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 №3**

**Дисциплина: Методы разработки трансляторов**

**Тема: «Перевод ОПЗ исходного выражения в текст на выходном языке. Генерация машинного кода»**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. Д. Воробьев

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

Краснодар

2024

**СОДЕРЖАНИЕ**

[1 Вариант задания 3](#_Toc104196070)

[2 Базовые понятия 3](#_Toc104196071)

[3 Правила генерации машинного кода 4](#_Toc104196072)

[3 Результаты экспериментов 7](#_Toc104196073)

[Приложение А Описание правил записи элементов заданного выходного языка 8](#_Toc104196074)

[Приложение Б Листинг программы и комментарии к нему 14](#_Toc104196075)

**1 Вариант задания**

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

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| № варианта | Входной язык | Выходной язык |
| 2 | Python | Java |

Разработать программу для формирования по обратной польской записи текста на выходном языке.

Программа получает на входе файл, содержащий ОПЗ исходной программы, и строит текст программы на машинном язык в соответствии с заданием.

Отчет по работе должен содержать описание правил записи перечисленных выше элементов заданного выходного языка, алгоритм работы МП-автомата и описание семантических процедур, листинг программы и комментарии к нему, пример.

**2 Базовые понятия**

Перевод ОПЗ в текст на выходном (машинном) языке представляет собой следующий этап трансляции исходной программы в машинные коды. Для реализации этой процедуры также используется автомат с магазинной памятью (МП-автомат).

Эту процедуру можно схематично представить следующим образом:



Рисунок 1 – Схематичное представление процедуры

Чтение символов операций из ОПЗ инициирует семантические процедуры, которые генерируют соответствующие заготовки машинного кода. Так же обрабатываются собранные в стеке операнды данной. Например, в случае обработки переменных и констант извлекается соответствующая им информация из таблиц идентификаторов и констант, которая используется для образования правильных адресных частей соответствующих машинных команд.

**3 Правила генерации машинного кода**

Пусть в качестве машинного языка выступает язык программирования Бейсик. Его особенностью является обязательная нумерация строк и отсутствие символьных меток. Поэтому в операторах перехода в качестве меток используют номера строк, на которые нужно передать управление.

Для построения МП-автомата по переводу ОПЗ в машинные коды и его семантических процедур введем ряд внутренних переменных:

1. P – счетчик вспомогательных переменных;
2. STR – счетчик строк.

Кроме того, организуем таблицу меток, которая реализует отображение символьных меток исходного языка в номера строк машинного языка:

Таблица 2 – Таблица меток

|  |  |
| --- | --- |
| Метка | Номер строки |

Эта таблица потребуется в дальнейшем для замены символьных меток на номера строк, что также является особенностью Бейсика как выходного языка.

Рассмотрим работу МП-автомата.

1. Если элемент входной строки – идентификатор или константа, то он заносится в стек (в исходном виде, т.е. не условное обозначение, а имя из таблицы идентификаторов или константа из таблицы констант); вспомогательные переменные и константы переносятся без изменения.

2. Для каждой операции и оператора определяется арность, т.е. количество операндов, и соответствующая семантическая процедура.

3. После выполнения каждой семантической процедуры в выходную строку заносится символ <ВК>, счетчик строк STR наращивается на единицу и заносится в начало новой строки.

Семантические процедуры для операторов и операций приведены в таблице 3.

Таблица 3 – Семантические процедуры для операторов и операций

|  |  |
| --- | --- |
| Лексема | Действия |
| НП | Извлечь из стека два элемента, занести в выходную строку текст *"REM Начало процедуры арг2, арг1"* |
| КП | Занести в выходную строку текст  *"REM Конец процедуры"* |
| DFD  BFD  DFT  CHAR | Извлечь из стека *арг1* – число переменных k; извлечь из стека k аргументов; занести в выходную строку текст  *"REM Вещественные переменные арг1, арг2, …, аргk"* |
| КО | Извлечь из стека два аргумента |
| УПЛ | Извлечь из стека два аргумента, занести в выходную строку текст  *" IF NOT(арг2) THEN GOTO арг1"* |

