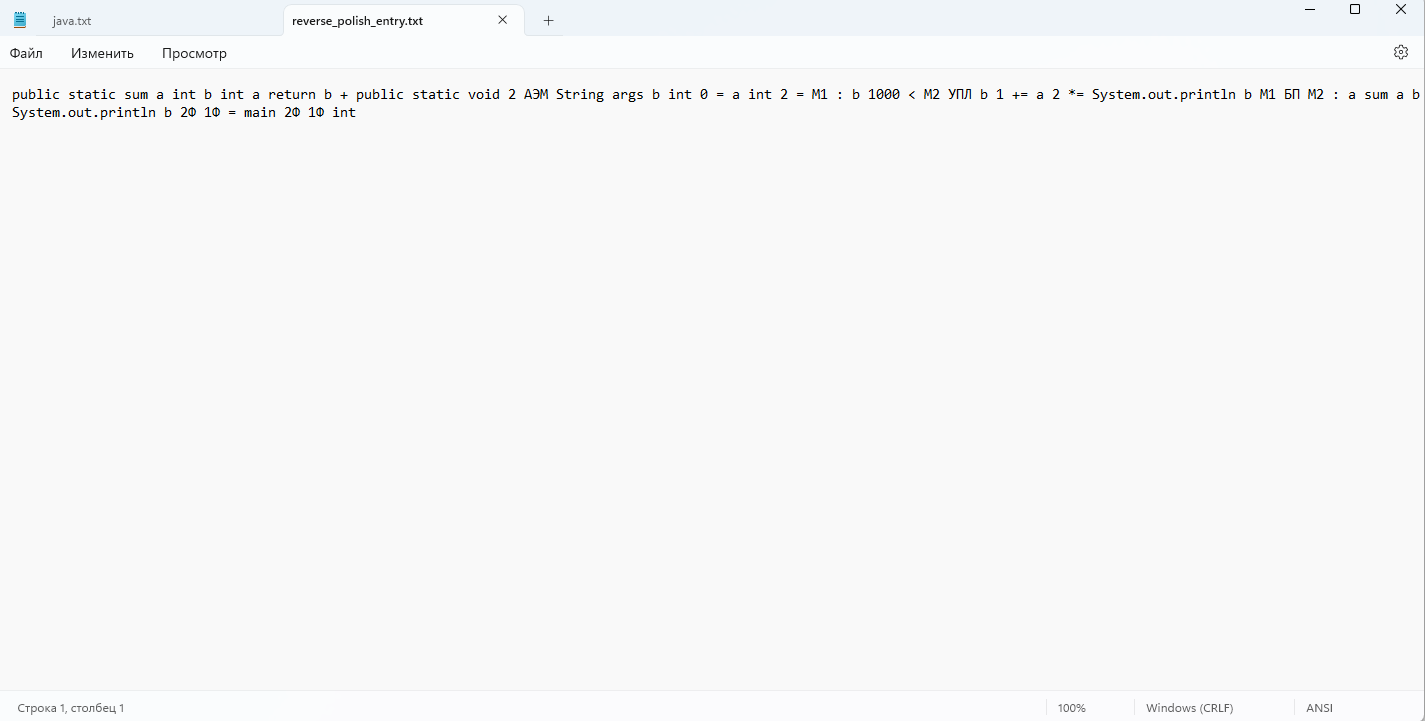


Рисунок 5 – Результат работы программы №2

