Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Институт информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника" профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

**Курсовая работа по дисциплине**

**Теория языков программирования и методы трансляции**

Вариант 7

Выполнил:

Студент гр. ИП-013

« » 2023 г.

Проверил:

/Копытина Т.А./

ФИО студента

Старший преподаватель кафедры ПМиК \_\_\_\_\_\_\_\_\_\_/Павлова У. В./

ФИО преподавателя

« » 2023 г. Оценка

Новосибирск, 2023 г

**Оглавление**

[Постановка задачи 3](#_Toc153742820)

[Описание алгоритма 4](#_Toc153742821)

[Описание основных блоков программы 5](#_Toc153742822)

[Результаты работы программы 6](#_Toc153742823)

[Текст программы 9](#_Toc153742824)

# **Постановка задачи**

Тема 1 Построение конструкций, задающих язык.

Вариант №7: Написать программу, которая по предложенному описанию языка построит регулярное выражение, задающее этот язык, и сгенерирует с его помощью все цепочки языка в заданном диапазоне длин. Предусмотреть также возможность генерации цепочек по введённому пользователем РВ (в рамках варианта). Варианты задания языка: Алфавит, кратность длины и заданная фиксированная подцепочка всех цепочек языка.

# **Описание алгоритма**

Необходимо написать программу, которая построит по заданному языку, регулярное выражение. Затем по полученному РВ сгенерирует цепочки с заданными подцепочками.

Предусмотрена возможность сохранения сгенерированных цепочек с помощь кнопки в меню «Вывод в файл».

В форме расположены пункты «Тема курсовой», «Автор работы», в которых представлена информация о задании курсовой работы, также сведения об авторе работы.

Для того, чтобы построить регулярное выражение (РВ), программа берет входные данные: «длина цепочки», «кратность цепочки», «подцепочка». Затем определяется длина подцепочки sub\_len и количество символов, которое необходимо добавить до достижения кратности to\_krat. Создается пустая строка add, которая будет использована для добавления символов до кратности. Если необходимо добавить некоторое количество символов, то создается список строк str\_list. Затем к add добавляется строка, представляющая алфавит в скобках. Затем формируется двумерный список str\_list, в котором на главной диагонали находится подцепочка s, а в остальных местах добавленные символы.

Далее идет построение самих цепочек по полученному РВ. В этот процесс включается проверка построенных цепочек на наличие обязательной подцепочки и кратности длины самой цепочки.

# **Описание основных блоков программы**

**Основные функции программы:**

1. def build\_core(s) – Эта функция используется для построения "ядра" регулярного выражения. Она объединяет элементы входного списка s с использованием разделителя separator и добавляет скобки;
2. def main(alphabet, kratnost, subchain) – функция создает регулярное выражение на основе входных параметров;
3. def SeparatedByPlus(chain) – функция разбивает входную цепочку chain по символу separator и выдает список подцепочек;
4. def SeparatedByBrackets(chain) – функция разбивает входную цепочку chain по скобкам и возвращает список подцепочек;
5. def BuildBricks(s, max) – функция строит блоки для регулярного выражения, вызывая другие функции;
6. def ConcatenateBlocks(Mas, max, index, s, set\_) – функция соединяет блоки, вызывая рекурсивно себя;
7. def MergeBlocks(s, max, inf) – функция объединяет подцепочки и вызывает другие функции для построения регулярного выражения;
8. def generator(merge\_list, max, s, set\_) – Функция генерирует цепочки на основе списка подцепочек;

