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 №3

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 Базовые понятия 4](#_Toc97893077)

[3 Правила генерации машинного кода 5](#_Toc97893078)

[4 Результаты экспериментов 8](#_Toc97893079)

[Приложение А Описание правил записи элементов заданного выходного языка 12](#_Toc97893080)

[Приложение Б 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 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.

**4 Результаты экспериментов**

Пример работы программы для программы для тестирования: на рисунке 2 представлен скриншот файла, содержащего текст на входном языке программирования, на рисунке 3 – скриншоты результата работы программы. На рисунках 4 и 5 представлены скриншоты состояния интерфейса до и после выполнения программы.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – Скриншот части файла, содержащего текст на входном языке программирования

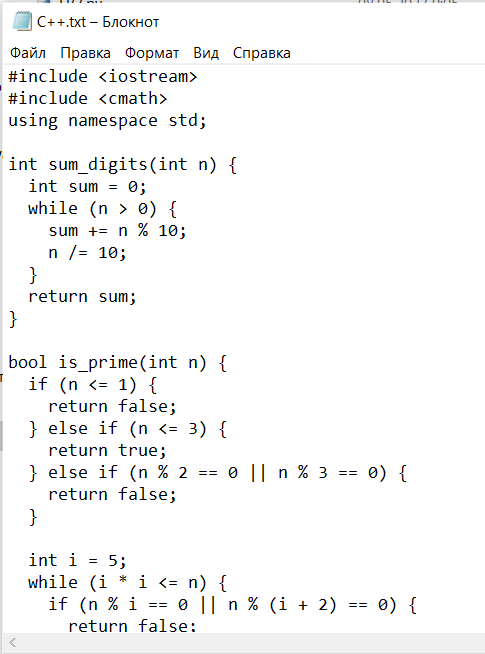


Рисунок 3 – Скриншот части файла, содержащего текст на входном языке программирования



