Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Кубанский государственный университет»

Кафедра вычислительных технологий

**ОТЧЕТ**

о выполнении лабораторной работы №2

по дисциплине “Методы разработки трансляторов”

Выполнил: ст. гр. 36/2

Баева Д. Н.

Проверил: Вишняков Ю.М

Краснодар

2023

**Содержание**

[1 Вариант задания 3](#_Toc97893076)

[2 Понятие обратной польской записи 4](#_Toc97893077)

[3 Алгоритм Дейкстры 5](#_Toc97893078)

[4 Перевод операторов в ОПЗ 8](#_Toc97893079)

[5 Результаты экспериментов 9](#_Toc97893080)

[Приложение А Листинг программы и комментарии к нему 12](#_Toc97893082)

**1 Вариант задания**

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

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| № варианта | Входной язык | Выходной язык |
| 27 | R | C++ |

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:

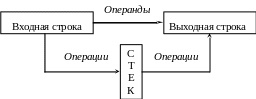


Рисунок 1 – Суть алгоритма Дейкстры

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 |
| (  for  if  while  [  АЭМ  Ф  { | 0 |
| )  ,  :  else  ] | 1 |
| =  <- | 2 |
| | | 3 |
| & | 4 |
| ! | 5 |
| <  <=  !=  ==  >  >= | 6 |
| +  - | 7 |

Продолжение таблицы 2

|  |  |
| --- | --- |
| Входной элемент | Приоритет |
| \*  ^  /  %%  %/% | 8 |
| }  function  return  print  paste  c | 9 |

**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:.

Оператор цикла с счетчиком FOR (A; B; C) D; заменяется на   
A; WHILE (B) { D; C; }.

**5 Результаты экспериментов**

Пример работы программы для программы для тестирования: на рисунке 1 представлен скриншот файла, содержащего текст на входном языке программирования, на рисунке 2 – скриншот результата работы программы.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как диаграмма

Автоматически созданное описаниеРисунок 1 – Скриншот файла, содержащего текст на входном языке программирования

