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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. М. Нагалевский

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 |
| 42 | Java | 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:

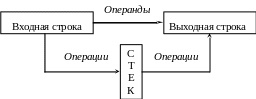


Рисунок 1 – Суть алгоритма Дейкстры

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  begin | 0 |
| )  ,  ;  do  else  ] | 1 |
| = | 2 |
| || | 3 |
| && | 4 |
| ! | 5 |

Продолжение таблицы 2

|  |  |
| --- | --- |
| Входной элемент | Приоритет |
| <  <=  !=  =  >  >= | 6 |
| +  -  +=  -=  \*=  /= | 7 |
| \*  /  % | 8 |
| }  public  static  void  procedure  int  double  boolean  string  float  args  return  system | 9 |

Продолжение таблицы 2

|  |  |
| --- | --- |
| out  println  main | 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:.

Обработка оператора цикла с постусловием REPEAT оператор UNTIL выражение; можно заменить последовательностью операторов

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

Обработка оператора цикла с счетчиком FOR переменная = значение1 TO значение2 DO оператор;:

1. FOR переменная из входной строки заносится в стек.
2. Символ TO выталкивает в выходную строку все операции из стека до ближайшего FOR переменная. В стеке к FOR переменная добавляется рабочая метка Mi и после этого в выходную строку записывается Mi: переменная.
3. Символ DO выталкивает в выходную строку все операции из стека до ближайшего FOR переменная Mi. В стеке к FOR переменная Mi добавляется рабочая метка Mi+1 и после этого в выходную строку записывается <= Mi+1 УПЛ.
4. Символы ‘;’ и END указывают на конец оператора цикла с счетчиком и выталкивают из стека все символы до ближайшего FOR переменная Mi Mi+1, при этом сам FOR уничтожается, а в выходную строку помещается метка переменная переменная 1 + := Mi БП Mi+1:.

**5 Результаты экспериментов**

Примеры работы программы для программы для тестирования.

