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 6

Выполнил:

Cтудент гр. ИП-017 \_\_\_\_\_\_\_\_\_\_\_\_\_/Костин А.В./

ФИО студента

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Проверил:

Ассистент кафедры ПМиК \_\_\_\_\_\_\_\_\_\_\_\_\_/Новожилов Д.И./

ФИО преподавателя

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г. Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Новосибирск 2023 г.

Оглавление

[Задание 3](#_Toc153663911)

[Описание алгоритма решения задачи 4](#_Toc153663912)

[Описание основных функций программы 6](#_Toc153663913)

[Код программы 7](#_Toc153663914)

[Результат тестирования 18](#_Toc153663915)

**Задание**

*Вариант 6*

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.

Вариант задания языка: Алфавит, кратность вхождения некоторого символа алфавита и обязательная фиксированная подцепочка, на которую заканчиваются все цепочки языка.

**Описание алгоритма решения задачи**

Регулярная грамматика строится следующему принципу (лл и пл):

1. Считаем сколько раз встречается искомый символ внутри конечной подцепочки.
2. Создаем правила для дополнения до кратности. Если в подцепочке недостает искомого символа, то создается правила по кол-ву недостающих символов. Эти правила должны гарантировать необходимую кратность символа.
3. Теперь мы можем создать сколько угодно символов, пока не решим закончить генерацию. При этом сохраняется кратность символа.
4. Создаем конечную подцепочку.