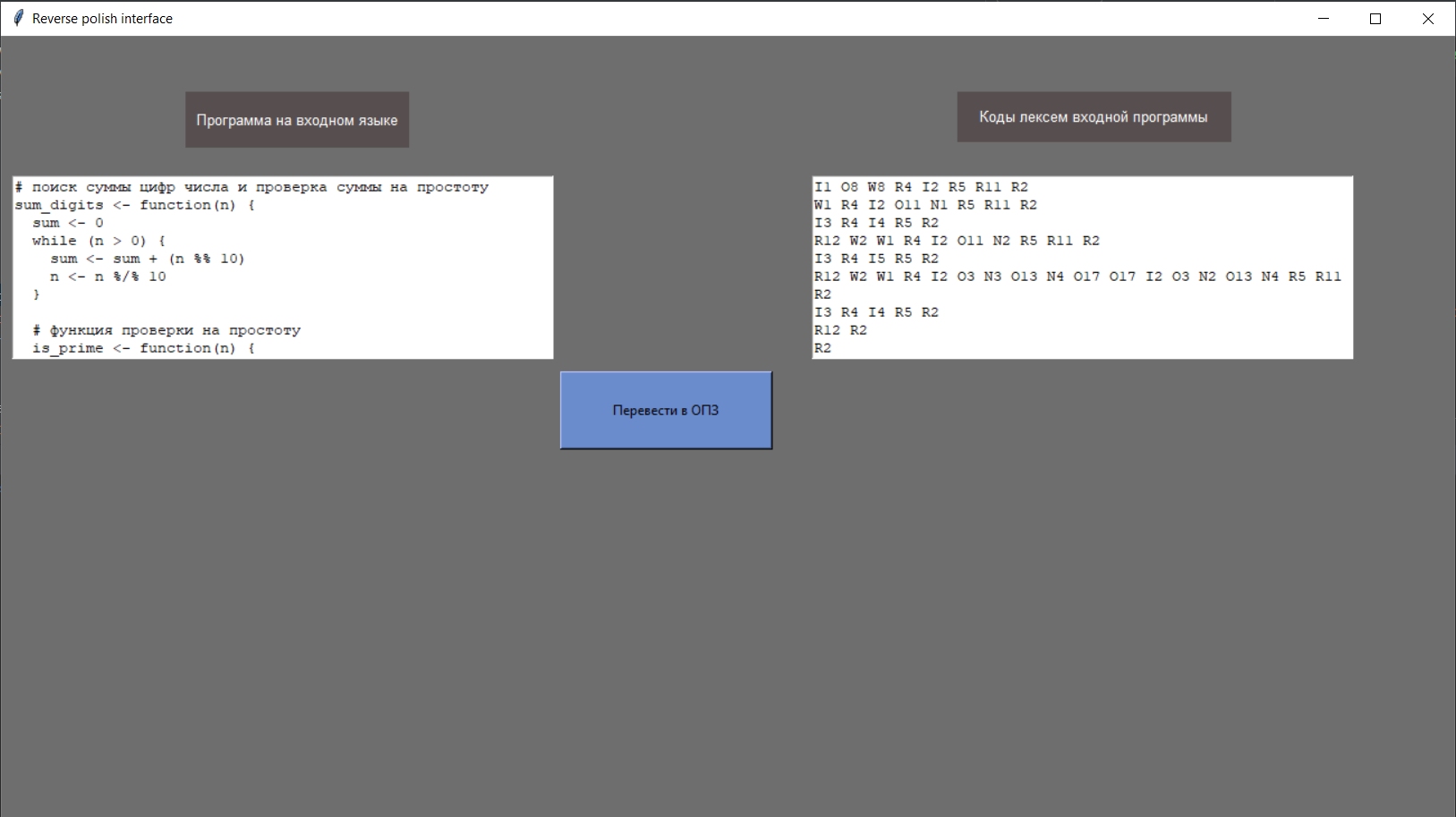
Рисунок 2 – Результат работы программы

Рисунок 3 – Скриншот начального интерфейса

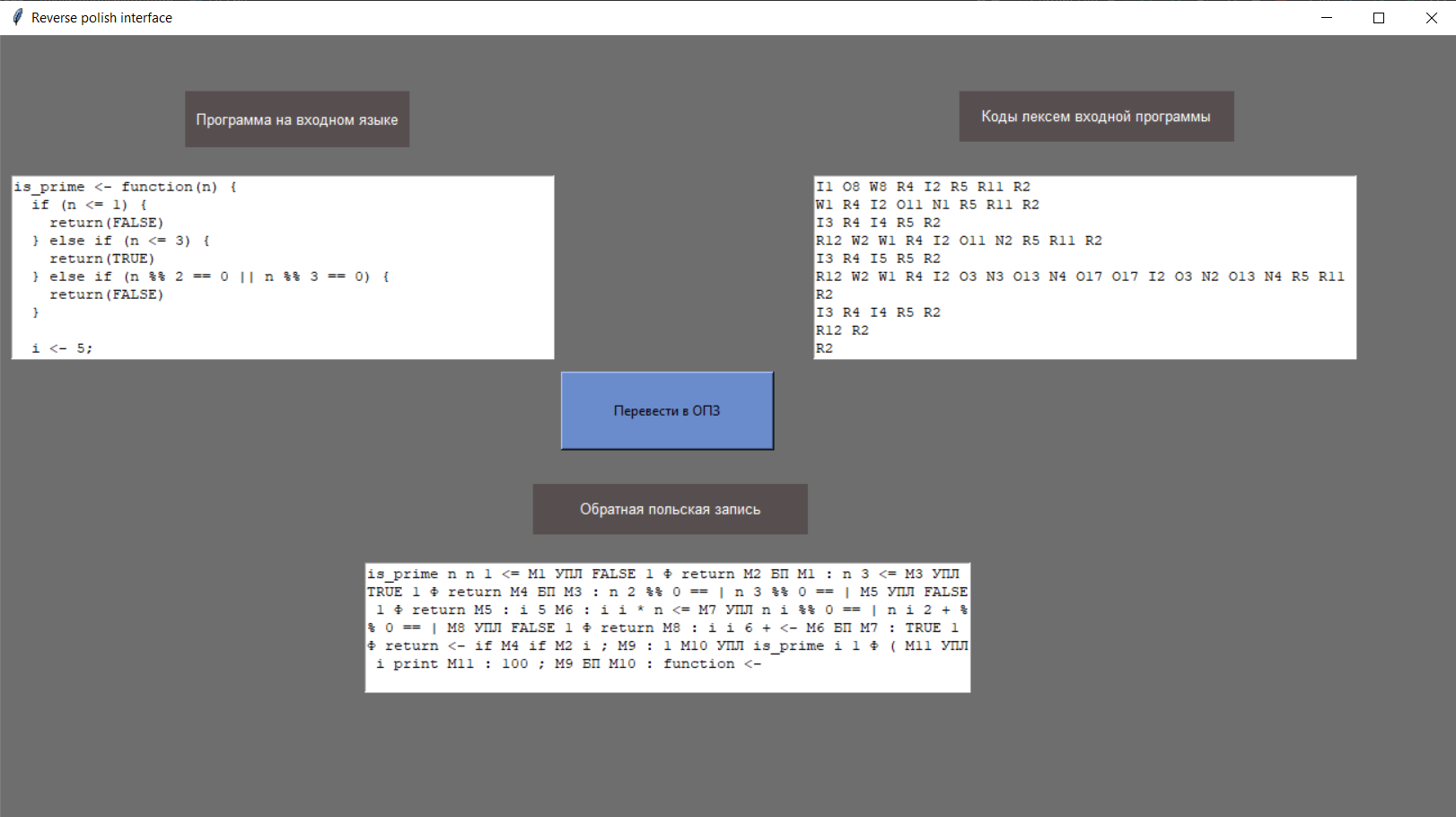


Рисунок 4 – Скриншот интерфейса после выполнения программы

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

**import** json  
**import** re  
**import** tkinter  
**from** tkinter **import** \*  
  
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** [**')'**, **','**, **':'**, **'else'**, **']'**]:  
 **return** 1  
 **if** token == **'=' or** token == **'<-'**:  
 **return** 2  
 **if** token == **'|'**:  
 **return** 3  
 **if** token == **'&'**:  
 **return** 4  
 **if** token == **'!'**:  
 **return** 5  
 **if** token **in** [**'!='**, **'<'**, **'<='**, **'=='**, **'>'**, **'>='**]:  
 **return** 6  
 **if** token == **'+' or** token == **'-'**:  
 **return** 7  
 **if** token **in** [**'\*'**, **'^'**, **'/'**, **'%%'**]:  
 **return** 8  
 **if** token **in** [**'}'**, **'function'**,**'return'**, **'print'**,]:  
 **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'**:  
 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'**)  
 **if** re.match(**r'^var'**, stack[-1]):  
 operand\_count += 1  
 **elif** t[i] == **'goto'**:  
 out\_seq += t[i + 1] + **' БП '** i += 2  
 **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] == **'var'**:  
 stack.append(t[i] + **' '** + str(proc\_num) + **' '** + str(proc\_level))  
 operand\_count = 1  
 is\_description\_var = **True  
 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:  
 **while not**(re.match(**r'^var'**, stack[-1])):  
 out\_seq += stack.pop() + **' '** 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() + **' '***# файл, содержащий обратную польскую запись*f = open(**'reverse\_polish\_entry.txt'**, **'w'**, encoding=**'ANSI'**)  