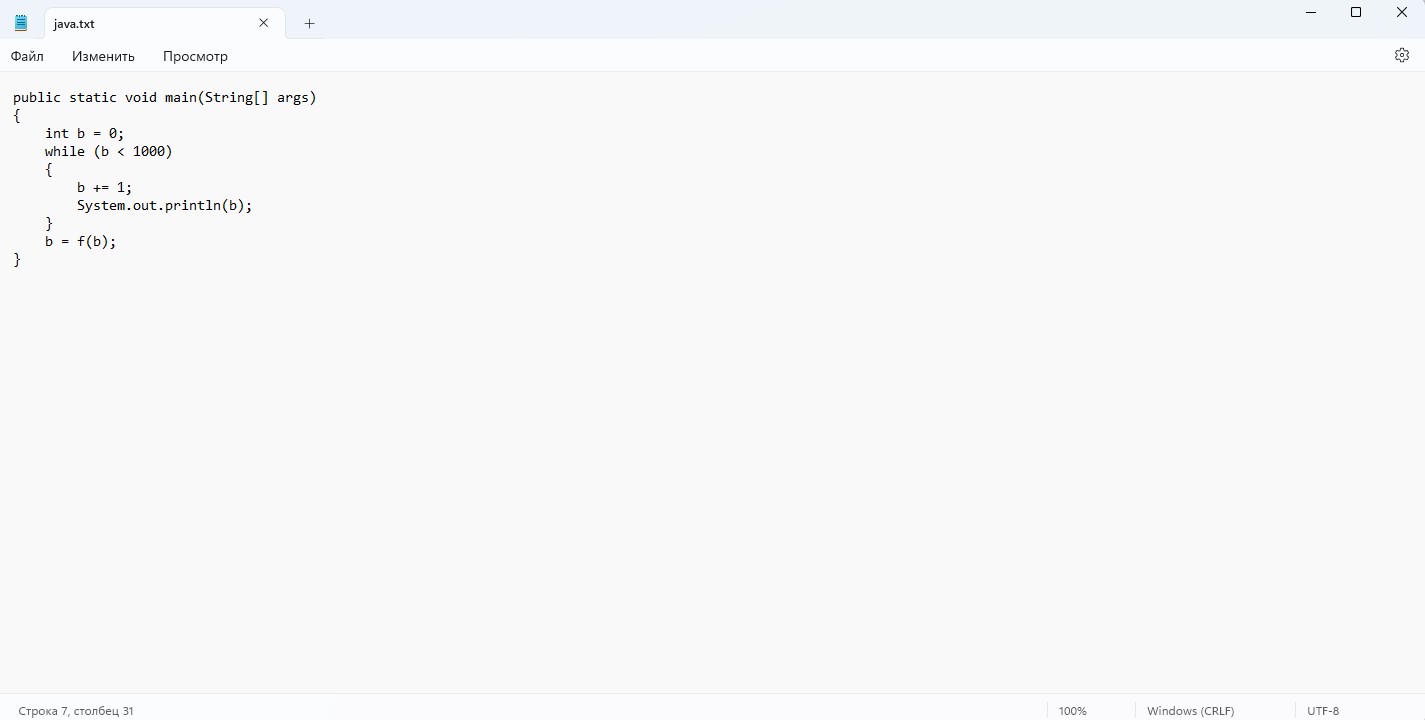


Рисунок 2 – Скриншот файла №1, содержащего текст на входном языке программирования