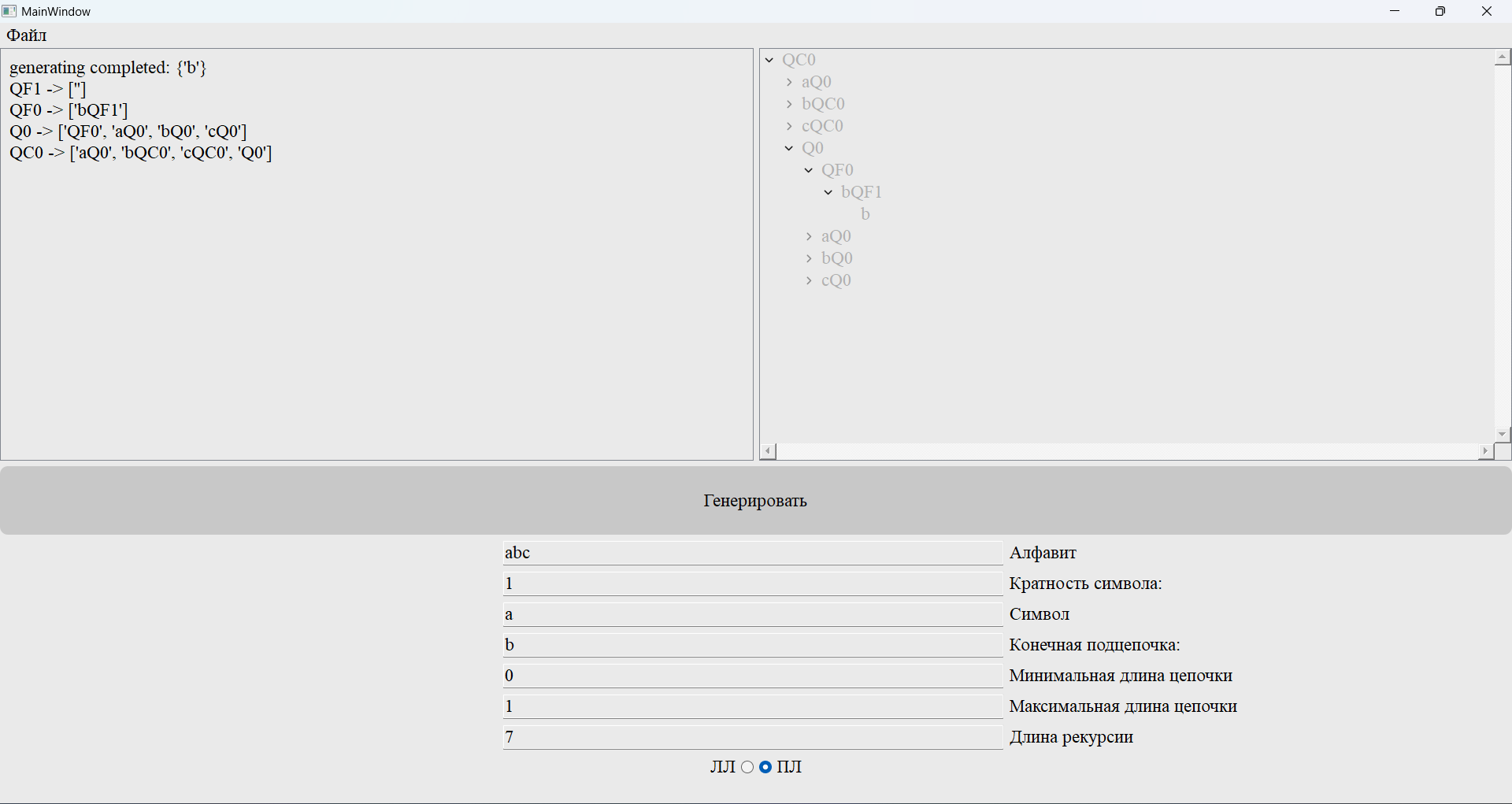


Рис1. Скриншот программы

В программе блоки правил отличаются друг от друга различными приписками к букве Q. Так, например, блок правил, генерирующий конечную подцепочку имеет приписку F, блок, добивающий кратность – С, блок, просто генерирующий символы не имеет приписки.

Начало в блоке С, программа генерирует различные символы пока не будет сгенерирован искомый символ, в данном примере – а. Как только кол-во искомый символов будет удовлетворять кратность, можно продолжить генерацию, либо уйти в построение конечной подцепочки. Если продолжать, то мы попадаем в блок Q и генерируем символы пока не будет снова достигнута кратность, после этого мы снова можем перейти к генерации конечной подцепочки или продолжить.

**Описание основных функций программы**

file\_import – отвечает за импорт параметров из файла

file\_export – отвечает за экспорт параметров в файл

init\_style – устанавливает стили на виджеты

on\_click – переключатель между лл и пл

generating\_rules – генерирует пл правила

generating\_left\_rules – генерирует лл правила

init\_data – задает изначальные значения переменным

start\_generating – начало генерации при нажатии на кнопку

generate – генерирует цепочки

normaliseResult – убирает недоделанные цепочки из результата

println – выводит сообщение в лог

**Код программы**

*Main.py*

import os.path  
import sqlite3  
import sys  
import time  
from PyQt5 import uic, QtCore, QtWidgets  
from PyQt5.QtSql import QSqlDatabase, QSqlQuery, QSqlTableModel, QSqlQueryModel  
from PyQt5.QtWidgets import \*  
from PyQt5.QtGui import \*  
from PyQt5.QtCore import \*  
  
import main\_window  
  
app = QApplication(sys.argv)  
  
Application = main\_window.App()  
Application.show()  
  
try:  
 sys.exit(app.exec\_())  
except SystemExit:  
 print('Closing Window...')

*Main\_window.py*

import os.path  
import sqlite3  
import sys  
import time  
from PyQt5 import uic, QtCore, QtWidgets, QtGui  
from PyQt5.QtSql import QSqlDatabase, QSqlQuery, QSqlTableModel, QSqlQueryModel  
from PyQt5.QtWidgets import \*  
from PyQt5.QtGui import \*  
from PyQt5.QtCore import \*  
from datetime import datetime  
import sqlite3  
  
  
class App(QMainWindow):  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.init\_style()  
 self.init\_data()  
 self.btn\_generate.clicked.connect(self.start\_generating)  
 self.btn\_aftor.triggered.connect(  
 lambda: QMessageBox.about(self, "Автор", "Костин Андрей Викторович ИП - 017"))  
 self.btn\_tema.triggered.connect(lambda: QMessageBox.about(self, "Тема",  
 "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.\nВариант задания языка: Алфавит, кратность вхождения некоторого символа алфавита и обязательная фиксированная подцепочка, на которую заканчиваются все цепочки языка"))  
 self.btn\_about.triggered.connect(lambda: QMessageBox.about(self, "Справка",  
 "Эта программа 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\nАлфавит - сюда написать алфавит языка\nКратность - сюда писать кратность символа языка(целое число)\nПодцепочка - обязательная фиксированная конечная последовательность в цепочке (состоит из символов алфавита)\nМинимальная длинна - это минимальная длина итоговых цепочек(целое число)\nМаксимальная длинна - это максимальная длина итоговых цепочек(целое число)\nДлинна рекурсии - количесто оборотов алгоритма"))  
  