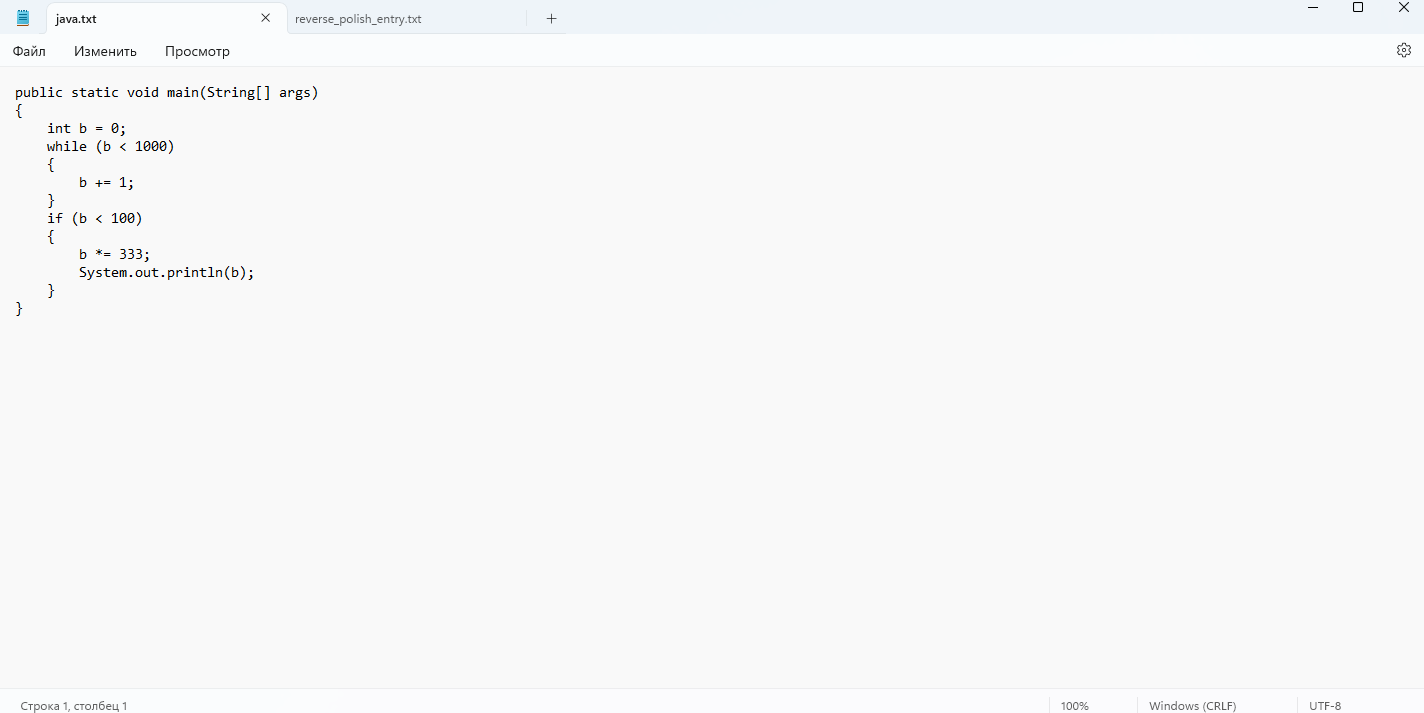


Рисунок 6 – Скриншот файла №3, содержащего текст на входном языке программирования