Продолжение таблицы 3

|  |  |
| --- | --- |
| БП | Извлечь из стека один аргумент, занести в выходную строку текст  *"GOTO арг1"* |
| : | Извлечь из стека один аргумент, занести в таблицу меток *арг1* и значение счетчика STR |
| +  \*  >  < | Извлечь из стека два аргумента, нарастить счетчик вспомогательных переменных Р, занести в выходную строку текст  *"Rp = арг2 <операция> арг1",*  занести в стек Rp. |
| = | Извлечь из стека 2 аргумента и занести в выходную строку текст *"арг2 = арг1"* |

В стеке *арг1* – это элемент, находящийся в вершине стека. Увеличение номера аргумента показывает его удаление от вершины стека и обратно порядку занесения элементов в стек.

Для арифметических выражений в целях уменьшения количества операторов присваивания и временных переменных возможен вариант формирования строки "(арг2 <операция> арг1)" и занесение ее в стек как единого аргумента для последующих операций и операторов. Недостатком такого подхода является избыточность круглых скобок в выражениях.

**3 Результаты экспериментов**

Пример работы программы для программы для тестирования.

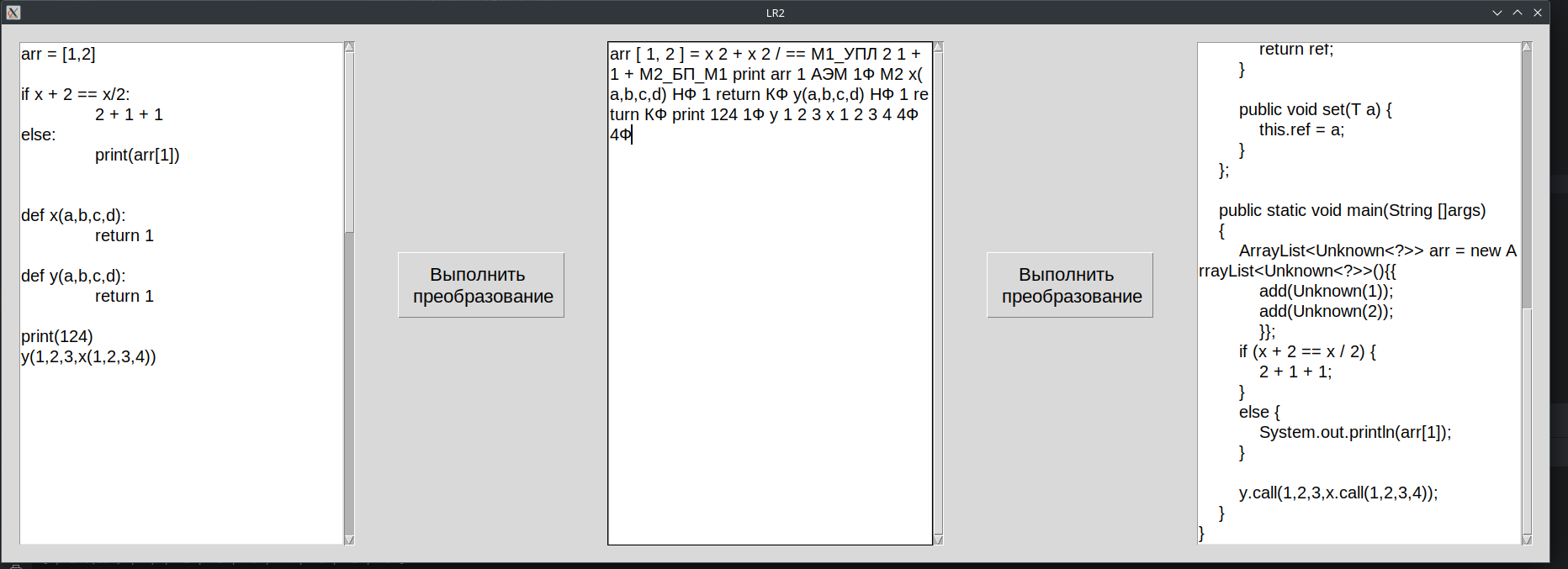


Рисунок 2 – Скриншот результата выполнения программы

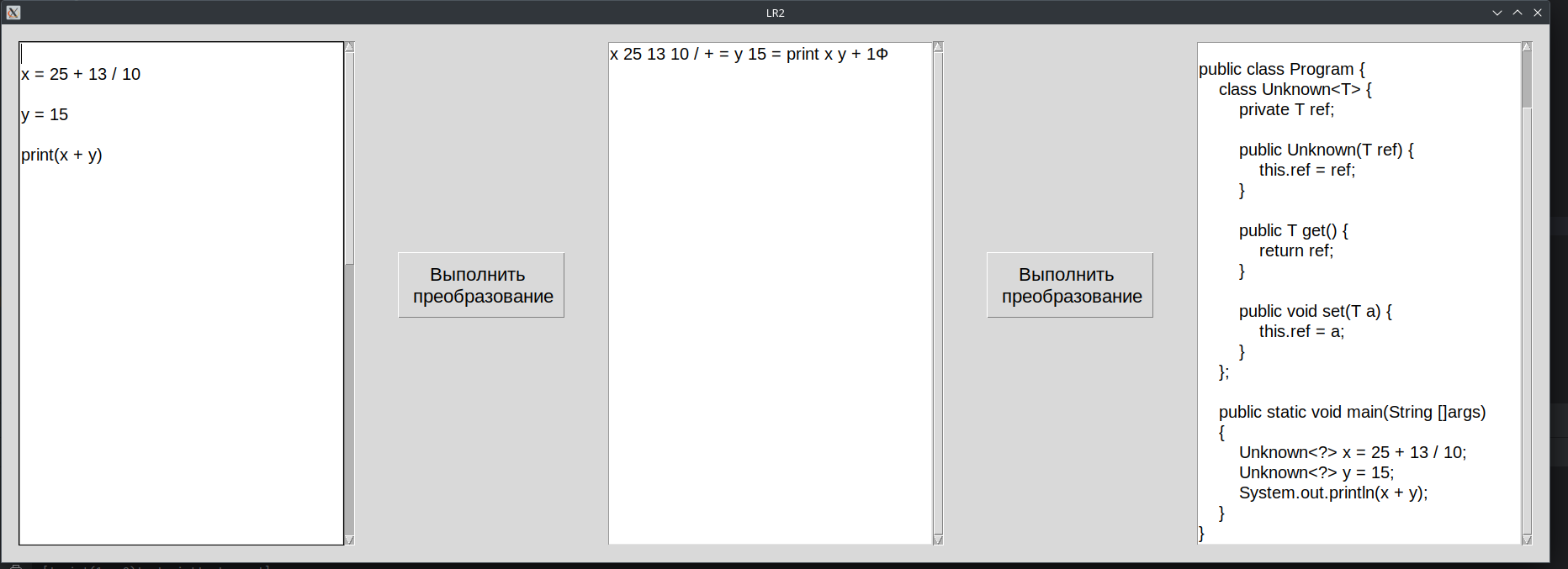


Рисунок 3 – Скриншот результата выполнения программы

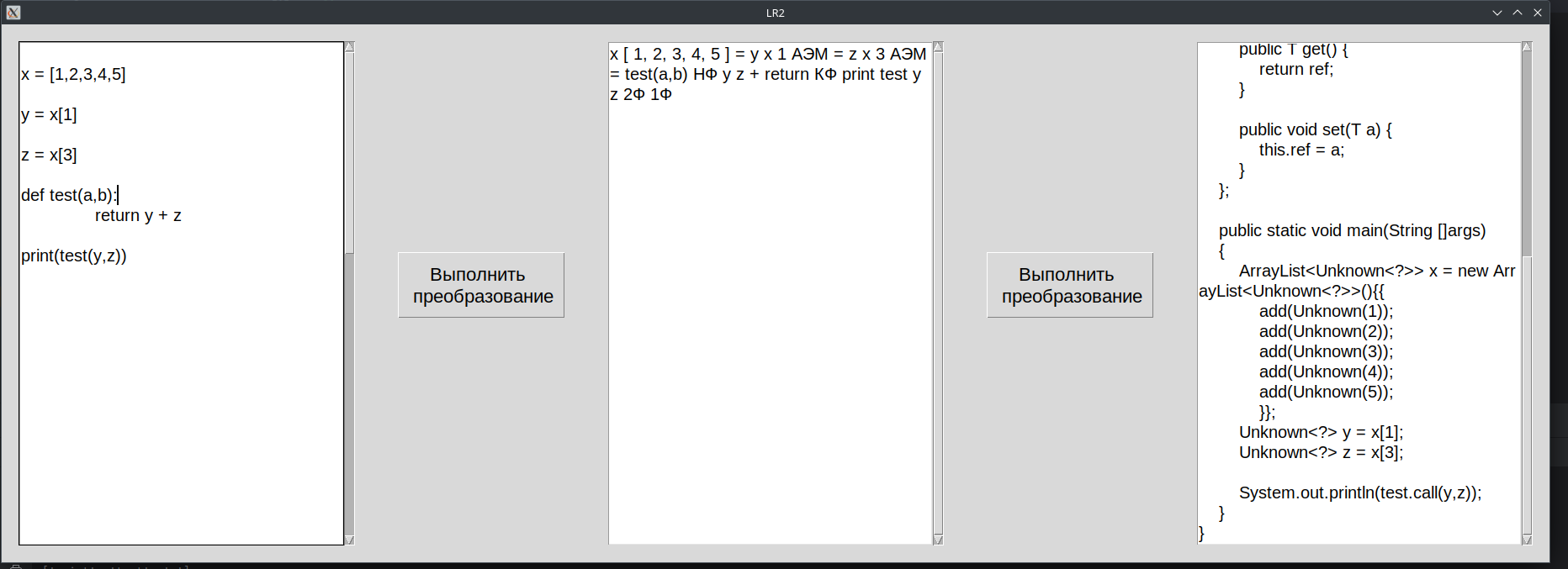


Рисунок 4 – Скриншот результата выполнения программы

**Приложение А   
Описание правил записи элементов заданного входного языка**

***Язык программирования Java***

**Идентификаторы**

Произвольная последовательность букв и цифр, начинающаяся с буквы. Может включать символы подчеркивания.

**Числовые константы целого типа**

Произвольная последовательность цифр без знака.

**Числовые константы вещественного типа, представленные с фиксированной точкой**

Последовательность цифр, включающая одну десятичную точку вида

123.45

**Символьные (строковые) константы**

Последовательность символов, заключенная в кавычки, расположенная в пределах одной строки, вида:

“Hello world 123 %\_ крот”

**Переменные с индексами (массивы и элементы массивов)**

Тип массива, квадратные скобки и идентификатор, после которого через запятую перечислены выражения‑индексы, вида:

int[] abc = {1,2,3,4}

abc[1+i]

**Комментарии (строчные и блочные)**

Блочные – последовательность символов, заключенная в /\* \*/, возможно содержащая несколько строк:

/\* Это комментарий,

Который содержит 2 строки \*/

Строчные – от символов «//» до конца строки.

i:=i+1; // это инкремент

**Обращения к процедурам и функциям пользователя**

Идентификатор, после которого в круглых скобках следует последовательность выражений‑аргументов, разделенных запятыми. Отсутствие аргументов допускается:

f(12, 4, i)

f()

**Арифметические операции**

Сложение +

Вычитание -

Умножение \*

Деление /

Целочисленное деление div

Остаток от деления mod

**Операции сравнения**

Меньше <

Больше >

Равно =

Не равно <>

Меньше или равно <=

Больше или равно >=

**Оператор присваивания**

Имеет вид «=». Слева стоит идентификатор, а справа – выражение. Заканчивается символом «;», например:

a=b+с;

