МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

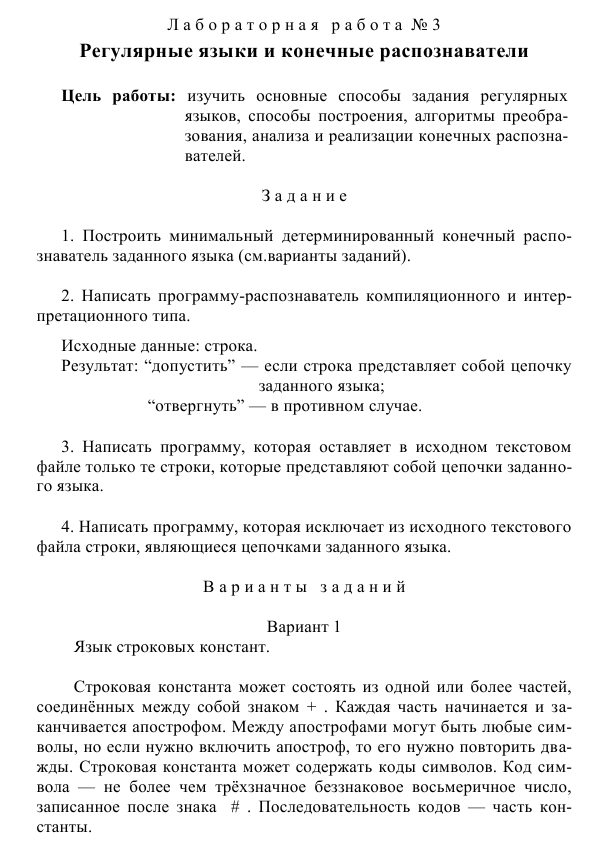
**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №3

по дисциплине «Регулярные языки и конечные распознователи»

Белгород 2020 г.