# **Результаты работы программы**

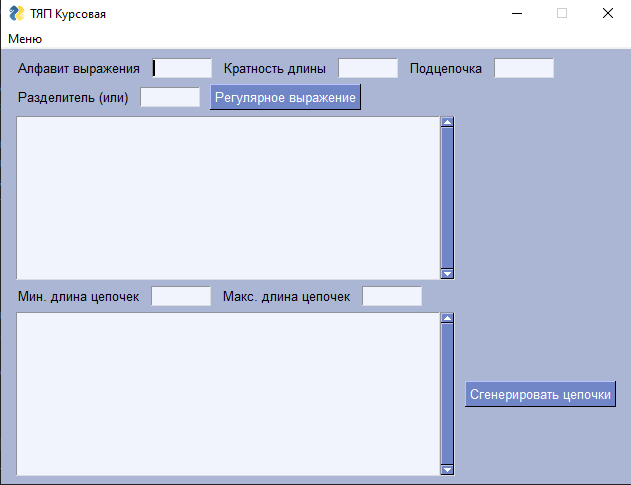


Рис 1. Начальный интерфейс

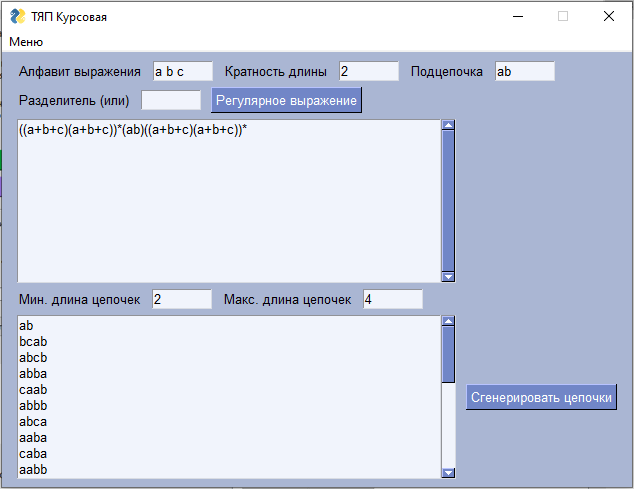
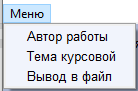


Рис 2. Заполнение данных



Ри 3. Отображение пункта «Меню»

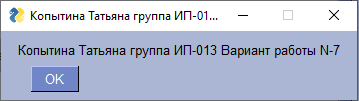


Рис 4. Отображение информации об авторе

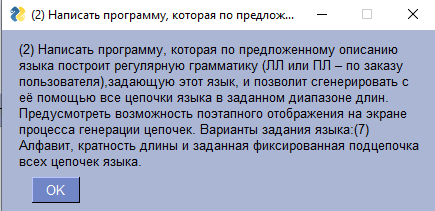


Рис 5. Отображение информации о задании на курсовую работу

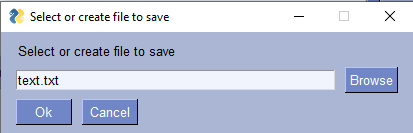


Рис 6. Заполнение формы сохранения выводимых цепочек в файл

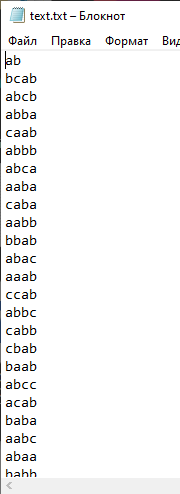


Рис 7. Отображение выходных данных в файле.

# **Текст программы**

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import PySimpleGUI as sg

def build\_core(s):

global separator

out\_str = ''

for i in s:

out\_str += f"{''.join(k for k in i)}"

out\_str += separator

out\_str\_out = f"({out\_str[:-1]})"

return out\_str\_out

#создает регулярное выражение на основе входных параметров

def main(alphabet, kratnost, subchain):

global separator

a = alphabet

k = kratnost

s = subchain

sub\_len = len(s) # длина подцепочки

to\_krat = k - (sub\_len % k) # сколько не хватает до кратности

add = "" # добавлеяем до кратности

if k != sub\_len and k != 1:

to\_krat += 1

str\_list = [[] for i in range(to\_krat)]

add += f"({separator.join(i for i in a)})"

for i in range(to\_krat):

for j in range(to\_krat):

if j == i:

str\_list[i].append(s)

else:

str\_list[i].append(add)

print(str\_list)

core = build\_core(str\_list)

else:

core = f"({s})"

additional\_part = "("

for i in range(k):

additional\_part += f"({separator.join(i for i in a)})"

additional\_part += ")\*"

res = additional\_part

res += core

res += additional\_part

return res

#main(['0','1','a'], 3, 'a' )