 self.btn\_import.triggered.connect(lambda: self.file\_import())  
 self.btn\_export.triggered.connect(lambda: self.file\_export())  
 self.rbtn\_pl.toggled.connect(self.on\_click)  
 self.rbtn\_ll.toggled.connect(self.on\_click)  
  
 def file\_import(self):  
 path = QtWidgets.QFileDialog.getOpenFileName()[0]  
 print(path)  
  
 try:  
 file = open(path)  
 data = file.readlines()  
 self.le\_alphabet.setText(data[0][:-1])  
 self.le\_rate.setText(data[1][:-1])  
 self.le\_char.setText(data[2][:-1])  
 self.le\_child\_word.setText(data[3][:-1])  
 self.le\_min\_len.setText(data[4][:-1])  
 self.le\_max\_len.setText(data[5][:-1])  
 self.le\_step\_count.setText(data[6])  
 file.close()  
 except:  
 QMessageBox.about(self, "Ошибка", "Не корректный файл")  
  
  
 def file\_export(self):  
 path = QtWidgets.QFileDialog.getOpenFileName()[0]  
 print(path)  
  
 try:  
 file = open(path, "w")  
 file.write(self.le\_alphabet.text() + "\n")  
 file.write(self.le\_rate.text() + "\n")  
 file.write(self.le\_char.text() + "\n")  
 file.write(self.le\_child\_word.text() + "\n")  
 file.write(self.le\_min\_len.text() + "\n")  
 file.write(self.le\_max\_len.text() + "\n")  
 file.write(self.le\_step\_count.text())  
 file.close()  
 except:  
 QMessageBox.about(self, "Ошибка", "Не корректный файл")  
  
  
 def init\_style(self):  
 self.setWindowTitle("Курсовая легенды тяпа")  
 self.tableStyle = "QTableWidget{\ngridline-color: #666666}"  
 self.treeStyle = "QHeaderView::section {background-color: rgb(50, 50, 50);\ncolor: #b1b1b1;\npadding-left: 4px;\nborder: 1px solid #6c6c6c;\n}\n" \  
 "QHeaderView::section:hover{background-color: rgb(50, 50, 50);\nborder: 2px solid #ca8ad8;\ncolor: #fff;\n}\n" \  
 "QTreeView{show-decoration-selected: 1;\noutline: 0;\n}\n" \  
 "QTreeView::item {\ncolor: #b1b1b1;\n}\n" \  
 "QTreeView::item:hover{background: rgba(80, 120, 242, 100);\nborder-top: 1px solid #002cf2;\nborder-bottom: 1px solid #002cf2;\n}\n" \  
 "QTreeView::item:selected {background: rgb(80, 120, 242)}"  
 self.headerStyle = "::section:pressed {background-color: #323232;\nborder: none;}\n::section {background-color: #323232;\nborder: none;}"  
 self.btnCloseStyle = ":hover{\nbackground-color: darkred;\n}\n:pressed{\nbackground-color: red;\n}\nQPushButton{border:none}"  
 self.btnChangeStyle = ":hover{\nbackground-color: darkorange;\n}\n:pressed{\nbackground-color: orange;\n}\nQPushButton{border:none}"  
 self.btnOpenStyle = ':hover{\nbackground-color: darkgreen;\n}\n:pressed{\nbackground-color: green;\n}\nQPushButton{border:none} '  
 self.btnFolderStyle = ':hover{\nbackground-color: darkgreen;\n}\n:pressed{\nbackground-color: green;\n}\nQPushButton{border:none;\ntext-align: left;\nfont: 20px;} '  
  
 uic.loadUi('main\_window.ui', self)  
 self.treeWidget.setColumnCount(1)  
 self.treeWidget.setHeaderLabels([''])  
 self.treeWidget.header().hide()  
 self.treeWidget.setStyleSheet(self.treeStyle)  
  
 def on\_click(self):  
 if self.rbtn\_ll.isChecked():  
 self.rbtn\_pl.setChecked(False)  
  