Рисунок 4 – Скриншот начального интерфейса

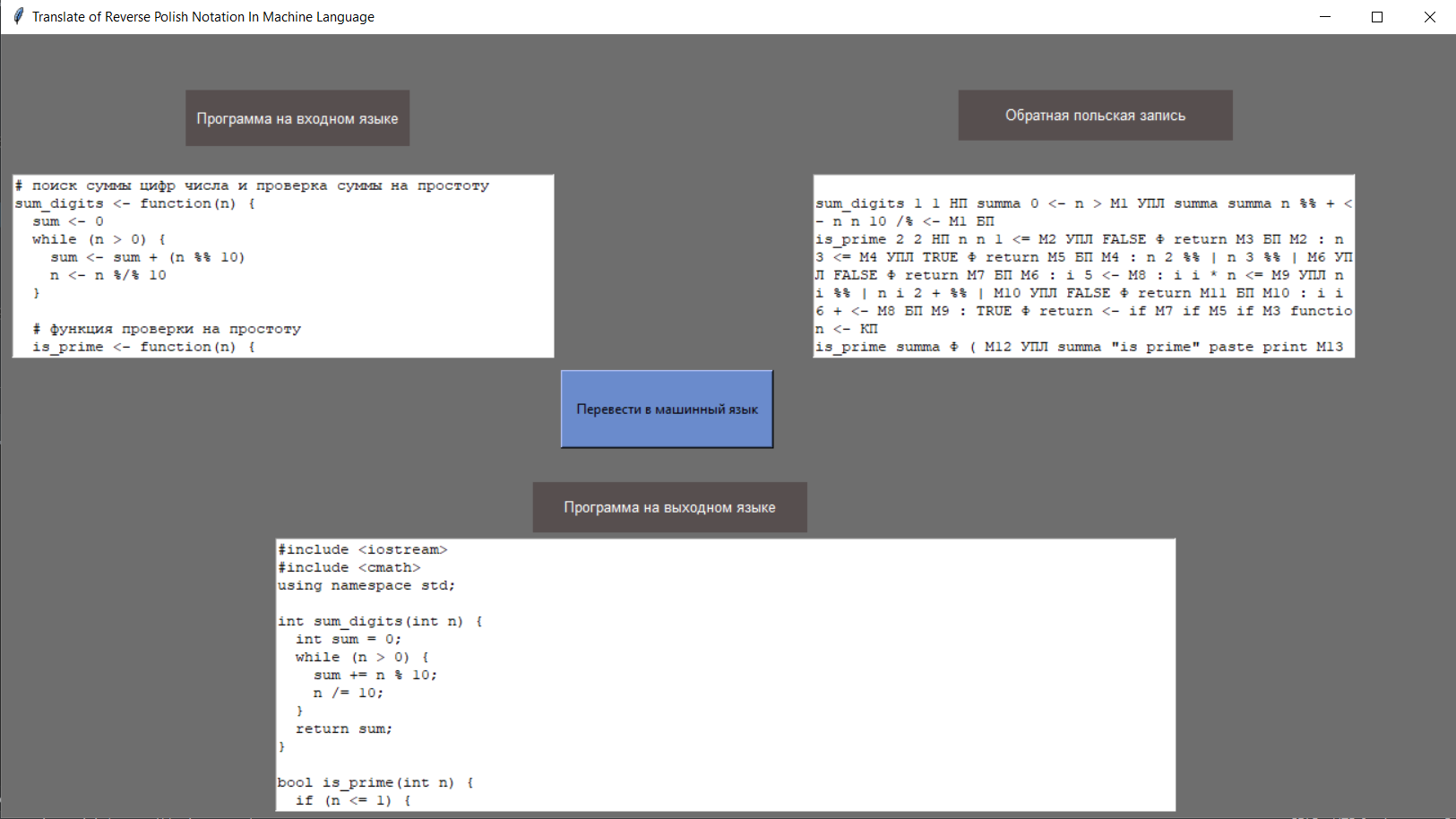


Рисунок 5 – Скриншот интерфейса после выполнения программы

**ПРИЛОЖЕНИЕ А**

**Описание правил записи элементов заданного выходного языка**

***Язык программирования С++ (ISO/IEC 14882)***

**Идентификаторы**

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

**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

.25

25.

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

Последовательность, включающая цифры, десятичную точку (необязательную), символ «е» или «Е», а также знак «+» или «-» вида (необязательный):

1.23е-25

1.23Е-25

1.23е+25

1.23Е+25

1.23е2

1.23Е2

**Символьные (строковые) константы**

Символьная константа – один символ, заключенный в апострофы:

„a‟

Строковая константа - последовательность символов, заключенная в кавычки, расположенная в пределах одной строки, вида:

“acb 12\_& ?tu”

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

Идентификатор, после которого в квадратных скобках перечислены выражения-индексы, вида:

Abc[12][I][i-6] C[1+i]

**Комментарии (строчные и блочные)**

Блочные – последовательность символов, начинающаяся с «/\*» и оканчивающаяся «\*/», возможно содержащая несколько строк:

/\* Это комментарий, Который содержит 2 строки\*/

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

i=i+1; // это инкремент

**Обращения к функциям пользователя**

Идентификатор, после которого в круглых скобках следует последовательность выражений-аргументов, разделенных запятыми. Пустые скобки не допускаются:

F(12,4,i);

F(av-6);

**Арифметические операции**

Сложение +

Вычитание -

Умножение \*

Деление / 89

|  |  |
| --- | --- |
| **Операции сравнения** | |
| Меньше | < |
| Больше | > |
| Равно | == |
| Не равно | != |
| Меньше или равно | <= |
| Больше или равно | >= |

**Оператор присваивания**

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

a=b+с;

b[2][i-9]=12;

**Операторы блока**

{ – начало блока

…

} - конец блока

**Структура программы**

Программа начинается операторами описания данных. Затем могут идти описания данных и функций, а затем основная функция программы int main () и её тело, заключённое в операторы блока, оканчивающееся точкой.

Описания int main()

{

…

return 0;

}

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

Начинается с ключевого слова типа и содержит перечисление идентификаторов через запятую. Оканчивается знаком «;»

<тип> <список элементов>;

Типы переменных: int (целый), float (вещественный), char (символьный)

Элементом списка может быть массив, для которого указывается идентификатор и размерности:

int a,b,c;

float d[3][4], c[78];

**Операторы описания функций**

Функции имеют заголовок вида

<тип> <идентификатор> (<список формальных параметров>);

и тело – список операторов, заключенный в операторы блока

{ … };

Например:

int abc (float r)

{

float r1,r2; y:=sinr(r1)/cos(r2)\*tan(r);

}

В теле функции может присутствовать оператор

return (<значение>);

**Оператор безусловного перехода и метки**

goto <метка> ;

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

a: str=‟ujhti‟;