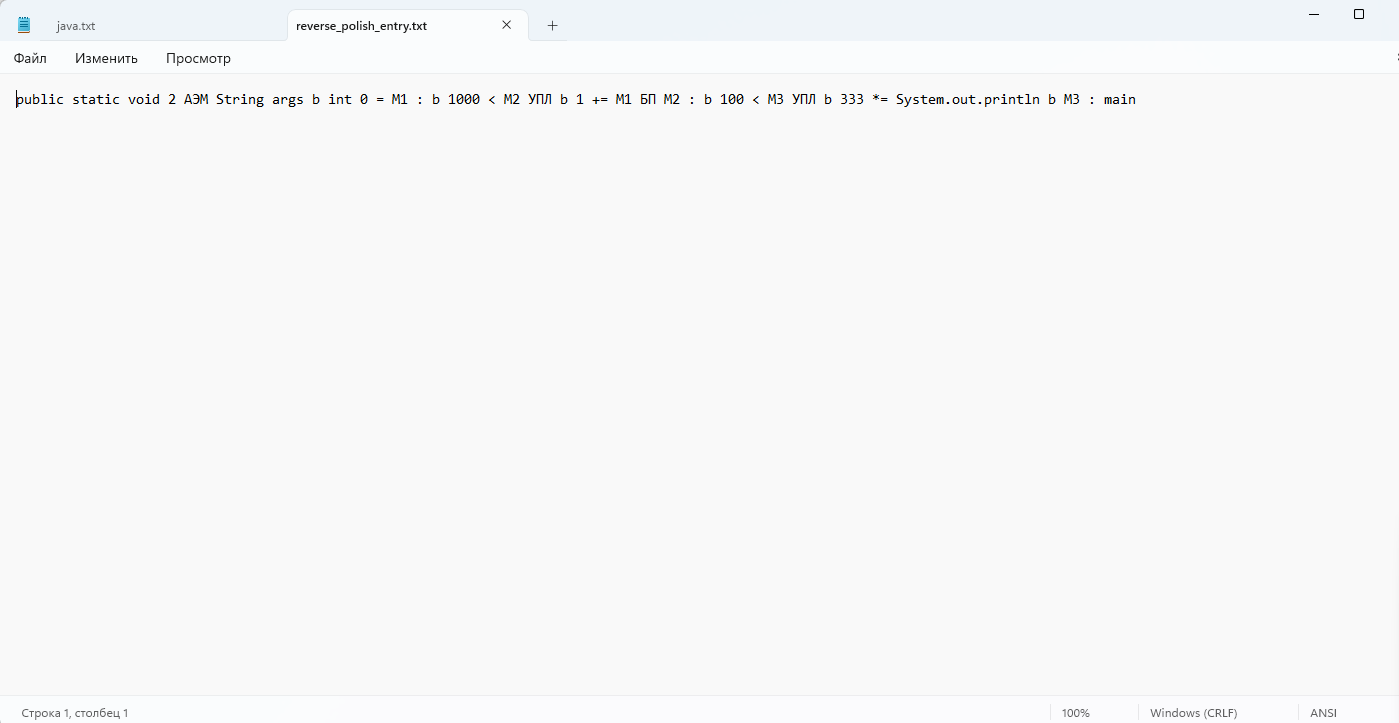


Рисунок 7 – Результат работы программы №3

**ПРИЛОЖЕНИЕ А   
Листинг программы и комментарии к нему**

import json

import re

from tkinter import \*

import tkinter.scrolledtext as st

import re

def prog():

#LAB1

SERVICE\_WORDS = ['abstract', 'case', 'continue', 'extends', 'goto', 'int', 'package', 'short',

'try', 'assert', 'catch', 'default', 'final', 'if', 'private',

'static', 'this', 'void', 'boolean', 'char', 'do','long', 'protected',

'throw', 'volatile', 'break', 'class', 'double', 'float', 'import', 'native',

'public', 'super','throws','while','byte','const','else','for','instanceof',

'new','return','switch','transient','print','println','main','System',

'out','String','args']

OPERATIONS = ['\*','+','-','%', '/','++','\*=','+=','-=','%=','/=','==', '<', '<=', '!=', '=', '>', '>=','&','|']

SEPARATORS = ['\t', '\n', ' ', '(', ')', ',', '.', ':', ';', '[', ']','{','}']

def check(tokens, token\_class, token\_value):

if not(token\_value in tokens[token\_class]):

token\_code = str(len(tokens[token\_class]) + 1)

tokens[token\_class][token\_value] = token\_class + token\_code

def get\_operation(input\_sequence, i):

for k in range(2, 0, -1):

if i + k < len(input\_sequence):

buffer = input\_sequence[i:i + k]

if buffer in OPERATIONS:

return buffer

return ''

def get\_separator(input\_sequence, i):

buffer = input\_sequence[i]

if buffer in SEPARATORS:

return buffer

return ''

# лексемы

tokens = {'W': {}, 'I': {}, 'O': {}, 'R': {}, 'N': {}, 'C': {}}

for service\_word in SERVICE\_WORDS:

check(tokens, 'W', service\_word)

for operation in OPERATIONS:

check(tokens, 'O', operation)

for separator in SEPARATORS:

check(tokens, 'R', separator)

