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 /Копытина Т.А./
«____»____2023 г. ФИО студента

Проверил:

Старший преподаватель кафедры ПМиК _____/Павлова У. В./
ФИО преподавателя

«____»____2023 г. Оценка

Новосибирск, 2023 г

Оглавление

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

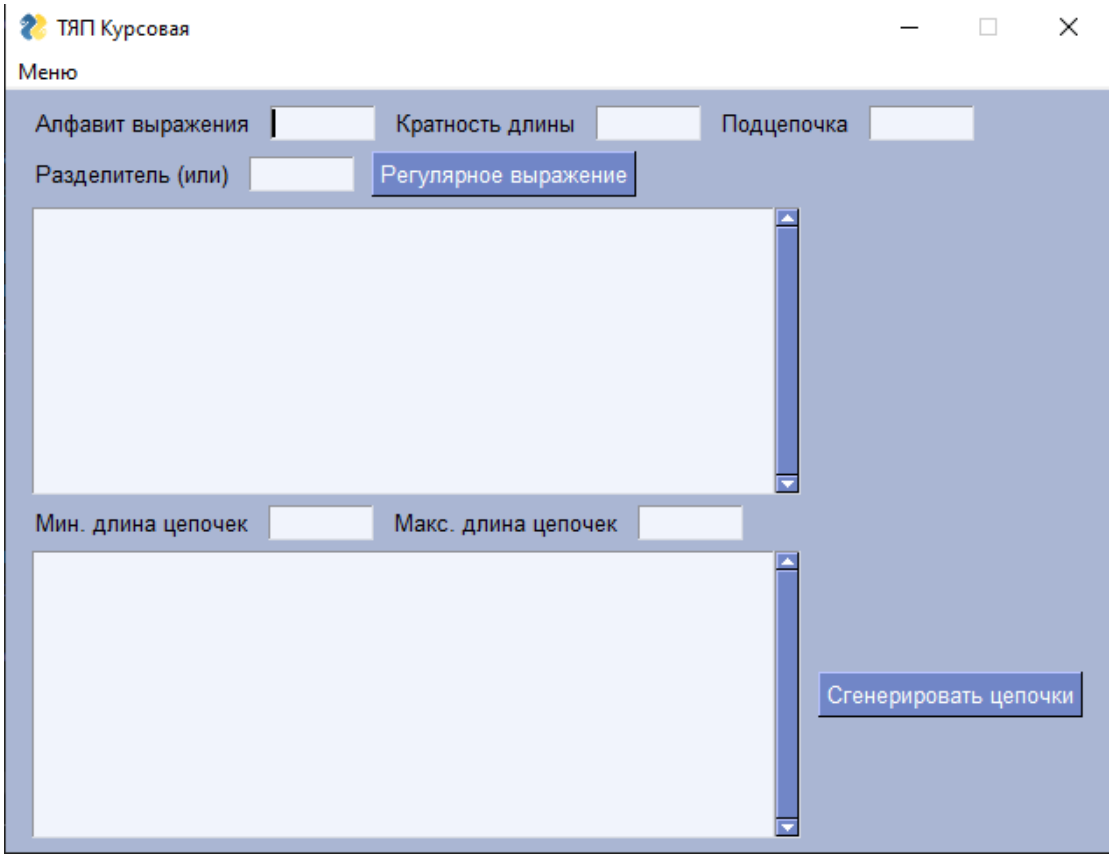


Рис 1. Начальный интерфейс

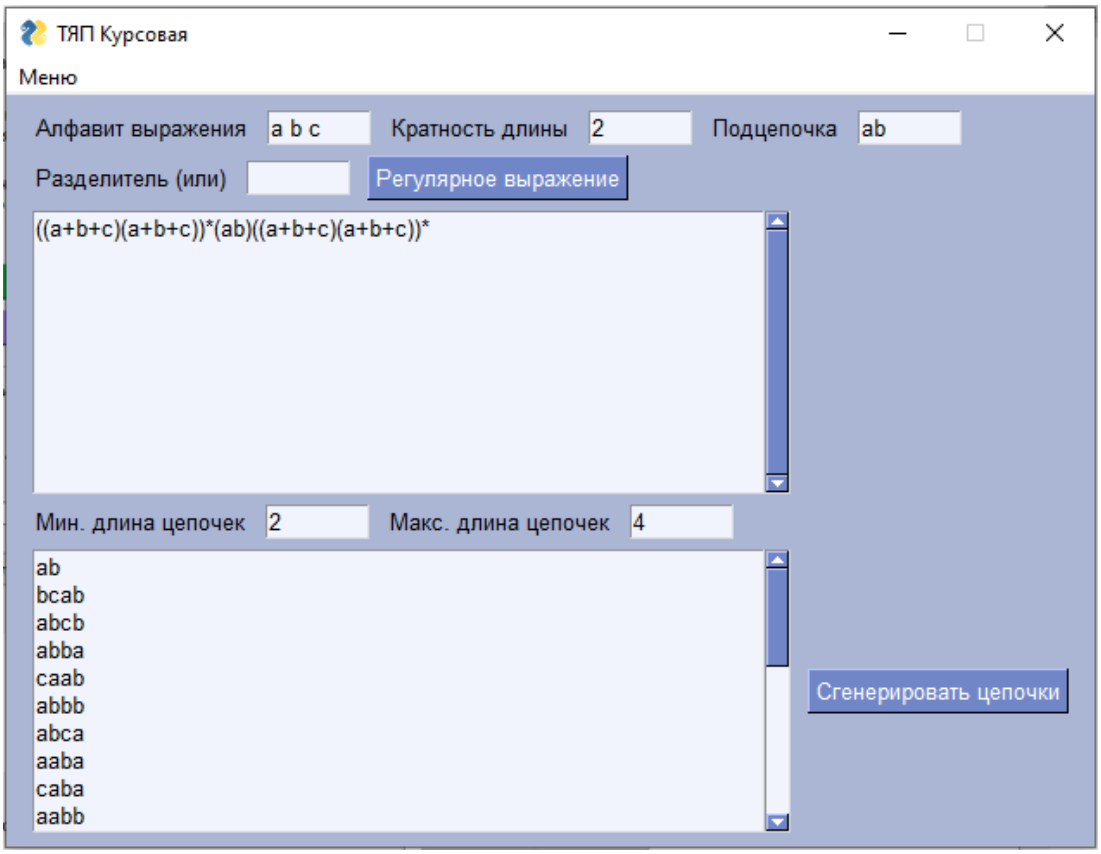
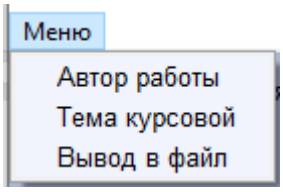


Рис 2. Заполнение данных