f.write(out\_seq)  
f.close()  
  
*# действие после нажатия на кнопку***def** rpn():  
 label4 = tkinter.Label(window, text=**'Обратная польская запись'**, font=(**"Arial"**, 10), foreground=**"white"**,background=**"#574f4f"**)  
 label4.place(x=475, y=400, width=245, height=45)  
 rpn\_text = open(**'reverse\_polish\_entry.txt'**,encoding=**'ANSI'**).readlines()  
 rpn\_text = **''**.join(rpn\_text)  
 textline1 = Text(window, height=7, width=67)  
 textline1.insert(1.0, rpn\_text)  
 textline1.place(x=325, y=470)  
  
  
*# создание окна интерфейса*window = tkinter.Tk()  
window.geometry(**'1300x700'**)  
window.title(**"Reverse polish interface"**)  
window.configure(bg=**'#6e6e6e'**)  
  
*# расположение всех необходимых текстовых окошек, лэйблов*label2 = tkinter.Label(window, text=**'Программа на входном языке'**,font=(**"Arial"**, 10),foreground=**"white"**, background=**"#574f4f"**)  
label2.place(x=165, y=50, width=200, height=50)  
  
text = open(**'R.txt'**,encoding=**'utf-8'**).readlines()  
text=**''**.join(text)  
textline=Text(window,height=10, width=60)  
textline.insert(1.0,text)  
textline.place(x=10, y=125)  
  
label3 = tkinter.Label(window, text=**'Коды лексем входной программы'**, font=(**"Arial"**, 10), foreground=**"white"**,  
 background=**"#574f4f"**)  
label3.place(x=855, y=50, width=245, height=45)  
  
token\_text = open(**'tokens.txt'**, encoding=**'utf-8'**).readlines()  
token\_text = **''**.join(token\_text)  
textline = Text(window, height=10, width=60)  
textline.insert(1.0, token\_text)  
textline.place(x=725, y=125)  
  
button = tkinter.Button(window, text=**'Перевести в ОПЗ'**, bg=**'#6a8bcc'**,command=rpn)  
button.place(x=500, y=300, width=190, height=70)  
  
window.mainloop()