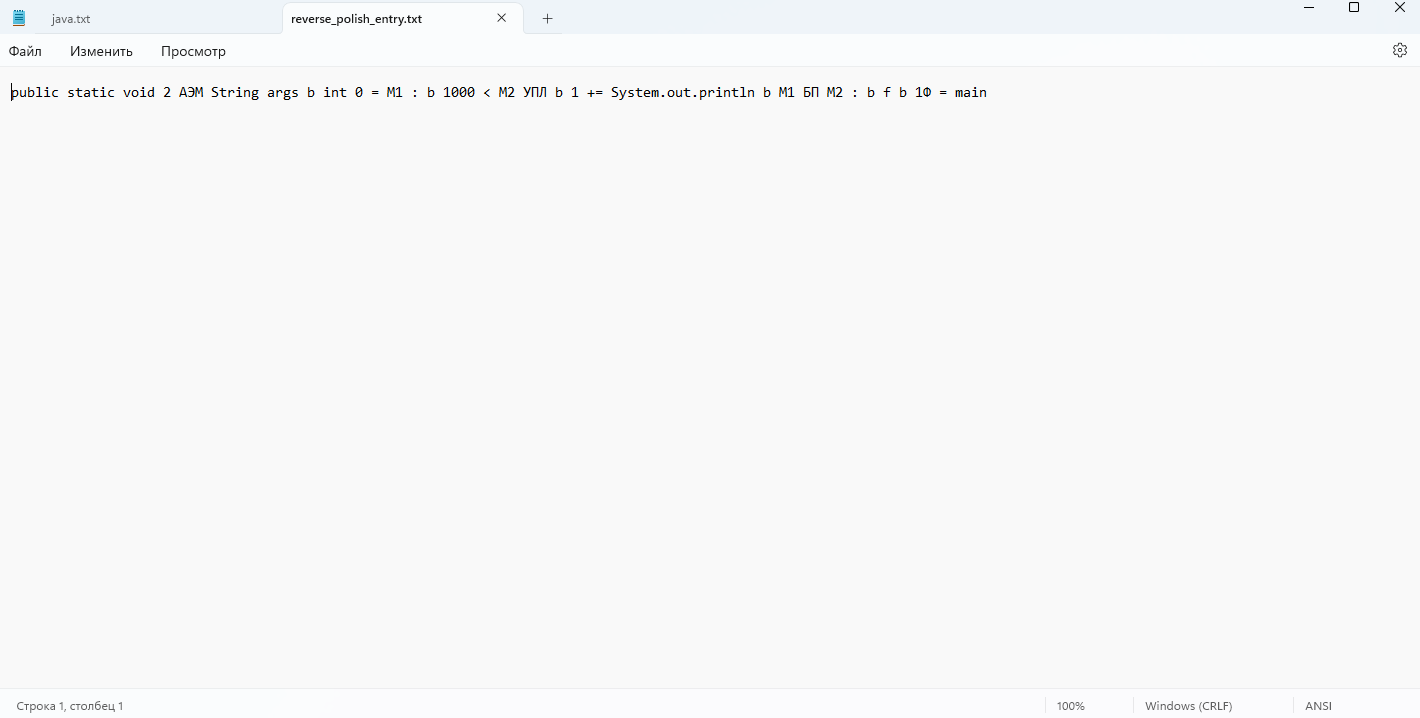


Рисунок 3 – Результат работы программы №1

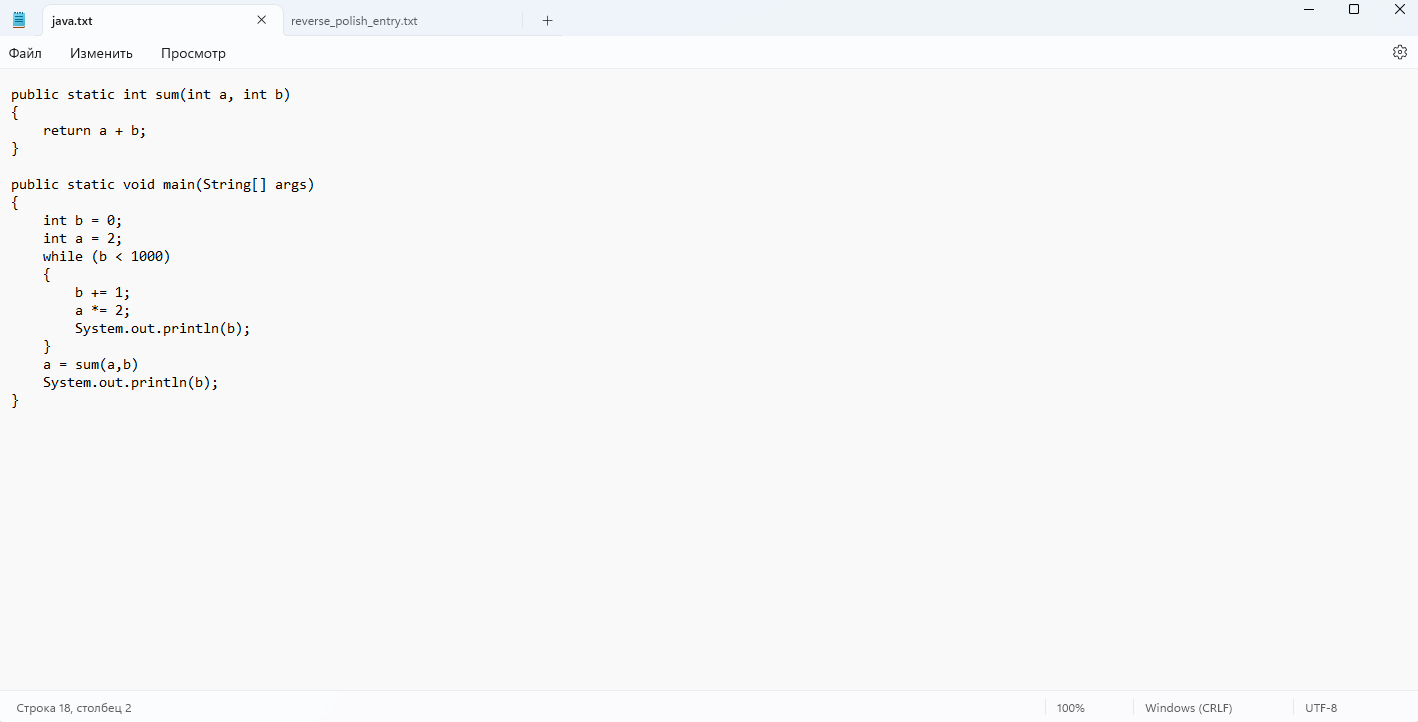


Рисунок 4 – Скриншот файла №2, содержащего текст на входном языке программирования