# файл, содержащий текст на входном языке программирования

f = open('java.txt', 'r')

input\_sequence = f.read()

f.close()

i = 0

state = 'S'

output\_sequence = buffer = ''

while i < len(input\_sequence):

symbol = input\_sequence[i]

operation = get\_operation(input\_sequence, i)

separator = get\_separator(input\_sequence, i)

if state == 'S':

buffer = ''

if symbol.isalpha():

state = 'q1'

buffer += symbol

elif symbol.isdigit():

state = 'q3'

buffer += symbol

elif symbol == "'":

state = 'q9'

buffer += symbol

elif symbol == '"':

state = 'q10'

buffer += symbol

elif symbol == '/':

state = 'q11'

elif operation:

check(tokens, 'O', operation)

output\_sequence += tokens['O'][operation] + ' '

i += len(operation) - 1

elif separator:

if separator != ' ':

check(tokens, 'R', separator)

output\_sequence += tokens['R'][separator]

if separator == '\n':

output\_sequence += '\n'

else:

output\_sequence += ' '

elif i == len(input\_sequence) - 1:

state = 'Z'

elif state == 'q1':

if symbol.isalpha():

buffer += symbol

elif symbol.isdigit():

state = 'q2'

buffer += symbol

else:

if operation or separator:

if buffer in SERVICE\_WORDS:

output\_sequence += tokens['W'][buffer] + ' '

elif buffer in OPERATIONS:

output\_sequence += tokens['O'][buffer] + ' '

else:

check(tokens, 'I', buffer)

output\_sequence += tokens['I'][buffer] + ' '

if operation:

check(tokens, 'O', operation)

output\_sequence += tokens['O'][operation] + ' '

i += len(operation) - 1

if separator:

if separator != ' ':

check(tokens, 'R', separator)

output\_sequence += tokens['R'][separator]

if separator == '\n':

output\_sequence += '\n'

else:

output\_sequence += ' '

state = 'S'

elif state == 'q2':

if symbol.isalnum():

buffer += symbol

else:

if operation or separator:

check(tokens, 'I', buffer)

output\_sequence += tokens['I'][buffer] + ' '

if operation:

check(tokens, 'O', operation)

output\_sequence += tokens['O'][operation] + ' '

i += len(operation) - 1

if separator:

if separator != ' ':

check(tokens, 'R', separator)

output\_sequence += tokens['R'][separator]

if separator == '\n':

output\_sequence += '\n'

else:

output\_sequence += ' '

state = 'S'

elif state == 'q3':

if symbol.isdigit():

buffer += symbol

elif symbol == '.':

state = 'q4'

buffer += symbol

elif symbol == 'e' or symbol == 'E':

state = 'q6'

buffer += symbol

else:

if operation or separator:

check(tokens, 'N', buffer)

output\_sequence += tokens['N'][buffer] + ' '

if operation:

check(tokens, 'O', operation)

output\_sequence += tokens['O'][operation] + ' '

i += len(operation) - 1

if separator:

if separator != ' ':

check(tokens, 'R', separator)

output\_sequence += tokens['R'][separator]

if separator == '\n':

output\_sequence += '\n'

else:

output\_sequence += ' '

state = 'S'

elif state == 'q4':

if symbol.isdigit():

state = 'q5'

buffer += symbol

elif state == 'q5':

if symbol.isdigit():

buffer += symbol

elif symbol == 'e' or symbol == 'E':

state = 'q6'

buffer += symbol

else:

if operation or separator:

check(tokens, 'N', buffer)

output\_sequence += tokens['N'][buffer] + ' '

if operation:

check(tokens, 'O', operation)

output\_sequence += tokens['O'][operation] + ' '

i += len(operation) - 1

if separator:

if separator != ' ':

check(tokens, 'R', separator)

output\_sequence += tokens['R'][separator]

if separator == '\n':

output\_sequence += '\n'

else:

output\_sequence += ' '

state = 'S'

elif state == 'q6':

if symbol == '-' or symbol == '+':

state = 'q7'

buffer += symbol

elif symbol.isdigit():

state = 'q8'

buffer += symbol

elif state == 'q7':

if symbol.isdigit():

state = 'q8'

buffer += symbol

elif state == 'q8':

if symbol.isdigit():

buffer += symbol

else:

if operation or separator:

check(tokens, 'N', buffer)

output\_sequence += tokens['N'][buffer] + ' '

if operation:

check(tokens, 'O', operation)

output\_sequence += tokens['O'][operation] + ' '

i += len(operation) - 1

if separator:

if separator != ' ':

check(tokens, 'R', separator)

output\_sequence += tokens['R'][separator]

if separator == '\n':

output\_sequence += '\n'

else:

output\_sequence += ' '

state = 'S'

elif state == 'q9':

if symbol != "'":

buffer += symbol

elif symbol == "'":

buffer += symbol

check(tokens, 'C', buffer)

output\_sequence += tokens['C'][buffer] + ' '

state = 'S'

elif state == 'q10':

if symbol != '"':

buffer += symbol

elif symbol == '"':

buffer += symbol

check(tokens, 'C', buffer)

output\_sequence += tokens['C'][buffer] + ' '

state = 'S'

elif state == 'q11':

if symbol == '/':

state = 'q12'

elif symbol == '\*':

state = 'q13'

elif state == 'q12':

if symbol == '\n':

state = 'S'

elif i == len(input\_sequence) - 1:

state = 'Z'

elif state == 'q13':

if symbol == '\*':

state = 'q14'

elif state == 'q14':

if symbol == '/':

state = 'q15'

elif state == 'q15':

if symbol == '\n':

state = 'S'

elif i == len(input\_sequence) - 1:

state = 'Z'

i += 1

# файлы, содержащие все таблицы лексем

for token\_class in tokens.keys():

with open('%s.json' % token\_class, 'w') as write\_file:

data = {val: key for key, val in tokens[token\_class].items()}

json.dump(data, write\_file, indent=4, ensure\_ascii=False)

# файл, содержащий последовательность кодов лексем входной программы

f = open('tokens.txt', 'w')

f.write(output\_sequence)

f.close()

#LAB2

CLASSES\_OF\_TOKENS = ['W', 'I', 'O', 'R', 'N', 'C']

def is\_identifier(token):

return re.match(r'^I\d+$', inverse\_tokens[token])

def get\_priority(token):

if token in ['(', 'for', 'if', 'while', '[', 'АЭМ', 'Ф', '{']:

return 0

if token in [')', ',', ';', 'do', 'else', ']']:

return 1

if token == '=':

return 2

if token == '||':

return 3

if token == '&&':

return 4

if token == '!':

return 5

if token in ['<', '<=', '!=', '=', '>', '>=']:

return 6

if token in ['+', '-', '+=', '-=', '\*=', '/=']:

return 7

if token in ['\*', '/', '%']:

return 8

if token in ['}', 'public.static.void', 'procedure','int', 'double', 'boolean', 'String', 'float', 'args','return','System.out.println', 'main']:

return 9

return -1

# лексемы (код-значение)

tokens = {}

# файлы, содержащие все таблицы лексем

for token\_class in CLASSES\_OF\_TOKENS:

with open('%s.json' % token\_class, 'r') as read\_file:

data = json.load(read\_file)

tokens.update(data)

# лексемы (значение-код)

inverse\_tokens = {val: key for key, val in tokens.items()}

# файл, содержащий последовательность кодов лексем входной программы

f = open('tokens.txt', 'r')

inp\_seq = f.read()

f.close()

regexp = '[' + '|'.join(CLASSES\_OF\_TOKENS) + ']' + '\d+'

match = re.findall(regexp, inp\_seq)

t = [tokens[i] for i in match]

i = 0

stack = []

out\_seq = ''

aem\_count = proc\_num = proc\_level = operand\_count = 1

func\_count = tag\_count = proc\_num = if\_count = while\_count = \

begin\_count = end\_count = bracket\_count = 0

is\_if = is\_while = is\_description\_var = False

while i < len(t):

p = get\_priority(t[i])

if p == -1:

if t[i] != '\n' and t[i] != '\t':

out\_seq += t[i] + ' '

else:

if t[i] == '[':

aem\_count += 1

stack.append(str(aem\_count) + ' АЭМ')

elif t[i] == ']':