**Операторы блока**

static/public { – начало блока

…

} - конец блока

**Оператор описания программы**

Программа начинается оператором public class с указанием имени программы. Затем могут идти описания данных, процедур и функций, а затем тело программы, заключенное в операторы блока, оканчивающееся точкой.

public class <идентификатор> {

…

}

**Операторы описания данных (идентификаторов и массивов)**

Начинается со слова типа и перечисления идентификаторов через запятую.

int a,c;

float b=100.2;

Типы переменных: int (целый), float (вещественный), string (строковый)

**Операторы описания процедур и функций**

Процедуры имеют заголовок вида

public/private <идентификатор> (<список формальных параметров>){

и тело – список операторов, заключенный в операторы блока

}

В остальном структура функций аналогична структуре процедур.

**Оператор безусловного перехода и метки**

goto <метка>;

Метка - идентификатор, расположенный в теле программы в начале строки, после которого стоит знак «:»:

label:

print(“Hi”);

**Оператор условного перехода**

Начинается с ключевого слова «if», имеет полный и неполный формат:

if <условие> then <оператор\_1> else <оператор\_2>;

if <условие> then <оператор\_1>;

**Арифметические циклы**

Синтаксис:

for (переменная = значение 1; значение 1 оператор сравнения значение 2; переменная оператор);

Оператор for вызывает оператор по одному разу для каждого значения в диапазоне от значения 1 до значения 2.

**Итерационные циклы с предусловием**

Синтаксис:

while (Логическое выражение) {

оператор;

}

Оператор после { будет выполняться до тех пор, пока логическое выражение принимает истинное значение (True). Логическое выражение является условием возобновления цикла. Его истинность проверяется каждый раз перед очередным повторением оператора цикла, который будет выполняться лишь до тех пор, пока логическое выражение истинно. Как только логическое выражение принимает значение ложь (False), осуществляется переход к оператору, следующему за while.

Выражение оценивается до выполнения оператора, так что если оно с самого начала было ложным (False), то оператор не будет выполнен ни разу.

**Итерационные циклы с постусловием**

Синтаксис:

do {

оператор;

оператор

...

Оператор;

} while (выражение);

Операторы между словами do и while повторяются, пока логическое выражение является ложным (False). Как только логическое выражение становится истинным (True), происходит выход из цикла.

Так как выражение оценивается после выполнения операторов, то в любом случае операторы выполнятся хотя бы один раз.

**Операторы завершения цикла**

Для всех операторов цикла выход из цикла осуществляется как вследствие естественного окончания оператора цикла, так и с помощью операторов перехода и выхода.

Break

Continue

Return

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

import ast

import re

from \_ast import Module, AST

from tkinter import \*

import tkinter.scrolledtext as st

binaryOperationNameToSymbol = {

'Mult': '\*',

'Add': '+',

'Sub': '-',

'Div': '/',

'FloorDiv': '//',

'Mod': '%',

'Pow': '\*\*',

}

compareEqNameToSymbol = {

'Lt': '<',

'LtE': '<=',

'Gt': '>',

'GtE': '>=',

'Eq': '==',

'NotEq': '!=',

}

compareBoolNameToSymbol = {

'And': 'and',

'Or': 'or',

}

compareUnaryNameToSymbol = {

'Not': 'not',

}

compareAugAssignNameToSymbol = {

'Add': '+=',

'Sub': '-=',

'Mult': '\*=',

'Div': '/=',

'MatMult': '\*\*='