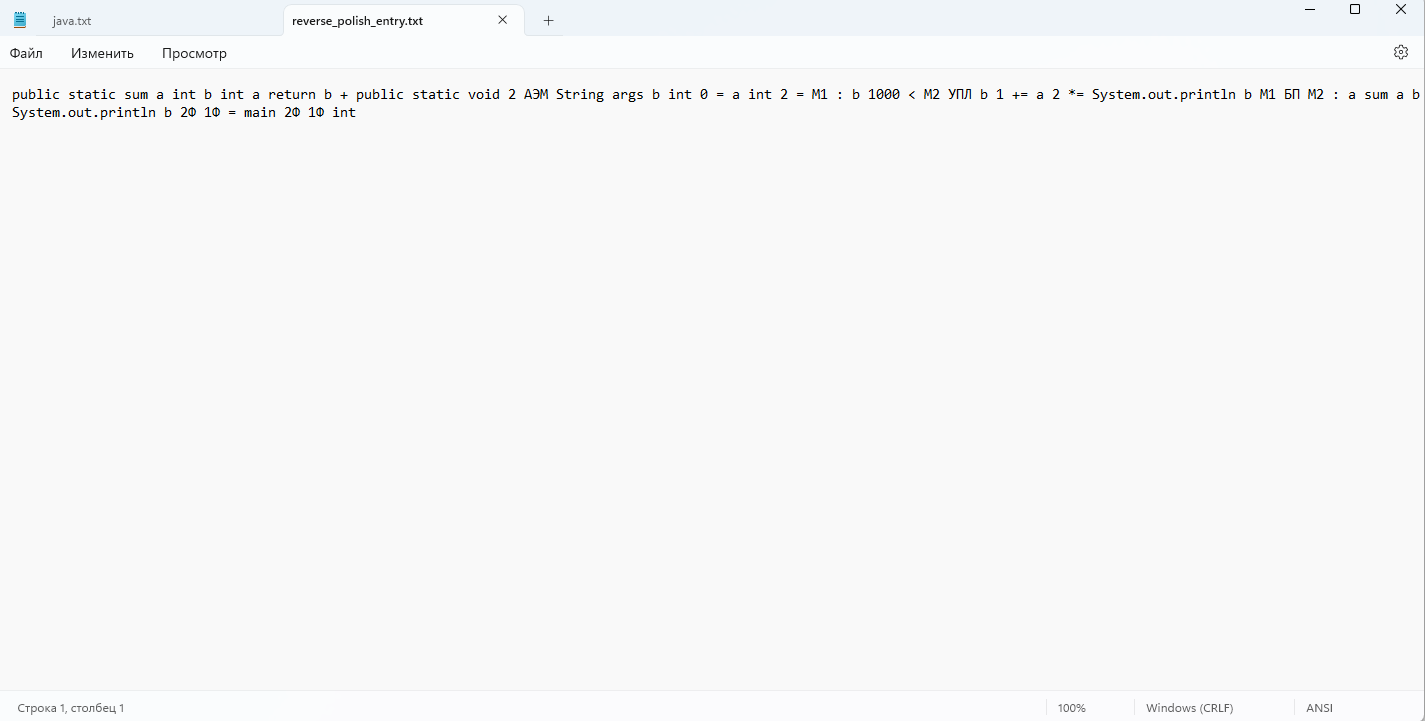


Рисунок 5 – Результат работы программы №2