Пункт 1

Изначально был построен следующий распознаватель

A=<S,X,s0,o,F>

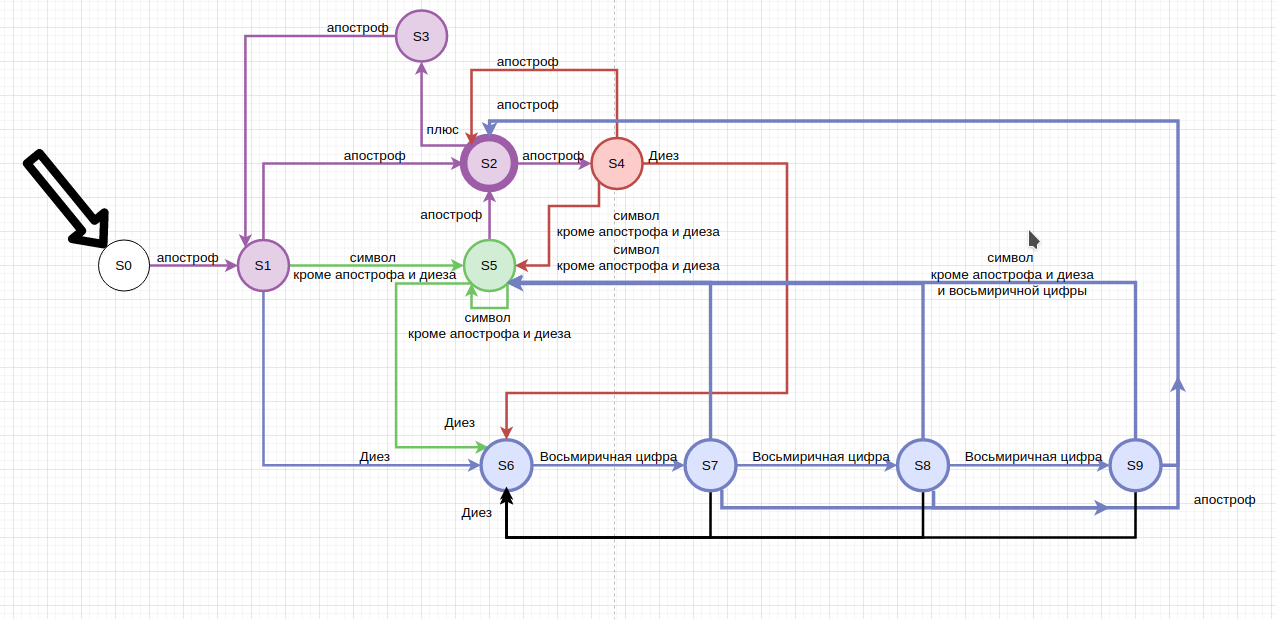
S={S0,S1,S2,S3,S4,S5,S6,S7,S8,S9}

X = {‘,+,#, ,} + другие символы которые есть в строках

s0=S0

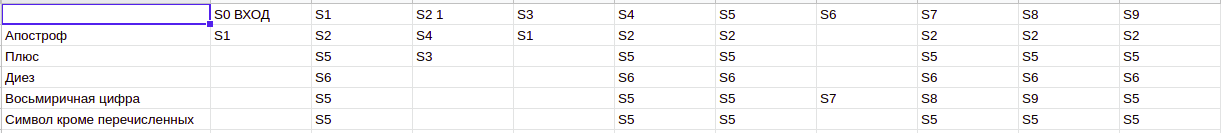
F={S2}

1.S->’SASDa

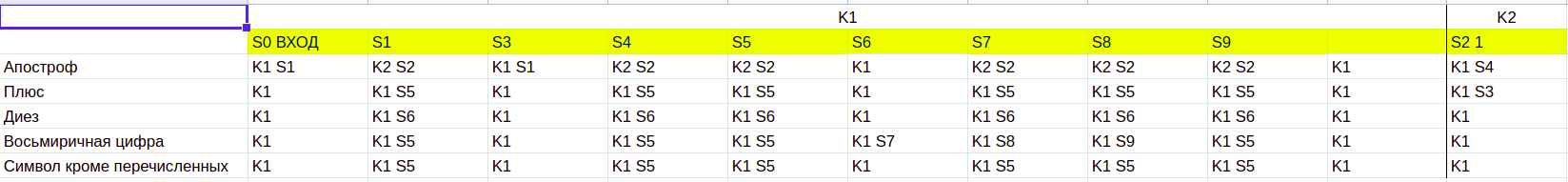


В ходе процесса минимамизации

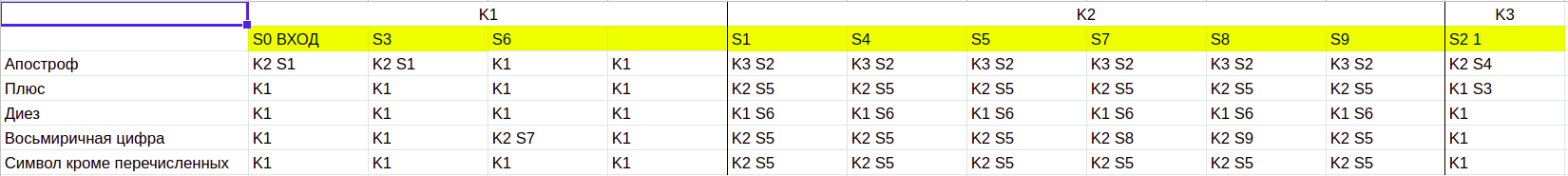
изначальная таблица распознователя



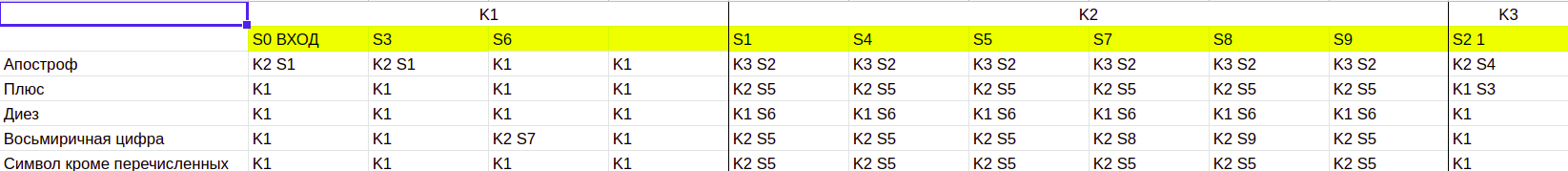