while not(re.match(r'^\d+ АЭМ$', stack[-1])):

out\_seq += stack.pop() + ' '

out\_seq += stack.pop() + ' '

aem\_count = 1

elif t[i] == '(':

if is\_identifier(t[i - 1]):

if t[i + 1] != ')':

func\_count += 1

stack.append(str(func\_count) + ' Ф')

else:

stack.append(t[i])

bracket\_count += 1

elif t[i] == ')':

while stack[-1] != '(' and not(re.match(r'^\d+ Ф$', stack[-1])):

out\_seq += stack.pop() + ' '

if re.match(r'^\d+ Ф$', stack[-1]):

stack.append(str(func\_count + 1) + ' Ф')

func\_count = 0

stack.pop()

bracket\_count -= 1

if bracket\_count == 0:

if is\_if:

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

out\_seq += stack.pop() + ' '

tag\_count += 1

stack[-1] += ' М' + str(tag\_count)

out\_seq += 'М' + str(tag\_count) + ' УПЛ '

is\_if = False

if is\_while:

while not(re.match(r'^while М\d+$', stack[-1])):

out\_seq += stack.pop() + ' '

tag\_count += 1

out\_seq += 'М' + str(tag\_count) + ' УПЛ '

stack[-1] += ' М' + str(tag\_count)

is\_while = False

elif t[i] == ',':

while not(re.match(r'^\d+ АЭМ$', stack[-1])) and \

not(re.match(r'^\d+ Ф$', stack[-1])) and \

not(re.match(r'^var', stack[-1])):

out\_seq += stack.pop() + ' '

if re.match(r'^\d+ АЭМ$', stack[-1]):

aem\_count += 1

stack.append(str(aem\_count) + ' АЭМ')

if re.match(r'^\d+ Ф$', stack[-1]):

func\_count += 1

stack.append(str(func\_count) + ' Ф')

elif t[i] == 'if':

stack.append(t[i])

if\_count += 1

bracket\_count = 0

is\_if = True

elif t[i] == 'else':

while not(re.match(r'^if М\d+$', stack[-1])):

out\_seq += stack.pop() + ' '

stack.pop()

tag\_count += 1

stack.append('if М' + str(tag\_count))

out\_seq += 'М' + str(tag\_count) + ' БП М' + str(tag\_count - 1) + ' : '

elif t[i] == 'while':

tag\_count += 1

stack.append(t[i] + ' М' + str(tag\_count))

out\_seq += 'М' + str(tag\_count) + ' : '

while\_count += 1

bracket\_count = 0

is\_while = True

elif t[i] == 'for':

j = i + 2

bracket\_count = 1

a = []

while t[j] != ';':

a.append(t[j])

j += 1

if t[j] == '(':

bracket\_count += 1

elif t[j] == ')':

bracket\_count -= 1

j += 1

b = []

while t[j] != ';':

b.append(t[j])

j += 1

if t[j] == '(':

bracket\_count += 1

elif t[j] == ')':

bracket\_count -= 1

j += 1

c = []

while bracket\_count != 0:

c.append(t[j])

j += 1

if t[j] == '(':

bracket\_count += 1

elif t[j] == ')':

bracket\_count -= 1

j += 1

d = []

while t[j] != ';' and t[j] != '{':

d.append(t[j])

j += 1

if t[j] == '{':

j += 1

bracket\_count = 1

d = ['{']

while bracket\_count != 0:

d.append(t[j])

j += 1

if t[j] == '{':

bracket\_count += 1

elif t[j] == '}':

bracket\_count -= 1

d.append('}')

j += 1

t = t[:i] + a + [';', '\n', 'while', '('] + b + [')', '{', '\n'] + d + \

['\n'] + c + [';', '\n', '}'] + t[j:]

i -= 1

elif t[i] == 'sub':

proc\_num += 1

stack.append('PROC ' + str(proc\_num) + ' ' + str(proc\_level))

elif t[i] == '{':

if len(stack) > 0 and re.match(r'^PROC', stack[-1]):

num = re.findall(r'\d+', stack[-1])

stack.pop()