 if self.rbtn\_pl.isChecked():  
 self.rbtn\_ll.setChecked(False)  
  
 def reverse\_rules(self):  
 for i in self.rules:  
 for j in range(len(self.rules[i])):  
 if len(self.rules[i][j]) != 0 and self.rules[i][j][0] in self.alphabet:  
 self.rules[i][j] = self.rules[i][j][1:] + self.rules[i][j][0]  
 print(self.rules[i][j])  
  
 def generating\_rules(self):  
 self.rules.clear()  
 count\_char = 0  
 for i in self.child\_word:  
 if i == self.char:  
 count\_char += 1  
 id = self.rate - count\_char % self.rate  
  
 if id == self.rate and count\_char > 0:  
 id = 0  
 self.rules["QC0"] = ["Q0"]  
 self.N.append("QC0")  
  
 for i in range(id):  
 l = []  
 for c in self.alphabet:  
 if c == self.char:  
 if i == id - 1:  
 l.append(c + "Q0")  
 else:  
 l.append(c + "QC" + str(i + 1))  
 else:  
 l.append(c + "QC" + str(i))  
 self.rules["QC" + str(i)] = l  
 self.N.append("QC" + str(i))  
  
 for i in range(self.rate):  
 l = []  
 if i == 0:  
 l.append("QF0")  
 for c in self.alphabet:  
 if c == self.char:  
 if i == self.rate - 1:  
 l.append(c + "Q" + str(0))  
 else:  
 l.append(c + "Q" + str(i + 1))  
 else:  
 l.append(c + "Q" + str(i))  
 self.rules["Q" + str(i)] = l  
 self.N.append("Q" + str(i))  
  
 for i in range(len(self.child\_word)):  
 self.rules["QF" + str(i)] = [self.child\_word[i] + "QF" + str(i + 1)]  
 self.N.append("QF" + str(i))  
 self.rules["QF" + str(len(self.child\_word))] = [""]  
 self.N.append("QF" + str(len(self.child\_word)))  
  
  
  
  
  
  
  
  
  
 def generating\_left\_rules(self):  
 self.rules.clear()  
 self.child\_word = self.child\_word[::-1]  
 for i in range(len(self.child\_word)):  
 self.rules["QF" + str(i)] = ["QF" + str(i + 1) + self.child\_word[i] ]  
 self.N.append("QF" + str(i))  
 self.rules["QF" + str(len(self.child\_word))] = [""]  
 self.N.append("QF" + str(len(self.child\_word)))  
  
 count\_char = 0  
 for i in self.child\_word:  
 if i == self.char:  
 count\_char+=1  
 id = self.rate - count\_char % self.rate  
  
 if id == self.rate and count\_char > 0:  
 id = 0  
 self.rules["QF" + str(len(self.child\_word))] = ["Q0"]  
 self.N.append("QF" + str(len(self.child\_word)))  
 else:  
 self.rules["QF" + str(len(self.child\_word))] = ["QC0"]  
 self.N.append("QF" + str(len(self.child\_word)))  
  
 for i in range(id):  
 l = []  
 for c in self.alphabet:  
 if c == self.char:  
 if i == id - 1:  
 l.append("Q0" + c)  
 else:  
 l.append("QC" + str(i+1) + c)  
 else:  
 l.append("QC" + str(i) + c)  
 self.rules["QC" + str(i)] = l  
 self.N.append("QC" + str(i))  
  
 for i in range(self.rate):  
 l = []  
 if i == 0:  
 l.append("")  
 for c in self.alphabet:  
 if c == self.char:  
 if i == self.rate - 1:  
 l.append("Q" + str(0) + c)  
 else:  
 l.append("Q" + str(i+1) + c)  
 else:  
 l.append("Q" + str(i) + c)  
 self.rules["Q" + str(i)] = l  
 self.N.append("Q" + str(i))  
  
  
 def init\_data(self):  
 self.rate = 3  
 self.child\_word = "aaa"  
 self.maxStepCount = 5  
 self.wordSizeMax = 5  
 self.wordSizeMin = 2  
 self.alphabet = ["a", "b", "c"]  
 self.N = ["Q0", "Q1", "Q2", "Q3", "Q4", "Q5", "Q6", "Q7", "Q8", "Q9", "Q10", "Q11"]  
 self.rules = {  
 "QI1": ["aQ11", "bQI2", "cQI2"],  
 "QI2": ["aQ21", "bQI3", "cQI3"],  
 "QI3": ["aQ31", "bQI1", "cQI1"],  
 "QI4": ["aQ41", "bQI1", "cQI1"],  
  