# разбивает входную цепочку chain по символу separator и выдает список подцепочек

def SeparatedByPlus(chain):

global separator

scobki = 0

SpPlus = []

str = ""

for i in range(len(chain)):

if chain[i] == '(':

scobki += 1

elif chain[i] == ')':

scobki -= 1

if chain[i] != separator or scobki != 0:

str += chain[i]

elif scobki == 0:

SpPlus.append(str)

str = ""

SpPlus.append(str)

print("SpPlus", SpPlus)

return SpPlus

# разбивает входную цепочку chain по скобкам и возвращает список подцепочек

def SeparatedByBrackets(chain):

scobki = 0

Split = []

str = ""

j = 0

for i in range(len(chain)):

if j >= len(chain):

break

if chain[j] == '(':

scobki += 1

elif chain[j] == ')':

scobki -= 1

if len(chain) > (j + 1) and chain[j + 1] == '\*':

str += chain[j]

j += 1

str += chain[j]

if scobki == 0:

Split.append(str)

str = ""

j += 1

print("SplitBrackets", Split)

return Split

#строит блоки для регулярного выражения, вызывая другие функции.

def BuildBricks(s, max):

SpScob = SeparatedByBrackets(s)

Mas = []

newinf = False

for i in range(len(SpScob)):

if SpScob[i][0] == '(':

tmp = ""

if SpScob[i][-1] == '\*':

tmp = SpScob[i][1:-2]

#print("TMP1", tmp)

newinf = True

elif SpScob[i][-1] == ')':

tmp = SpScob[i][1:-1]

#print("TMP2", tmp)

merge = MergeBlocks(tmp, max, newinf)

Mas.append(merge)

else:

str = [SpScob[i]]

Mas.append(str)

set\_ = set()

ConcatenateBlocks(Mas, max, 0, "", set\_)

NewBrick = list(set\_)

#print("BuildBricks", NewBrick)

return NewBrick

#соединяет блоки, вызывая рекурсивно себя.

def ConcatenateBlocks(Mas, max, index, s, set\_):

if index == len(Mas):

set\_.add(s)

return

for i in range(len(Mas[index])):

newS = s + Mas[index][i]

# print("ConcatenateBlocks", newS)

if len(newS) > max:

continue

ConcatenateBlocks(Mas, max, index + 1, newS, set\_)

# объединяет подцепочки и вызывает другие функции для построения рв

def MergeBlocks(s, max, inf):

t = 0

MergeList = []

SpPlus = SeparatedByPlus(s)

for i in range(len(SpPlus)):

buildB = BuildBricks(SpPlus[i], max)

print("MergeList: ", MergeList)

print("buildB: ", buildB)

for j in range(len(buildB)):

t = 0

for k in range(len(MergeList)):

print('k: ', k)

print('t: ', t)

#if t > len(MergeList) or t < 0:

#break

if MergeList[t] == buildB[j]:

MergeList.pop(t)

t -= 1

t += 1

for j in range(len(buildB)):

MergeList.append(buildB[j])

if inf:

set\_ = set()

generator(MergeList, max, "", set\_)

MergeList.clear()

MergeList.extend(set\_)

MergeList.append("")

print("INF MergeList: ", MergeList)

#print("MergeBlocks", MergeList)

return MergeList

# генерирует цепочки на основе списка подцепочек

def generator(merge\_list, max, s, set\_):

for i in range(len(merge\_list)):

new\_s = s + merge\_list[i]

set\_\_list = list(set\_)

if len(new\_s) > max or new\_s in set\_\_list:

continue

set\_.add(new\_s)

generator(merge\_list, max, new\_s, set\_)

separator = '+'