**Оператор условного перехода**

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

if (логическое выражение) оператор\_1 else

*оператор\_2;*

if (логическое выражение) оператор\_1;

Вместо отдельных операторов могут использоваться блоки операторов:

if (логическое выражение)

{операторы\_1} else

{операторы\_2}

**Приложение Б  
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** ((token **in** inverse\_tokens) **and** re.match(**r'^I\d+$'**, inverse\_tokens[token])) **or** token **in** [**'abs'**, **'cos'**, **'exp'**, **'ln'**, **'read'**, **'readln'**, **'sin'**, **'sqrt'**, **'write'**, **'writeln'**] **or** re.match(**r'^М\d+$'**, token)  
  
**def** is\_constant(token):  
 **return** ((token **in** inverse\_tokens) **and** re.match(**r'^C\d+$'**, inverse\_tokens[token])) **or** ((token **in** inverse\_tokens) **and** re.match(**r'^N\d+$'**, inverse\_tokens[token])) **or** token.isdigit()  
  
**def** is\_operation(token):  
 **return** (token **in** inverse\_tokens) **and** re.match(**r'^O\d+$'**, inverse\_tokens[token])  
  
*# 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)  
 **if** token\_class == **'C'**:  
 **for** k **in** data.keys():  
 data[k] = re.sub(**r"'([^']\*)'"**, **r'"\1"'**, data[k])  
 tokens.update(data)  
  
*# лексемы (значение-код)*inverse\_tokens = {val: key **for** key, val **in** tokens.items()}  
  
replace = { **'scan'**: **'cin'**, **'print'**: **'cout'**, **'integer'**: **'integer'**, **'function'**:**'function'**, **'<-'**: **'='**, **'||'**: **'||'**, **'&&'**: **'&&'**, **'!='**: **'!='**, **'='**: **'='**, **'/'**: **'/'**, **'%%'**: **'%'**, **'!'**: **'!'**}  
*# файл, содержащий обратную польскую запись*f = open(**'reverse\_polish\_entry.txt'**, **'r'**)  
inp\_seq = f.read()  
inp\_seq = re.sub(**r"'([^']\*)'"**, **r'"\1"'**, inp\_seq)  
f.close()  
  
t = re.findall(**r'(?:\"[^\"]\*\")|(?:[^ ]+)'**, inp\_seq)  
  
i = 0  
stack = []  
out\_seq = **''**is\_func = **False**variable = {}  
**while** i < len(t):  
 **if** is\_func == **True and not**(is\_identifier(t[i])):  
 out\_seq += **'() {\n'** is\_func = **False  
 if** is\_identifier(t[i]) **or** is\_constant(t[i]):  
 stack.append(replace[t[i]] **if** t[i] **in** replace **else** t[i])  
 **elif** t[i] == **'НП'**:  
 arg1 = int(stack.pop())  
 stack.pop()  
 *#arg2 = stack.pop()* **if** arg1 == 1:  
 out\_seq += **f'void main'  
 else**:  
 out\_seq += **f'void {**arg2**}'** is\_func = **True  
 elif** t[i] == **'КП'**:  
 out\_seq += **'}'  
 elif** t[i] **in** [**'integer'**]:  
 k = int(stack.pop())  
 a = []  
 **while** k != 0:  
 a.append(stack.pop())  
 k -= 1  
 a.reverse()  
 out\_seq += replace[t[i]] + **' '** + **', '**.join(a) + **';\n'  
 for** j **in** a:  
 variable[j] = replace[t[i]]  
 **elif** t[i] == **'КО'**:  
 stack.pop()  
 stack.pop()  
 **elif** t[i] == **'УПЛ'**:  
 arg1 = stack.pop()  
 arg2 = stack.pop()  
 out\_seq += **f'if (!({**arg2**})) goto {**arg1**};\n'  
 elif** t[i] == **'БП'**:  
 arg1 = stack.pop()  
 out\_seq += **f'goto {**arg1**};\n'  
 elif** t[i] == **':'**:  
 arg1 = stack.pop()  
 out\_seq += **f'{**arg1**}: '  
 elif** is\_operation(t[i]):  
 **if** t[i] == **':='**:  
 arg1 = stack.pop()  
 arg2 = stack.pop()  
 out\_seq += **f'{**arg2**} = {**arg1**};\n'  
 else**:  
 operation = replace[t[i]] **if** t[i] **in** replace **else** t[i]  
 arg1 = stack.pop()  
 *# if t[i] != 'not':  
 # #arg2 = stack.pop()  
 # #stack.append(f'({arg2} {operation} {arg1})')  
 # else:* stack.append(**f'({**operation**}{**arg1**})'**)  
 **elif** t[i] == **'АЭМ'**:  
 k = int(stack.pop())  
 a = []  
 **while** k != 0:  
 a.append(stack.pop())  
 k -= 1  
 a.reverse()  
 out\_seq += a[0] + **'['** + **']['**.join(a[1:]) + **']'  
 elif** t[i] == **'Ф'**:  
 **try**:  
 k = int(stack.pop()) + 1  
 **except** ValueError:  
 k = 0  
 a = []  
 **while** k != 0:  
 a.append(stack.pop())  
 k -= 1  
 a.reverse()  
 **if** len(a) > 0 **and** a[0] == **'scan'**:  
 b = []  
 **for** j **in** a[1:]:  
 **if** variable[j] == **'int'**:  
 b.append(**'%d'**)  
 **elif** variable[j] == **'double'**:  
 b.append(**'%f'**)  
 out\_seq += a[0] + **'("'** + **' '**.join(b) + **'", '** + **', '**.join(map(**lambda** x: **'&'** + x, a[1:])) + **');\n'  
 elif** len(out\_seq)>0 **and** len(a)>0:  
 out\_seq += a[0] + **'('** + **', '**.join(a[1:]) + **');\n'** i += 1  
  