 "Q11": ["aQF2", "bQ1", "cQ1"],  
  
 "Q21": ["aQF1", "bQ1", "cQ2"],  
  
 "Q31": ["aQF4", "bQ1", "cQ1"],  
  
 "Q41": ["aQF3", "bQ1", "cQ1"],  
  
 "QF4": ["aQF3", "bQF3", "cQF3", ""],  
 "QF3": ["aQF2", "bQF2", "cQF2"],  
 "QF2": ["aQF1", "bQF1", "cQF1"],  
 "QF1": ["aQF4", "bQF4", "cQF4"],  
 }  
  
 self.rules = {  
 "QI1": ["aQ11", "bQI2", "cQI2"],  
 "QI2": ["aQ21", "bQI3", "cQI3"],  
 "QI3": ["aQ31", "bQI1", "cQI1"],  
 # 3 - ( 3 - 5%3 ) + 1  
 "Q11": ["aQ12", "bQI3", "cQI3"],  
 "Q12": ["aQF3", "bQI1", "cQI1"],  
  
 "Q21": ["aQ22", "bQI1", "cQI1"],  
 "Q22": ["aQF2", "bQI2", "cQI2"],  
  
 "Q31": ["aQ32", "bQI2", "cQI2"],  
 "Q32": ["aQF1", "bQI3", "cQI3"],  
  
 "QF3": ["aQF2", "bQF2", "cQF2", ""],  
 "QF2": ["aQF1", "bQF1", "cQF1"],  
 "QF1": ["aQF3", "bQF3", "cQF3"],  
 }  
 self.char = 'a'  
 self.result = []  
 self.absoluteResult = set()  
  
 def start\_generating(self):  
 self.maxStepCount = self.le\_step\_count.text()  
 if self.maxStepCount == "":  
 QMessageBox.about(self, "Ошибка", "Необходимо указать длину рекурсии")  
 return  
 if not self.maxStepCount.isdigit():  
 QMessageBox.about(self, "Ошибка", "Необходимо указать длину рекурсии целым числом")  
 return  
 self.maxStepCount = int(self.maxStepCount)  
  
 self.char = self.le\_char.text()  
 if self.char == "" or len(self.char) > 1:  
 QMessageBox.about(self, "Ошибка", "Необходимо указать один символ")  
 return  
  
 self.wordSizeMin = self.le\_min\_len.text()  
 if self.wordSizeMin == "":  
 QMessageBox.about(self, "Ошибка", "Необходимо указать минимальную длинну")  
 return  
 if not self.wordSizeMin.isdigit():  
 QMessageBox.about(self, "Ошибка", "Необходимо указать минимальную длину целым числом")  
 return  
 self.wordSizeMin = int(self.wordSizeMin)  
 self.wordSizeMax = self.le\_max\_len.text()  
 if self.wordSizeMax == "":  
 QMessageBox.about(self, "Ошибка", "Необходимо указать максимальную длинну")  
 return  
 if not self.wordSizeMax.isdigit():  
 QMessageBox.about(self, "Ошибка", "Необходимо указать максимальную длину целым числом")  
 return  
 self.wordSizeMax = int(self.wordSizeMax)  
  
 self.rate = self.le\_rate.text()  
 if self.rate == "":  
 QMessageBox.about(self, "Ошибка", "Необходимо указать кратность")  
 return  
 if not self.rate.isdigit():  
 QMessageBox.about(self, "Ошибка", "Необходимо указать кратность целым числом")  
 return  
 self.rate = int(self.rate)  
  
 self.child\_word = self.le\_child\_word.text()  
 # if self.child\_word == "":  
 # QMessageBox.about(self, "Ошибка", "Необходимо указать подцепочку")  
 # return  
  
 alphabet = self.le\_alphabet.text()  
 if alphabet == "":  
 QMessageBox.about(self, "Ошибка", "Необходимо указать алфавит")  
 return  
  