классы 0-эквивалентных состояний



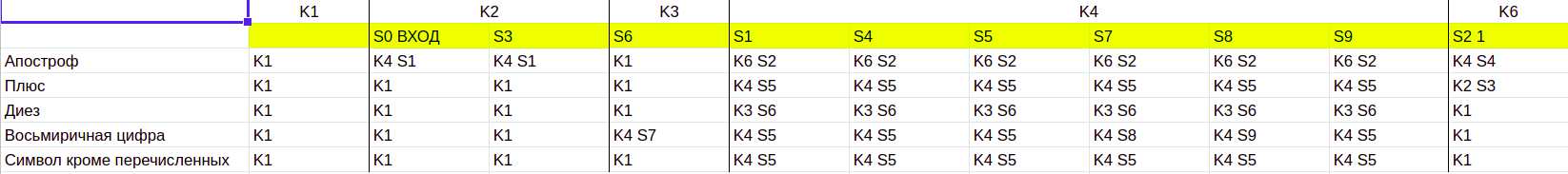
классы 1-эквивалентных состояний



классы 2-эквивалентных состояний



классы 3-эквивалентных состояний



из таблицы видно что классы 4-эквивалентных состояний будут такие же как 3

В ходе процесса минимизации получился

A=<S,X,s0,o,F>

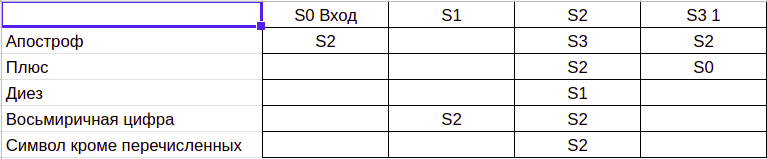
S={S0,S1,S2,S3}

X = {‘,+,#, ,} + другие символы которые есть в строках