}

def convert\_to\_rpn(node):

if isinstance(node, ast.BinOp):

left = convert\_to\_rpn(node.left)

right = convert\_to\_rpn(node.right)

return left + right + binaryOperationNameToSymbol.get(node.op.\_\_class\_\_.\_\_name\_\_) + " "

elif isinstance(node, ast.Num):

return str(node.n) + " "

elif isinstance(node, ast.Expr):

return convert\_to\_rpn(node.value)

elif isinstance(node, ast.FunctionDef):

args = ",".join([arg.arg for arg in node.args.args])

body = " ".join([convert\_to\_rpn(n) for n in node.body])

return node.name + "(" + args + ") " + "НФ " + body + "КФ "

elif isinstance(node, ast.Assign):

target = convert\_to\_rpn(node.targets[0])

value = convert\_to\_rpn(node.value)

return target + value + "= "

elif isinstance(node, ast.Compare):

left = convert\_to\_rpn(node.left)

ops = " ".join([compareEqNameToSymbol.get(op.\_\_class\_\_.\_\_name\_\_) for op in node.ops])

comparators = " ".join([convert\_to\_rpn(comp) for comp in node.comparators])

return left + comparators + ops + " "

elif isinstance(node, ast.Return):

value = convert\_to\_rpn(node.value)

return value + "return "

elif isinstance(node, ast.Yield):

value = convert\_to\_rpn(node.value)

return value + "yield "

elif isinstance(node, ast.AugAssign):

target = convert\_to\_rpn(node.target)

value = convert\_to\_rpn(node.value)

return target + value + compareAugAssignNameToSymbol.get(node.op.\_\_class\_\_.\_\_name\_\_) + " "

elif isinstance(node, ast.For):

target = convert\_to\_rpn(node.target)

print(node.target)

iter = convert\_to\_rpn(node.iter)

body = " ".join([convert\_to\_rpn(n) for n in node.body])

return target + iter + "in " + "НИЦ " + body + "КИЦ "

elif isinstance(node, ast.While):

test = convert\_to\_rpn(node.test)

body = " ".join([convert\_to\_rpn(n) for n in node.body])

return test + "НУЦ " + body + "КУЦ "

elif isinstance(node, ast.List):

elts = ", ".join([convert\_to\_rpn(elt).strip() for elt in node.elts])

return " [ " + elts + " ] "

elif isinstance(node, ast.Dict):

keys = [convert\_to\_rpn(key) for key in node.keys]

values = [convert\_to\_rpn(value) for value in node.values]

key\_value\_pairs = [k + ": " + v for k, v in zip(keys, values)]

pairs\_str = ", ".join(key\_value\_pairs)

return " { " + pairs\_str + " } "

elif isinstance(node, ast.Attribute):

value = convert\_to\_rpn(node.value)

attr = node.attr

return value + attr + " "

elif isinstance(node, ast.Num):

return str(node.n) + " "

elif isinstance(node, ast.Call):

args = [convert\_to\_rpn(arg) for arg in node.args]

return node.func.id + " " + "".join(args) + str(len(node.args)) + "Ф "

elif isinstance(node, ast.Expr):

return convert\_to\_rpn(node.value)

elif isinstance(node, ast.If):

test = convert\_to\_rpn(node.test)

body = " ".join([convert\_to\_rpn(n) for n in node.body])

if len(node.orelse) == 0:

return test + "M1\_УПЛ " + body + "М1 "

orelse = " ".join([convert\_to\_rpn(n) for n in node.orelse])

return test + "M1\_УПЛ " + body + "М2\_БП\_М1 " + orelse + "М2 "

elif isinstance(node, ast.Subscript):

value = convert\_to\_rpn(node.value)

slice\_value = convert\_to\_rpn(node.slice)

return value + slice\_value + "АЭМ "

elif isinstance(node, ast.Slice):

lower = convert\_to\_rpn(node.lower) if node.lower is not None else ""

upper = convert\_to\_rpn(node.upper) if node.upper is not None else ""

step = convert\_to\_rpn(node.step) if node.step is not None else ""

return lower + upper + step + "SLICE "

elif isinstance(node, ast.UnaryOp):

operand = convert\_to\_rpn(node.operand)

return operand + compareUnaryNameToSymbol.get(node.op.\_\_class\_\_.\_\_name\_\_) + " "

elif isinstance(node, ast.Name):

return node.id + " "

elif isinstance(node, ast.BoolOp):

if len(node.values) == 2:

left = convert\_to\_rpn(node.values[0])

right = convert\_to\_rpn(node.values[1])