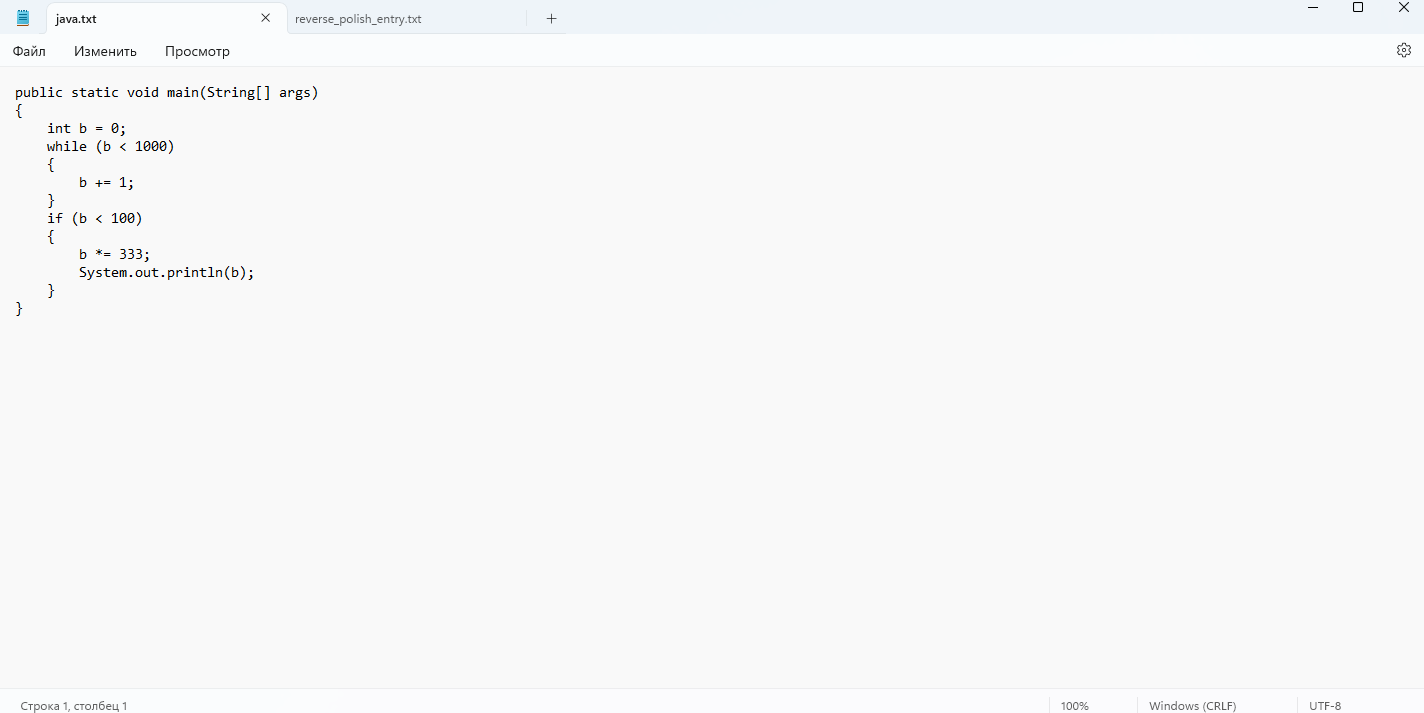


Рисунок 6 – Скриншот файла №3, содержащего текст на входном языке программирования