out\_seq = re.sub(**r'(М\d+): if \(!\((.\*)\)\) goto (М\d+);(?:\n|\n((?:.|\n)+)\n)goto \1;\n\3: '**, **r'while \2 {\n\4\n}\n'**, out\_seq)  
out\_seq = re.sub(**r'if \(!\((.\*)\)\) goto (М\d+);(?:\n|\n((?:.|\n)+)\n)goto (М\d+);\n\2: ((?:\n|.)+)\n?\4: '**, **r'if \1 {\n\3\n} else {\n\5\n}\n'**, out\_seq)  
out\_seq = re.sub(**r'if \(!\((.\*)\)\) goto (М\d+);(?:\n|\n((?:.|\n)+)\n)\2: '**, **r'if \1 {\n\3\n}\n'**, out\_seq)  
  
c = 0  
a = out\_seq.split(**'\n'**)  
**for** i **in** range(len(a)):  
 **if** len(a[i]) == 0:  
 **continue  
 if** a[i][0] == **'}'**:  
 c -= 1  
 a[i] = 4\*c\***' '** + a[i]  
 **if** a[i][len(a[i]) - 1] == **'{'**:  
 c += 1  
a = [i **for** i **in** a **if** len(i.strip()) > 0]  
out\_seq = **'\n'**.join(a)  
out\_seq = **'#include <stdio.h>\n\n'** + out\_seq  
**while** re.search(**r'= \(([^\)]+)\);\n'**, out\_seq):  
 out\_seq = re.sub(**r'= \(([^\)]+)\);\n'**, **r'= \1;\n'**, out\_seq)  
  
*# файл, содержащий текст на выходном языке программирования*f = open(**'c++.txt'**, **'w'**)  
f.write(out\_seq)  
f.close()  
  
*# действие после нажатия на кнопку***def** ml():  
 label4 = tkinter.Label(window, text=**'Программа на выходном языке'**, font=(**"Arial"**, 10), foreground=**"white"**,background=**"#574f4f"**)  
 label4.place(x=475, y=400, width=245, height=45)  
 ml\_text = open(**'С++.txt'**,encoding=**'UTF-8'**).readlines()  
 ml\_text = **''**.join(ml\_text)  
 textline1 = Text(window, height=15, width=100)  
 textline1.insert(1.0, ml\_text)  
 textline1.place(x=245, y=450)  
  
  
*# создание окна интерфейса*window = tkinter.Tk()  
window.geometry(**'1300x700'**)  
window.title(**"Translate of Reverse Polish Notation In Machine Language"**)  
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=**'Обратная польская запись'**, font=(**"Arial"**, 10), foreground=**"white"**,  
 background=**"#574f4f"**)  
label3.place(x=855, y=50, width=245, height=45)  
  
token\_text = open(**'reverse\_polish\_entry.txt'**, encoding=**'ANSI'**).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 машинный язык'**, bg=**'#6a8bcc'**,command=ml)  
button.place(x=500, y=300, width=190, height=70)  
  
window.mainloop()