out\_seq += '0 Ф ' + str(num[0]) + ' ' + str(num[1]) + ' НП '

stack.append('PROC ' + str(proc\_num) + ' ' + str(proc\_level))

begin\_count += 1

proc\_level = begin\_count - end\_count + 1

stack.append(t[i])

elif t[i] == '}':

end\_count += 1

proc\_level = begin\_count - end\_count + 1

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

out\_seq += stack.pop() + ' '

stack.pop()

if len(stack) > 0 and re.match(r'^PROC', stack[-1]):

stack.pop()

out\_seq += 'КП '

if if\_count > 0 and re.match(r'^if М\d+$', stack[-1]):

tag = re.search('М\d+', stack[-1]).group(0)

j = i + 1

while j < len(t) and t[j] == '\n':

j += 1

if j >= len(t) or t[j] != 'else':

stack.pop()

out\_seq += tag + ' : '

if\_count -= 1

if while\_count > 0 and re.match(r'^while М\d+ М\d+$', stack[-1]):

tag = re.findall('М\d+', stack[-1])

stack.pop()

out\_seq += tag[0] + ' БП ' + tag[1] + ' : '

while\_count -= 1

elif t[i] == ';':

if len(stack) > 0 and re.match(r'^PROC', stack[-1]):

num = re.findall(r'\d+', stack[-1])

stack.pop()

out\_seq += str(num[0]) + ' ' + str(num[1]) + ' НП '

elif len(stack) > 0 and stack[-1] == 'end':

stack.pop()

out\_seq += 'КП '

elif is\_description\_var:

proc\_num, proc\_level = re.findall('\d+', stack[-1])

stack.pop()

out\_seq += str(operand\_count) + ' ' + proc\_num + ' ' + proc\_level + \

' КО '

is\_description\_var = False

elif if\_count > 0 or while\_count > 0:

while not(len(stack) > 0 and stack[-1] == '{') and \

not(if\_count > 0 and re.match(r'^if М\d+$', stack[-1])) and \

not(while\_count > 0 and re.match(r'^while М\d+ М\d+$', stack[-1])):

out\_seq += stack.pop() + ' '

if if\_count > 0 and re.match(r'^if М\d+$', stack[-1]):

tag = re.search('М\d+', stack[-1]).group(0)

j = i + 1

while t[j] == '\n':

j += 1

if t[j] != 'else':

stack.pop()

out\_seq += tag + ' : '

if\_count -= 1

if while\_count > 0 and re.match(r'^while М\d+ М\d+$', stack[-1]):

tag = re.findall('М\d+', stack[-1])

out\_seq += tag[0] + ' БП ' + tag[1] + ' : '

while\_count -= 1

else:

while len(stack) > 0 and stack[-1] != '{':

out\_seq += stack.pop() + ' '

else:

while len(stack) > 0 and get\_priority(stack[-1]) >= p:

out\_seq += stack.pop() + ' '

stack.append(t[i])

i += 1

while len(stack) > 0:

out\_seq += stack.pop() + ' '

out\_seq = out\_seq.replace("System . out . println", "System.out.println")

out\_seq = re.sub(r'(\d) Ф', r'\1Ф', out\_seq)

# файл, содержащий обратную польскую запись

f = open('reverse\_polish\_entry.txt', 'w')

f.write(out\_seq)

f.close()

def write\_txt(data):

with open('java.txt','w') as file:

file.write(data)

def clicked():

write\_txt(codetxt.get("1.0","end"))

opzstext.delete("1.0",END)

prog()

f1 = open('reverse\_polish\_entry.txt','r')

text = f1.read()

opzstext.insert("1.0",text)

f1.close()

window=Tk()

window.title("LR2")

window.geometry('1100x500')

codetxt=st.ScrolledText(window)

codetxt.place(x=40,y=0,width=410,height=290)

opzstext=st.ScrolledText(window)

opzstext.place(x=600,y=80,width=500,height=140)

btngo=Button(window,text="Выполнить \n преобразование",command=clicked,font=("Arial", 10))

btngo.place(x=470,y=90,width=110,height=50)

window.mainloop()