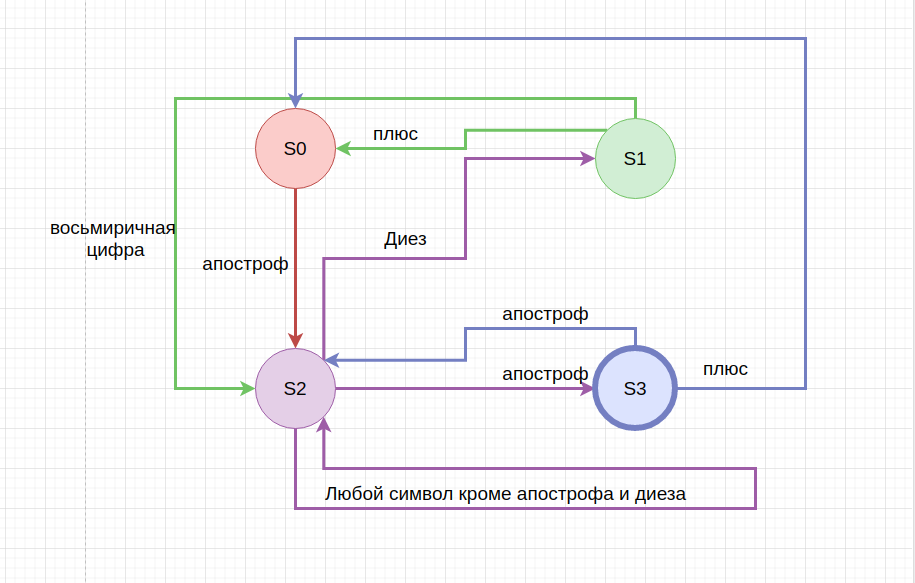
s0=S0

F={S3}

Окончательная таблица переходов минимального распознавателя



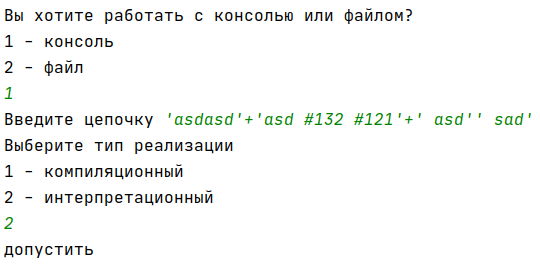
В виде графа

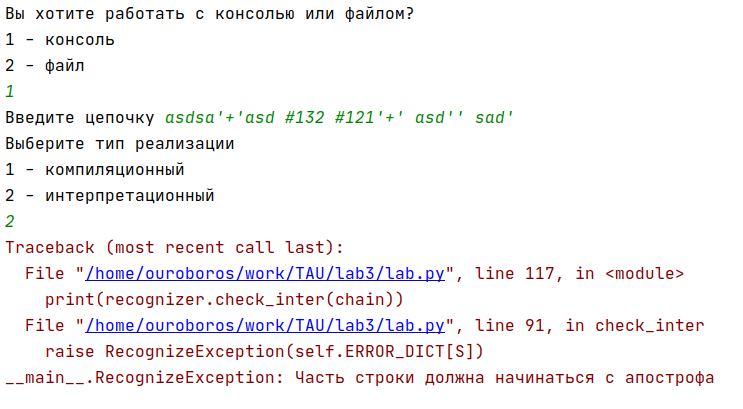


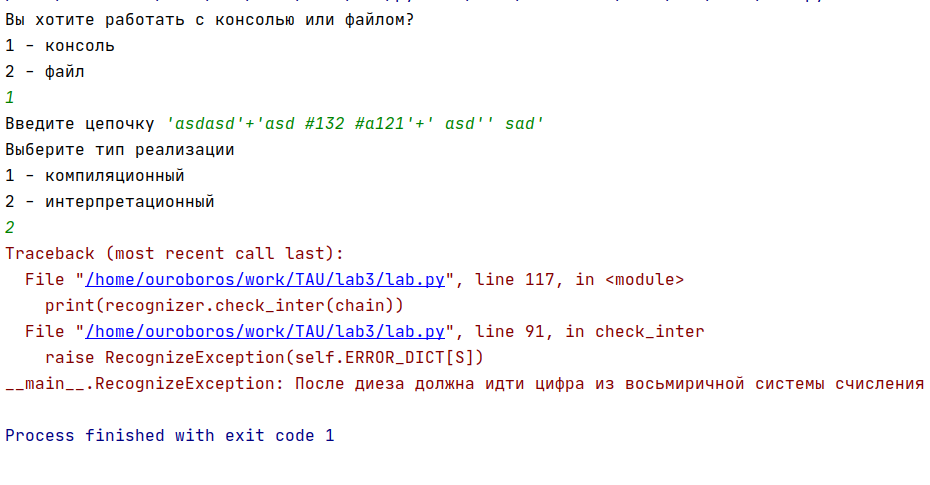
Результаты работы программы:

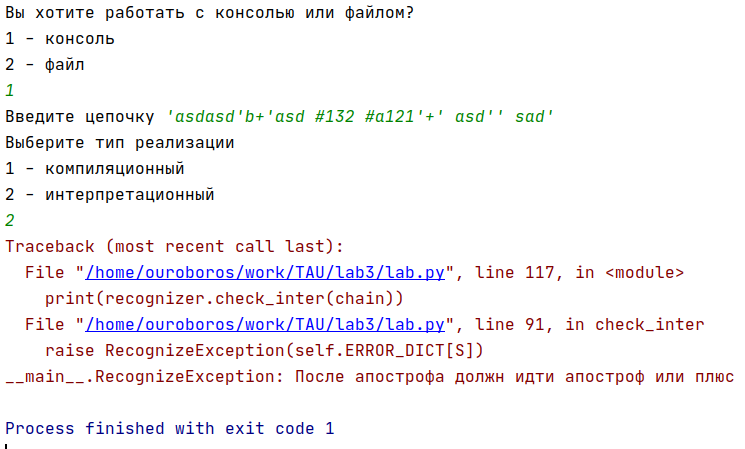
Задание 2

интерпретационного типа

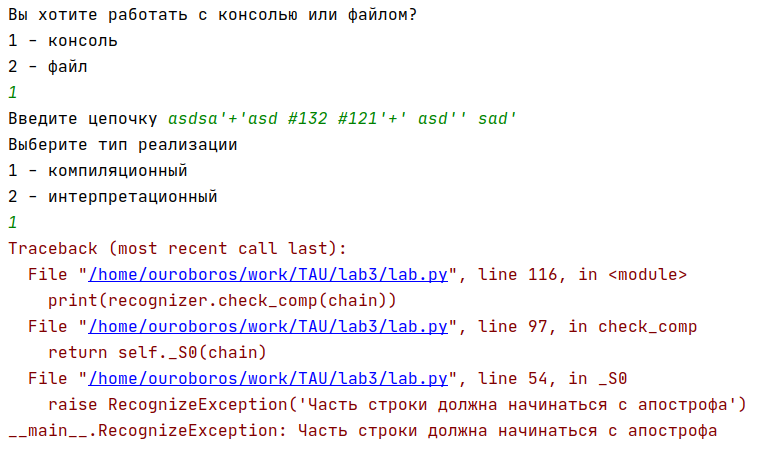
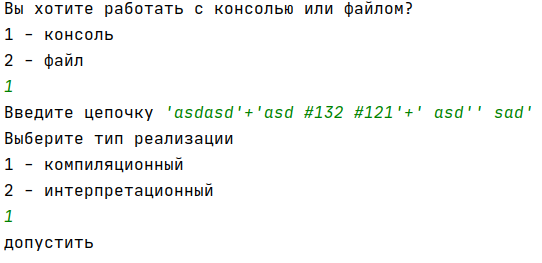