return left + right + compareBoolNameToSymbol.get(node.op.\_\_class\_\_.\_\_name\_\_) + " "

else:

print(node.op, node.values)

elif isinstance(node, ast.Constant):

return node.value + " "

elif isinstance(node, ast.IfExp):

test = convert\_to\_rpn(node.test)

body = convert\_to\_rpn(node.body)

orelse = convert\_to\_rpn(node.orelse)

return test + "M1\_УПЛ " + body + "М2\_БП\_М1 " + orelse + "М2 "

else:

print(node)

return ""

def python\_to\_rpn(source\_code):

tree = ast.parse(source\_code)

rpn\_expression = ""

for node in tree.body:

rpn\_expression += convert\_to\_rpn(node)

return rpn\_expression.strip().replace(' ', ' ')

def rpn\_to\_java(source\_code):

result = ""

word\_buffer: list[str] = source\_code.replace('\n', '').split(" ")

currentTabulation = 8

try:

while (word\_buffer.index('[')):

start = word\_buffer.index('[')

end = word\_buffer.index(']')

arr = word\_buffer[start + 1:end]

print(arr)

res = """new ArrayList<Unknown<?>>(){{

$1

}}""".replace(

'$1',

str.join("\n" + " " \* (currentTabulation + 4),

["add(Unknown($1));".replace("$1", item.replace(',', '')) for item in arr])

)

word\_buffer = word\_buffer[0:start] + [res] + word\_buffer[end + 1: len(word\_buffer)]

except:

pass

unaryOperators = ['return', 'not', 'is']

print(word\_buffer)

stack = []

userFuncStack = []

pythonFuncs = {

'print': 'System.out.println'

}

for word in word\_buffer:

print(stack)

if word == '=':

tmp = (" " \* currentTabulation + "$3 $1 = $2;\n"

.replace('$2', stack.pop())

.replace('$1', stack.pop())

)

if tmp.find("new ArrayList<Unknown<?>>") != -1:

tmp = tmp.replace('$3', "ArrayList<Unknown<?>>")

else:

tmp = tmp.replace('$3', "Unknown<?>")

result += tmp

elif word in ["+", "-", "/", "\*", "\*\*", "==", "!=", ">=", "<=", "<", ">"]:

stack.append(("$1 $3 $2"

.replace('$2', stack.pop())

.replace('$1', stack.pop())

.replace('$3', word)

))

elif word == "M1\_УПЛ":

result += (" " \* currentTabulation + "if ($1) {\n"

.replace('$1', stack.pop())

)

currentTabulation += 4

elif re.search("^[0-9]+Ф$", word):

tmp = "("

amount = int(re.compile("[0-9]+").search(word).group(0))

print(stack,amount)

stack\_reversed = []

for i in range(0, amount):

stack\_reversed.append(stack.pop())

for i in range(0, amount):

tmp += "$1,".replace("$1", stack\_reversed.pop())

funcName = stack.pop()

if funcName in userFuncStack:

funcName += '.call'

stack.append((funcName + tmp + ")").replace(',)', ')'))

elif word == "М2\_БП\_М1":

if (len(stack)):

result += " " \* currentTabulation + stack.pop() + ";\n"

currentTabulation -= 4

result += " " \* currentTabulation + "}\n" + " " \* currentTabulation + "else {\n"

currentTabulation += 4

elif word == 'АЭМ':

stack.append(("$1[$2]"

.replace('$2', stack.pop())

.replace('$1', stack.pop())

))

elif word == "М2":

if (len(stack)):

result += " " \* currentTabulation + stack.pop() + ";\n"

currentTabulation -= 4

result += " " \* currentTabulation + "}\n"

elif word == "НФ":

func = stack.pop()

funcName = func.split('(')[0]

userFuncStack.append(funcName)

funcArgs = str.join(", ", ["Unknown<?> $1".replace("$1", arg) for arg in

("(" + func.split('(')[1]).replace('(', "").split(",")])

tmp = " " \* currentTabulation + """static class $3 {

$1""".replace("$3", funcName)

currentTabulation += 4

tmp = tmp.replace("$1", " " \* currentTabulation + "public static Unknown<?> call(" + funcArgs + "{\n")

currentTabulation += 4

result += tmp

elif word == 'КФ':

if (len(stack)):

result += " " \* currentTabulation + stack.pop() + ";\n"

currentTabulation -= 4

result += " " \* currentTabulation + "}\n"

currentTabulation -= 4

result += " " \* currentTabulation + "}mark\n"

elif word in unaryOperators:

result += (" " \* currentTabulation + "$1 $2;\n"