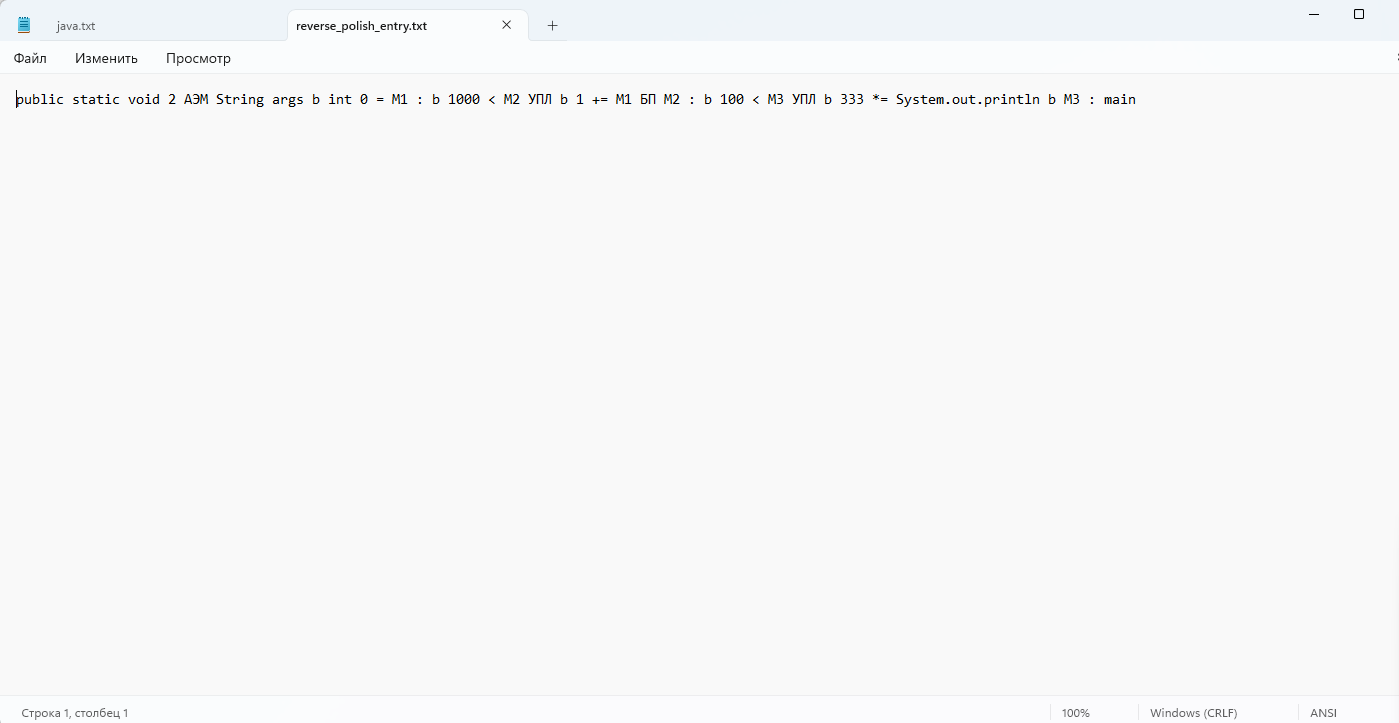


Рисунок 7 – Результат работы программы №3

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

**import** lab1  
**import** json  
**import** re  
  
CLASSES\_OF\_TOKENS = [**'W'**, **'I'**, **'O'**, **'R'**, **'N'**, **'C'**]  
  
**def** is\_identifier(token):  
 **return** re.match(**r'^I\d+$'**, inverse\_tokens[token]) **or** \  
 token **in** [**'abs'**, **'cos'**, **'exp'**, **'ln'**, **'read'**, **'readln'**, **'sin'**, **'sqrt'**, \  
 **'write'**, **'writeln'**]  
  
**def** get\_priority(token):  
 **if** token **in** [**'('**, **'for'**, **'if'**, **'repeat'**, **'while'**, **'['**, **'АЭМ'**, **'Ф'**, **'begin'**]:  
 **return** 0  
 **if** token **in** [**')'**, **','**, **';'**, **'do'**, **'downto'**, **'else'**, **'then'**, **'to'**, **'until'**, **']'**]:  
 **return** 1  
 **if** token == **':=' or** token == **'goto'**:  
 **return** 2  
 **if** token == **'or'**:  
 **return** 3  
 **if** token == **'and'**:  
 **return** 4  
 **if** token == **'not'**:  
 **return** 5  
 **if** token **in** [**'<'**, **'<='**, **'<>'**, **'='**, **'>'**, **'>='**]:  
 **return** 6  
 **if** token == **'+' or** token == **'-'**:  
 **return** 7  
 **if** token **in** [**'\*'**, **'/'**, **'div'**, **'mod'**]:  
 **return** 8  
 **if** token **in** [**'end'**, **'function'**, **'procedure'**, **'program'**, **'var'**]:  
 **return** 9  
 **return** -1  
  
**def** description\_var(block, out\_seq):  
 i = 0  
 var\_count = 0  
 **while** i < len(block):  
 **while** i < len(block) **and** block[i] != **':'**:  
 **if not**(block[i] **in** [**'\n'**, **','**, **';'**]):  
 out\_seq += block[i] + **' '** var\_count += 1  
 i += 1  
 **if** i == len(block):  
 **break** i += 1  
 **if** block[i] == **'array'**:  
 i += 1  
 aem\_count = 1  
 **while** block[i] != **']'**:  
 **if** block[i] == **','**:  
 out\_seq += **'.. '** aem\_count += 1  
 **else**:  
 out\_seq += block[i] + **' '** i += 1  
 i += 2  
 aem\_count += 1  
 out\_seq += **'.. '** + str(aem\_count) + **' АЭМ '** out\_seq += str(var\_count) + **' '** + block[i] + **' '** var\_count = 0  
 i += 1  
 **return** out\_seq  
  