Ри 3. Отображение пункта «Меню»

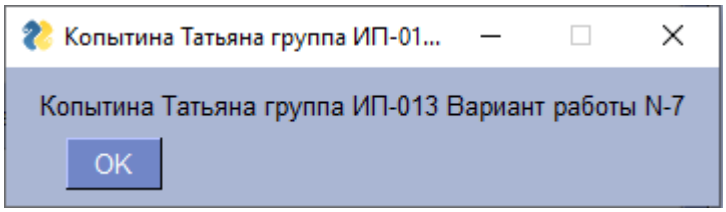


Рис 4. Отображение информации об авторе

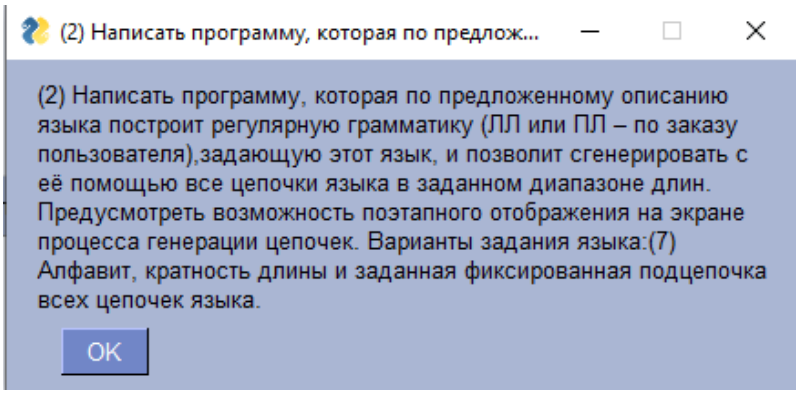


Рис 5. Отображение информации о задании на курсовую работу

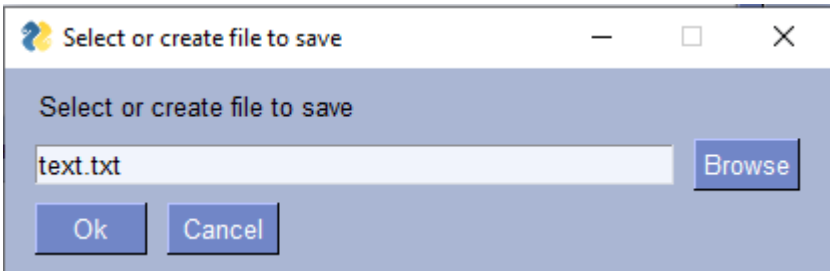


Рис 6. Заполнение формы сохранения выводимых цепочек в файл

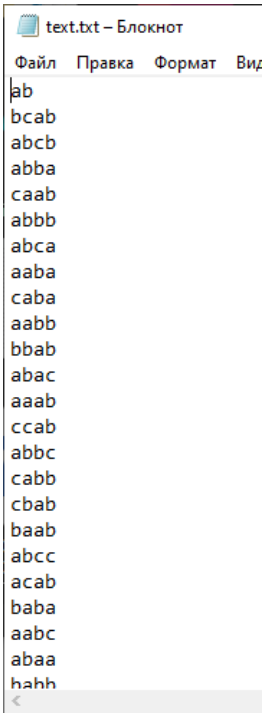


Рис 7. Отображение выходных данных в файле.

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()
```