.replace('$2', stack.pop())

.replace('$1', word)

)

else:

stack.append(word)

if (len(stack)):

result += " " \* currentTabulation + stack.pop() + ";"

for funcKey in pythonFuncs.keys():

result = result.replace(funcKey, pythonFuncs.get(funcKey))

print(stack)

funcie = None

if (re.search('static class[\s\S]\*}mark', result)):

funcie = re.search('static class[\s\S]\*}mark', result).group(0).replace('mark', ';\n')

funcieToRepl = re.search('static class[\s\S]\*}mark', result).group(0).split('};')

x= """

import java.util.ArrayList;

$2

public class Program {

class Unknown<T> {

private T ref;

public Unknown(T ref) {

this.ref = ref;

}

public T get() {

return ref;

}

public void set(T a) {

this.ref = a;

}

};

public static void main(String []args)

{

$1

}

}""".replace("$1", result)

if (funcie):

x = x.replace("$2", funcie)

for a in funcieToRepl:

x = x.replace(a, '')

else:

x = x.replace("$2", "")

return x

def prog():

f = open('./resources/python.txt', 'r')

input\_sequence = f.read()

f.close()

out\_seq = python\_to\_rpn(input\_sequence)

# файл, содержащий обратную польскую запись

f = open('gen/rpn.txt', 'w')

f.write(out\_seq)

f.close()

def prog2():

f = open('./gen/rpn.txt', 'r')

input\_sequence = f.read()

f.close()

out\_seq = rpn\_to\_java(input\_sequence)

# файл, содержащий обратную польскую запись

f = open('gen/res.txt', 'w')

f.write(out\_seq)

f.close()

def write\_txt(data, to):

with open(to, 'w') as file:

file.write(data)

def clicked():

write\_txt(codetxt.get("1.0", "end"), 'resources/python.txt')

opzstext.delete("1.0", END)

prog()

f1 = open('gen/rpn.txt', 'r')

text = f1.read()

opzstext.insert("1.0", text)

f1.close()

def clicked2():

write\_txt(opzstext.get("1.0", "end"), 'gen/rpn.txt')

restext.delete("1.0", END)

prog2()

f1 = open('gen/res.txt', 'r')

text = f1.read()

restext.insert("1.0", text)

f1.close()

window = Tk()

window.title("LR2")

f1 = open('resources/python.txt', 'r')

text = f1.read()

f2 = open('gen/rpn.txt', 'r')

text2 = f2.read()

f3 = open('gen/res.txt', 'r')

text3 = f3.read()

window.geometry('1840x640')

codetxt = st.ScrolledText(window, font=("Arial", 18))

codetxt.insert("1.0", text)

codetxt.place(x=20, y=20, width=400, height=600)

opzstext = st.ScrolledText(window, font=("Arial", 18))

opzstext.place(x=720, y=20, width=400, height=600)

opzstext.insert("1.0", text2)

restext = st.ScrolledText(window, font=("Arial", 18))

restext.place(x=1420, y=20, width=400, height=600)

restext.insert("1.0", text3)

btngo = Button(window, text="Выполнить \n преобразование", command=clicked, font=("Arial", 20))

btngo.place(x=510 + 150 - 100 - 90, y=10 + 300 - 40, width=200, height=80)

btngo2 = Button(window, text="Выполнить \n преобразование", command=clicked2, font=("Arial", 20))

btngo2.place(x=510 + 350 - 100 - 90 + 500, y=10 + 300 - 40, width=200, height=80)

window.mainloop()