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

#main(['0', '1', 'a'], 1, '0')

sg.theme('LightBlue2') # Add a touch of color

# All the stuff inside your window.

menu\_def = [['Меню', ['Автор работы', 'Тема курсовой', 'Вывод в файл']], ]

layout = [[sg.Menubar(menu\_def)],

[sg.Text('Алфавит выражения'), sg.Input(key="-ALPHABET-", size=(8, 10)) , sg.Text('Кратность длины'), sg.Input(key="-KRATNOST-", size=(8, 10)),sg.Text('Подцепочка'), sg.Input(key="-SUBCHAIN-", size=(8, 10))],

[sg.Text('Разделитель (или)'), sg.Input(key="-SEPARATE-", size=(8, 10)) ,sg.Button('Регулярное выражение')],

[sg.Multiline(key="-REGEX-", size=(60, 10))],

[sg.Text('Мин. длина цепочек'),sg.Input(key="-LEFTBORDER-", size=(8, 10)),

sg.Text('Макс. длина цепочек'), sg.Input(key="-RIGHTBORDER-", size=(8, 10))],

[sg.Multiline(key="-CHAINS-", size=(60, 10)),sg.Button('Сгенерировать цепочки')]]

# Create the Window

window = sg.Window('ТЯП Курсовая', layout)

# Event Loop to process "events" and get the "values" of the inputs

while True:

event, values = window.read()

if event != sg.WIN\_CLOSED:

if values[0] == 'Вывод в файл':

res = sg.popup\_get\_file("Select or create file to save")

if res:

try:

file = open(res, 'r+')

except FileNotFoundError:

file = open(res, 'w+')

file.writelines(str(values["-CHAINS-"]))

file.close()

if values[0] == 'Тема курсовой':

sg.popup\_ok("(2) Написать программу, которая по предложенному описанию языка построит регулярную грамматику (ЛЛ или ПЛ – по заказу пользователя),задающую этот язык, и позволит сгенерировать с её помощью все цепочки языка в заданном диапазоне длин. Предусмотреть возможность поэтапного отображения на экране процесса генерации цепочек. Варианты задания языка:(7) Алфавит, кратность длины и заданная фиксированная подцепочка всех цепочек языка.")

if values[0] == 'Автор работы':

sg.popup\_ok('Копытина Татьяна группа ИП-013 Вариант работы N-7')

if event == 'Регулярное выражение':

if values["-ALPHABET-"] == '':

sg.popup\_error("Алфавит не введен")

elif values["-KRATNOST-"] == '':

sg.popup\_error("Кратность цепочки не введена")

elif values["-SUBCHAIN-"] == '':

sg.popup\_error("Обязательная подцепочка не введена")

else:

if values["-SEPARATE-"] != '':

separator = str(values["-SEPARATE-"])

alphabet = str(values["-ALPHABET-"]).split(' ')

kratnost = int(values["-KRATNOST-"])

subchain = str(values["-SUBCHAIN-"])

check = True

for el in subchain:

if el not in alphabet:

check = False

if check:

regex = main(alphabet, kratnost, subchain)

window['-REGEX-'].update(regex)

else:

sg.popup\_error("В подцепочке есть символы не из алфавита")

if event == 'Сгенерировать цепочки':

if values["-REGEX-"] == '':

sg.popup\_error("Регулярное выражение не сформировано!")

else:

res\_str = ''

res\_list = []

all\_list = MergeBlocks(values['-REGEX-'], int(values["-RIGHTBORDER-"]), False)

for i in all\_list:

if int(values["-KRATNOST-"]) == 0:

window["-CHAINS-"].update('lambda')

break

if len(i) >= int(values["-LEFTBORDER-"]) and len(i) % int(values["-KRATNOST-"]) == 0 and str(values["-SUBCHAIN-"]) in i:

res\_list.append(i)

res\_list.sort(key=lambda x: len(x))

if len(res\_list) == 0:

window["-CHAINS-"].update('lambda')

else:

for i in res\_list:

res\_str += i + '\n'

window["-CHAINS-"].update(res\_str)

if event == sg.WIN\_CLOSED: # if user closes window or clicks cancel

break

window.close()