*# 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\_level = 1  
func\_count = tag\_count = proc\_num = if\_count = while\_count = repeat\_count = \  
 for\_count = begin\_count = end\_count = 0  
is\_func = **False  
while** i < len(t):  
 p = get\_priority(t[i])  
 **if** t[i] == **'.'**:  
 stack.pop()  
 out\_seq += **'КП '  
 elif** p == -1:  
 **if** t[i] != **'\n'**:  
 out\_seq += t[i] + **' '  
 if** is\_func **and** t[i + 1] == **'('**:  
 i += 2  
 block = []  
 **while** t[i] != **')'**:  
 block.append(t[i])  
 i += 1  
 out\_seq = description\_var(block, out\_seq)  
 i -= 1  
 is\_func = **False  
 else**:  
 **if** t[i] == **'['**:  
 aem\_count += 1  
 stack.append(str(aem\_count) + **' АЭМ'**)

**elif** t[i] == **']'**:  
 **while not**(re.match(**r'^\d+ АЭМ$'**, stack[-1])):  
 out\_seq += stack.pop() + **' '** out\_seq += stack.pop() + **' '** aem\_count = 1  
 **elif** t[i] == **'('**:  
 **if** is\_identifier(t[i - 1]):  
 func\_count += 1  
 stack.append(str(func\_count) + **' Ф'**)  
 **else**:  
 stack.append(t[i])  
 **elif** t[i] == **')'**:  
 **if** t[i + 1] == **':'**:  
 i += 2  
 out\_seq += **'1 '** + t[i] + **' КО '  
 else**:  
 **while** stack[-1] != **'(' and not**(re.match(**r'^\d+ Ф$'**, stack[-1])):  
 out\_seq += stack.pop() + **' '  
 if** re.match(**r'^\d+ Ф$'**, stack[-1]):  
 stack.append(str(func\_count + 1) + **' Ф'**)  
 func\_count = 0  
 stack.pop()  
 **elif** t[i] == **','**:  
 **while not**(re.match(**r'^\d+ АЭМ$'**, stack[-1])) **and** \  
 **not**(re.match(**r'^\d+ Ф$'**, stack[-1])):  
 out\_seq += stack.pop() + **' '  
 if** re.match(**r'^\d+ АЭМ$'**, stack[-1]):  
 aem\_count += 1  
 stack.append(str(aem\_count) + **' АЭМ'**)  
 **if** re.match(**r'^\d+ Ф$'**, stack[-1]):  
 func\_count += 1  
 stack.append(str(func\_count) + **' Ф'**)  
 stack.pop()  
 **elif** t[i] == **'goto'**:  
 out\_seq += t[i + 1] + **' БП '** i += 2  
 **elif** t[i] == **'if'**:  
 stack.append(t[i])  
 if\_count += 1  
 **elif** t[i] == **'then'**:  
 **while** stack[-1] != **'if'**:  
 out\_seq += stack.pop() + **' '** tag\_count += 1  
 stack[-1] += **' М'** + str(tag\_count)  
 out\_seq += **'М'** + str(tag\_count) + **' УПЛ '  
 elif** t[i] == **'else'**:  
 **while not**(re.match(**r'^if М\d+$'**, 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  
 **elif** t[i] == **'do'**:  
 **while not**(re.match(**r'^while М\d+$'**, stack[-1])) **and** \  
 **not**(re.match(**r'^for [a-z][a-z\d]+ М\d+$'**, stack[-1])):  
 out\_seq += stack.pop() + **' '** tag\_count += 1  
 **if** re.match(**r'^for [a-z][a-z\d]+ М\d+$'**, stack[-1]):  
 out\_seq += **'<= '** out\_seq += **'М'** + str(tag\_count) + **' УПЛ '** stack[-1] += **' М'** + str(tag\_count)  
 **elif** t[i] == **'repeat'**:  
 tag\_count += 1  
 stack.append(t[i] + **' М'** + str(tag\_count))  
 out\_seq += **'М'** + str(tag\_count) + **' : '** repeat\_count += 1  
 **elif** t[i] == **'until'**:  
 **while not**(re.match(**r'^repeat М\d+$'**, stack[-1])):  
 out\_seq += stack.pop() + **' '  
 elif** t[i] == **'for'**:  
 stack.append(t[i] + **' '** + t[i + 1])  