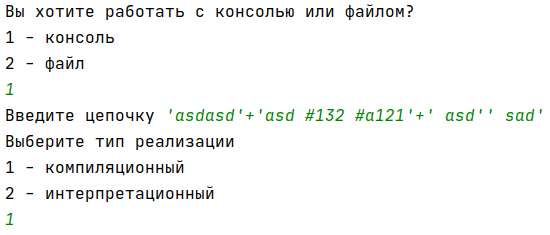


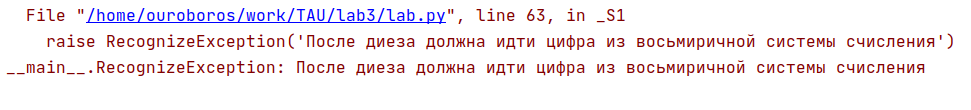




компиляционного типа





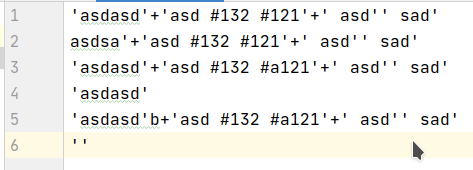




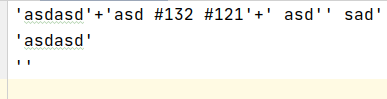
Задание 3



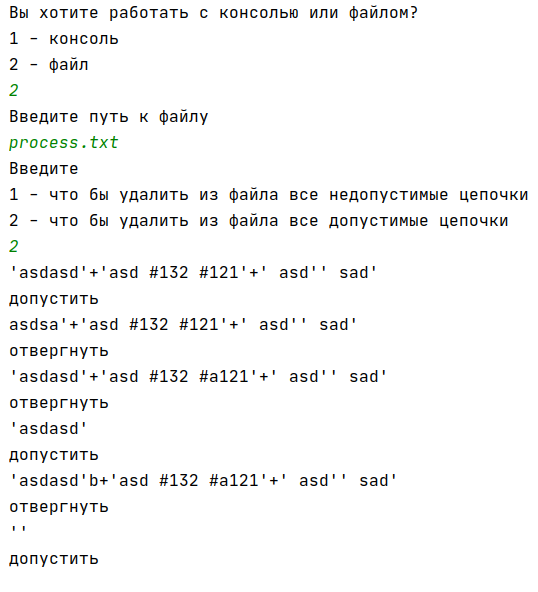
было



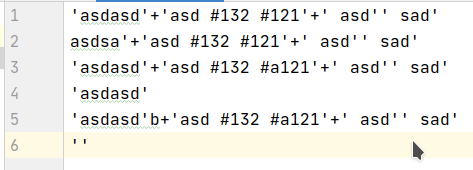
стало



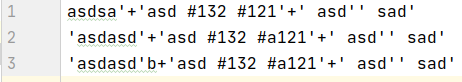
Задание 4



было



стало



Для 2го, 3го и 4го задания был написан следующий код

**class** RecognizeException(Exception):

**def** \_\_init\_\_(self, message):

self.message = message

**def** \_\_str\_\_(self):

**return** str(self.message)

**class** Recognizer:

YES = **'допустить'**

NO = **'отвергнуть'**

**def** check\_inter(self, chain: str):

S = 0

**for** ch **in** chain:

**if** S < 0:

**raise** RecognizeException(self.ERROR\_DICT[S])

S = self.TABLE[self.L[ch]][S]

print(S)