 self.alphabet = []  
 for i in alphabet:  
 if i in self.alphabet:  
 QMessageBox.about(self, "Предупреждение",  
 "Вы меня не победить,\nПовторяющиеся символы в алфавите были пропущенны")  
 else:  
 self.alphabet.append(i)  
  
 if self.wordSizeMax < self.wordSizeMin:  
 QMessageBox.about(self, "Ошибка",  
 "Вы меня не победить,\nМаксимальная длинна должна быть больше минимальной")  
 return  
  
 for i in self.child\_word:  
 if i not in self.alphabet:  
 QMessageBox.about(self, "Ошибка",  
 "Вы меня не победить,\nПодцепочка содержит символы не из алфавита")  
 return  
 self.result = []  
 self.absoluteResult = set()  
  
 self.treeWidget.clear()  
 root = QTreeWidgetItem(self.treeWidget)  
 root.setText(0, "QC0")  
 if self.rbtn\_ll.isChecked():  
 root.setText(0, "QF0")  
 self.generating\_left\_rules()  
 else:  
 self.generating\_rules()  
  
 # if (self.rbtn\_ll.isChecked()):  
 # self.reverse\_rules()  
 for i in self.rules:  
 self.println(f"{i} -> {self.rules[i]}")  
 self.generate(root)  
 self.normaliseResult()  
  
 self.println(f"generating completed: {self.absoluteResult}")  
  
 def generate(self, thisNode, stepCount=0):  
 if stepCount >= self.maxStepCount or thisNode.text(0) == "":  
 # print(f"answer = {thisWord}")  
 return  
  
 for key in reversed(self.rules):  
 if key not in thisNode.text(0):  
 continue  
  
 for val in self.rules[key]:  
 # print(f"{stepCount}) {thisNode.text(0)} ({key}: {val})")  
 child = QTreeWidgetItem(thisNode)  
 child.setText(0, thisNode.text(0).replace(key, val, 1))  
 self.result.append(child.text(0))  
 self.generate(child, stepCount + 1)  
 break  
  
 def normaliseResult(self):  
 for word in self.result:  
 if not (self.wordSizeMin <= len(word) <= self.wordSizeMax):  
 continue  
  
 is\_correct = True  
 for key in self.N:  
 if key in word:  
 is\_correct = False  
 break  
  
 if is\_correct:  
 self.absoluteResult.add(word)  
  
 def println(self, text):  
 current\_datetime = datetime.now()  
 self.log.setText(str(text) + "\n" + self.log.text())  
 logFile = open("log.txt.", "a")  
 logFile.write(str(current\_datetime) + ": " + str(text) + "\n")  
 logFile.close()