 for\_count += 1  
 **elif** t[i] == **'to'**:  
 **while not**(re.match(**r'^for [a-z][a-z\d]+$'**, stack[-1])):  
 out\_seq += stack.pop() + **' '** variable = re.search(**'[a-z][a-z\d]'**, stack[-1]).group(0)  
 tag\_count += 1  
 stack[-1] += **' М'** + str(tag\_count)  
 out\_seq += **'М'** + str(tag\_count) + **' : '** + variable + **' '  
 elif** t[i] == **'var'**:  
 block = []  
 i += 1  
 **while not**(t[i] **in** [**'begin'**, **'function'**, **'procedure'**]):  
 block.append(t[i])  
 i += 1  
 i -= 1  
 out\_seq = description\_var(block, out\_seq)  
 out\_seq += str(proc\_num) + **' '** + str(proc\_level) + **' КО '  
 elif** t[i] **in** [**'function'**, **'procedure'**, **'program'**]:  
 **if** t[i] == **'function' or** t[i] == **'procedure'**:  
 is\_func = **True** proc\_num += 1  
 stack.append(**'PROC '** + str(proc\_num) + **' '** + str(proc\_level))  
 **elif** t[i] == **'begin'**:  
 begin\_count += 1  
 proc\_level = begin\_count - end\_count + 1  
 stack.append(t[i])  
 **elif** t[i] == **'end'**:  
 end\_count += 1  
 proc\_level = begin\_count - end\_count + 1  
 **while** stack[-1] != **'begin'**:  
 out\_seq += stack.pop() + **' '** stack.pop()  
 **if not**(if\_count > 0 **and** re.match(**r'^if М\d+$'**, stack[-1])) **and** \  
 **not**(while\_count > 0 **and** re.match(**r'^while М\d+ М\d+$'**, stack[-1])) **and** \  
 **not**(for\_count > 0 **and** re.match(**r'^for [a-z][a-z\d]+ М\d+ М\d+$'**, stack[-1])):  
 stack.append(t[i])  
 **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** if\_count > 0 **or** while\_count > 0 **or** repeat\_count > 0 **or** \  
 for\_count > 0:  
 **while not**(len(stack) > 0 **and** stack[-1] == **'begin'**) **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])) **and** \  
 **not**(repeat\_count > 0 **and** re.match(**r'^repeat М\d+$'**, stack[-1])) **and** \  
 **not**(for\_count > 0 **and** re.match(**r'^for [a-z][a-z\d]+ М\d+ М\d+$'**, stack[-1])):  
 out\_seq += stack.pop() + **' '  
 if** if\_count > 0 **and** re.match(**r'^if М\d+$'**, stack[-1]):  
 tag = re.search(**'М\d+'**, stack[-1]).group(0)  
 out\_seq += tag + **' : '** if\_count -= 1  
 **if** while\_count > 0 **and** re.match(**r'^while М\d+ М\d+$'**, stack[-1]):  
 tag = re.findall(**'М\d+'**, stack[-1])  
 out\_seq += tag[0] + **' БП '** + tag[1] + **' : '** while\_count -= 1  
 **if** repeat\_count > 0 **and** re.match(**r'^repeat М\d+$'**, stack[-1]):  
 tag\_count += 1  
 out\_seq += str(tag\_count) + **' УПЛ '** tag = re.search(**'М\d+'**, stack[-1]).group(0)  
 out\_seq += tag + **' БП '** + str(tag\_count) + **' : '** repeat\_count -= 1  
 **if** for\_count > 0 **and** re.match(**r'^for [a-z][a-z\d]+ М\d+ М\d+$'**, stack[-1]):  
 out\_seq += **'1 + := '** tag = re.findall(**'М\d+'**, stack[-1])  
 out\_seq += tag[0] + **' БП '** + tag[1] + **' : '** for\_count -= 1  
 **if** len(stack) > 0 **and** stack[-1] != **'begin'**:  
 stack.pop()  
 **else**:  
 **while** len(stack) > 0 **and** stack[-1] != **'begin'**:  
 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'**)  
f.write(out\_seq)  
f.close()