**return** self.YES **if** S==3 **else** self.NO

**def** check\_comp(self, chain: str):

**return** self.\_S0(chain)

**def** \_\_init\_\_(self):

self.TABLE = [[2, -2, 3, 2], *# Апостроф*

[-1, -2, 2, 0], *# Плюс*

[-1, -2, 1, -3]] *# Диез*

self.TABLE += [[-1, 2, 2, -3] **for** i **in** range(0, 8)] *# Восьмиричные числа*

self.TABLE += [[-1, -2, 2, -3] **for** ch **in** self.ELSE] *# Все остальные символы*

self.L = {

**"'"**:0,

**"+"**:1,

**"#"**:2

}

index = 3

**for** ch **in** range(0, 8):

self.L.update({str(ch): index})

index += 1

**for** ch **in** self.ELSE:

self.L.update({str(ch): index})

index += 1

ERROR\_DICT = {

-1: **'Часть строки должна начинаться с апострофа'**,

-2: **'После диеза должна идти цифра из восьмиричной системы счисления'**,

-3: **'После апострофа должн идти апостроф или плюс'**

}

APOSTROPH = **"'"**

PLUS = **'+'**

DIEZ = **'#'**

EIGTH\_DIGIT = **'01234567'**

ELSE = **'йцукенгшщзхъэждлорпавыфячсмитьбю.,-!"№;%:?\*()\_='** \

**'qwertyuiop[];lkjhgf dsa'** *#Множество всех входных символов за исключением тех что выше*

**def** \_S0(self, chain: str):

**if not** chain:

**return** self.NO

ch = chain[0]

**if** ch **in** self.APOSTROPH:

**return** self.\_S2(chain[1:])

**else**:

**raise** RecognizeException(**'Часть строки должна начинаться с апострофа'**)

**def** \_S1(self, chain: str):

**if not** chain:

**return** self.NO

ch = chain[0]

**if** ch **in** self.EIGTH\_DIGIT:

**return** self.\_S2(chain[1:])

**else**:

**raise** RecognizeException(**'После диеза должна идти цифра из восьмиричной системы счисления'**)

**def** \_S2(self, chain: str):

**if not** chain:

**return** self.NO

ch = chain[0]

**if** ch **in** self.APOSTROPH:

**return** self.\_S3(chain[1:])

**elif** ch **in** self.DIEZ:

**return** self.\_S1(chain[1:])

**else**:

**return** self.\_S2(chain[1:])

**def** \_S3(self, chain: str):

**if not** chain:

**return** self.YES

ch = chain[0]

**if** ch **in** self.APOSTROPH:

**return** self.\_S2(chain[1:])

**elif** ch **in** self.PLUS:

**return** self.\_S0(chain[1:])

**else**:

**raise** RecognizeException(**'После апострофа должн идти апостроф или плюс'**)

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

recognizer = Recognizer()

print(**'Вы хотите работать с консолью или файлом?'**)

print(**'1 - консоль'**)

print(**'2 - файл'**)

flag = int(input())

**if** flag == 1:

print(**'Введите цепочку'**, end=**' '**)

chain = input()

print(**'Выберите тип реализации'**)

print(**'1 - компиляционный'**)

print(**'2 - интерпретационный'**)

flag = int(input())

**if** flag == 1:

print(recognizer.check\_comp(chain))

**elif** flag == 2:

print(recognizer.check\_inter(chain))

**elif** flag == 2:

print(**'Введите путь к файлу'**)

path = input()

print(**'Введите '**)

print(**'1 - что бы удалить из файла все недопустимые цепочки'**)

print(**'2 - что бы удалить из файла все допустимые цепочки'**)

flag = int(input())

**with** open(path) **as** f:

all\_chain = f.read().split(**'\n'**)

delete\_chain = []

**if** flag == 1:

**for** chain **in** all\_chain:

print(chain)

**try**:

print(recognizer.check\_comp(chain))

**except**:

print(**'отвергнуть'**)

delete\_chain.append(chain)

**if** flag == 2:

**for** chain **in** all\_chain:

print(chain)

**try**:

print(recognizer.check\_comp(chain))

**except**:

print(**'отвергнуть'**)

**continue**

delete\_chain.append(chain)

**with** open(path, **'w'**) **as** f:

**for** chain **in** all\_chain:

**if** chain **not in** delete\_chain:

f.write(chain+**'\n'**)