Результат тестирования

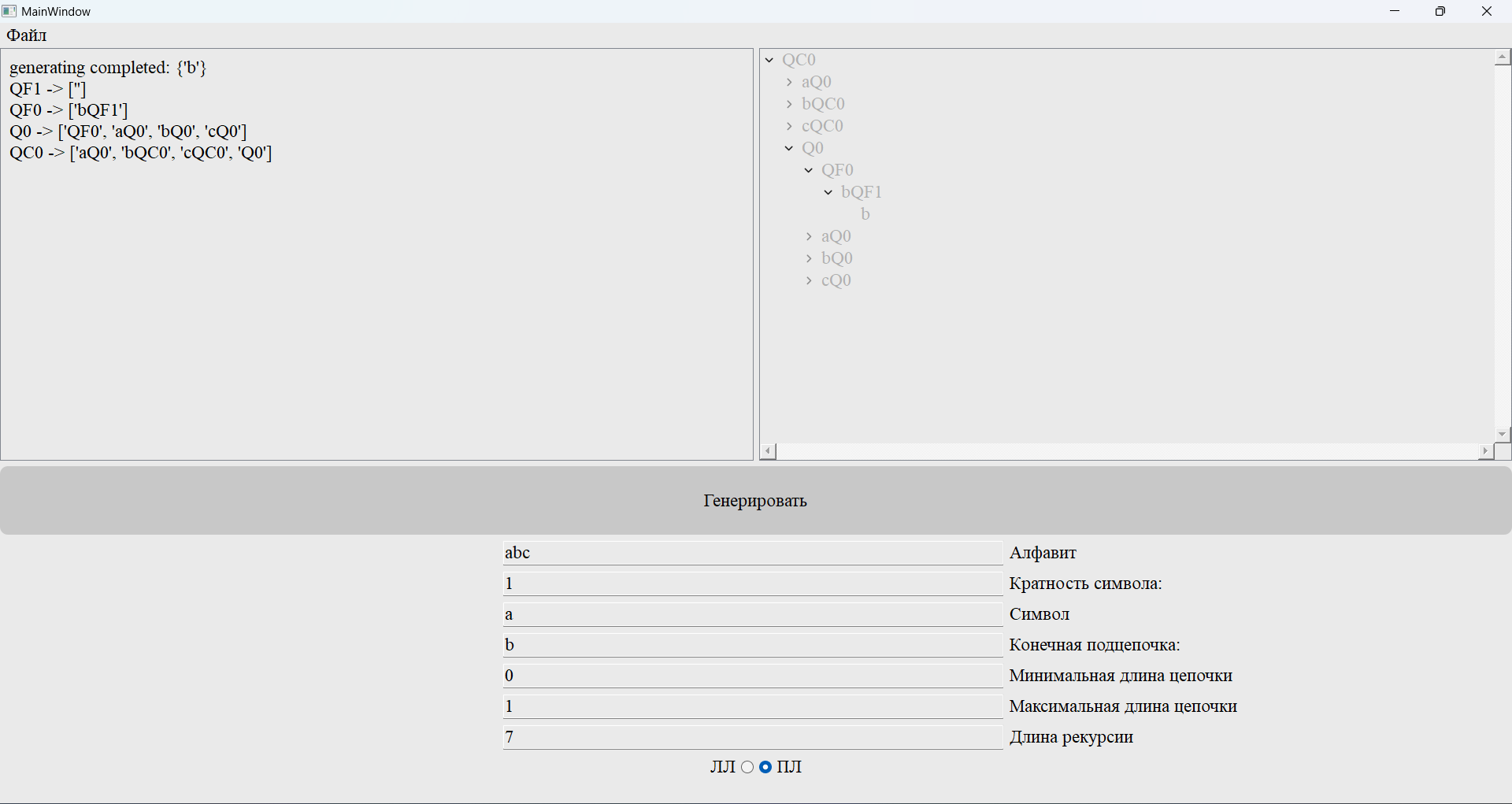


Рис2. Скриншот программы

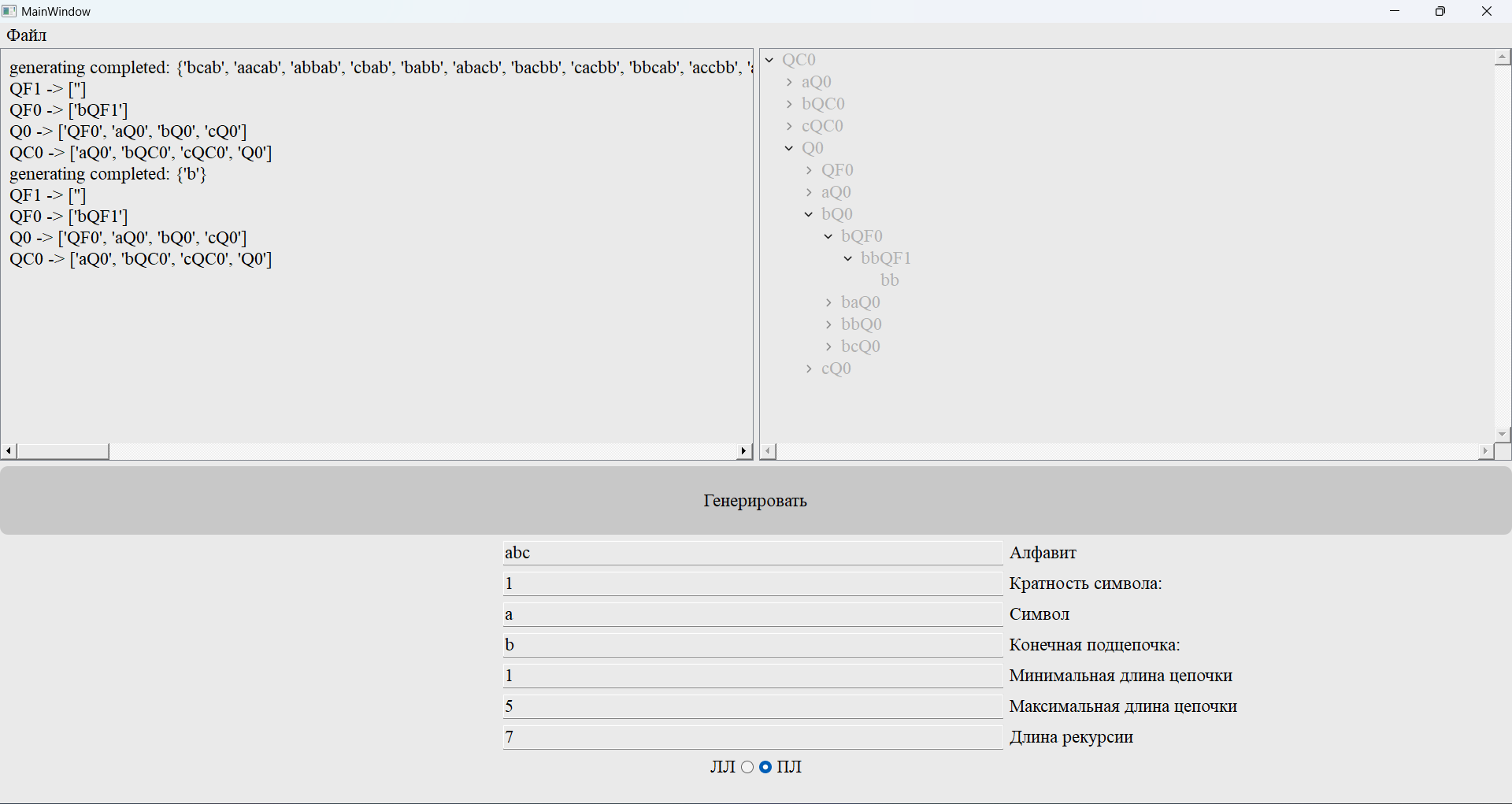


Рис3. Скриншот программы

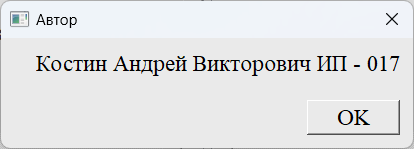


Рис4. Скриншот окна автора

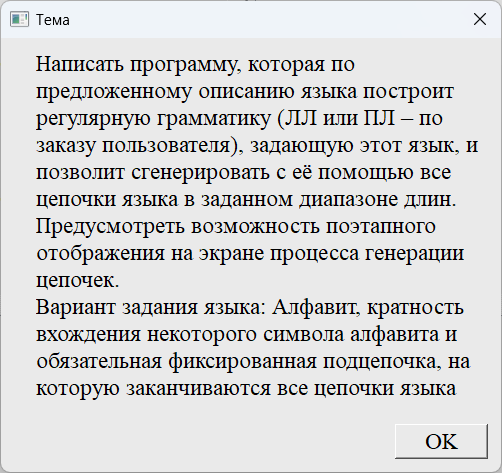


Рис5. Скриншот окна темы

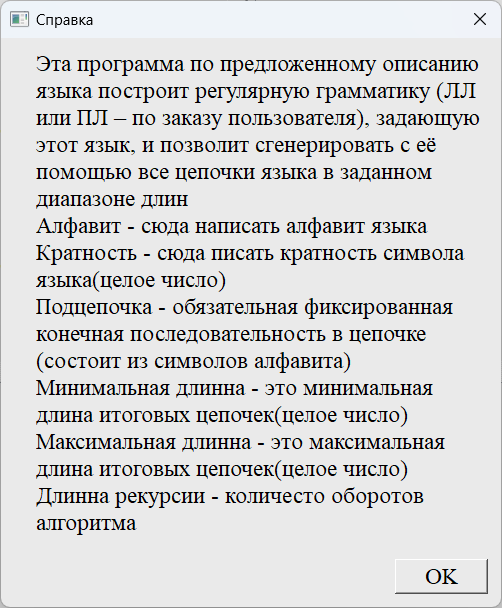


Рис6. Скриншот окна справки