

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ  
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ  
ПЕТРА ВЕЛИКОГО»**

Институт Компьютерных Наук и Кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление 02.03.01 Математика и Компьютерные Науки

Лабораторная работа №2

По дисциплине:

«Математическая логика и теория автоматов»

Тема Работы:

«Проверка и генерация регулярных выражений»  
«Вариант 14»

Обучающийся: \_\_\_\_\_ Черепанов Михаил Дмитриевич

Руководитель: \_\_\_\_\_ Востров Алексей Владимирович

«\_\_\_\_\_» \_\_\_\_\_ 20\_\_ г.

Санкт-Петербург 2024

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Математическое описание</b>	<b>4</b>
1.1 Регулярные множества и регулярные выражения . . . . .	4
1.2 Регулярное выражение CSS-атрибута . . . . .	4
1.3 Недетерминированный конечный автомат . . . . .	5
1.4 Детерминированный конечный автомат . . . . .	6
<b>2 Особенности реализации</b>	<b>7</b>
2.1 Реализация состояний конечного автомата . . . . .	8
2.2 Реализация таблицы переходов конечного автомата . . . . .	8
2.3 Реализация анализа строки на соответствие регулярному выражению	8
2.4 Реализация автоматической генерации строки . . . . .	9
2.5 Функция main . . . . .	10
<b>3 Результаты работы</b>	<b>12</b>
<b>4 Заключение</b>	<b>15</b>
<b>Список использованных источников</b>	<b>15</b>

# Введение

В данной работе требуется:

По заданному регулярному выражению построить вначале недетерминированный конечный автомат, затем детерминировать его. Реализовать программу, которая проверяет введенный текст, через реализацию конечного автомата (варианты вывода: строка соответствует, не соответствует, символы не из алфавита). Также необходимо реализовать функцию случайной генерации верной строки по полученному конечному автомату.

Мой вариант:

№14. Поиск CSS-атрибутов

Регулярное выражение:

$\wedge \backslash s * [a-zA-Z \backslash -] + \backslash s * [ : ] \{ 1 \} \backslash s [a-zA-Z 0-9 \backslash s . \#] + [ ; ] \{ 1 \}$

# 1 Математическое описание

## 1.1 Регулярные множества и регулярные выражения

**Регулярные множества** – это (бесконечные) множества цепочек, построенных из символов конечного словаря по определенным правилам.

Формально класс регулярных множеств над конечным словарем  $\Sigma$ , определяется так:

- $\emptyset = \{\}$  - регулярное множество;
- $\{\varepsilon\}$  - регулярное множество;
- $\{a\}$  - регулярное множество для любого  $a \in \Sigma$ ;

Если  $P$  и  $Q$  - регулярные множества, то регулярны следующие операции над ними:

1. Объединение  $P \cup Q$ ;
2. Произведение  $P \cap Q$ ;
3. Итерация  $P^*$ ;
4. Усеченная итерация  $P^+$ .

**Регулярные выражения** — формальный язык шаблонов для поиска и выполнения манипуляций с подстроками в тексте.

Регулярное выражение показывает, как можно построить регулярное множество цепочек из одноэлементных множеств с использованием трех вышеперечисленных операций.

Формально класс регулярных выражений над конечным словарем  $\Sigma$  определяется так:

- $\emptyset$  - регулярное множество
- $\emptyset, \varepsilon$  - и любой символ  $a \in \Sigma$  - регулярные выражения;

Если  $R_1, R_2$  - регулярные выражения, то регулярными выражениями будут:

1. их сумма  $R_1 + R_2$ ;
2. их произведение (конкатенация)  $R_1 R_2$ ;
3. итерация  $R_1^*$  и усеченная итерация  $R_1^+$ ;

## 1.2 Регулярное выражение CSS-атрибута

**Регулярное выражение CSS-атрибута:**

$\backslash s * [a - z A - Z \backslash -] + \backslash s * [;] \{1\} \backslash s [a - z A - Z 0 - 9 \backslash s . \#] + [;] \{1\}$

$*$  – повторение фрагмента 0 или более раз.

$+$  – повторение фрагмента 1 или более раз.

$\{N\}$  – ровно  $N$  повторений предыдущего символа (группы).

Таким образом, данное регулярное выражение можно интерпретировать следующим образом:

1.  $\backslash s^*$  – любое количество пробельных символов.
2.  $[a - zA - Z\backslash -]^+$  – последовательность из одного или более буквенных символов английского алфавита, дефисов;
3.  $\backslash s^*$  – любое количество пробельных символов;
4.  $[:]\{1\}$  – двоеточие;
5.  $\backslash s$  – пробельный символ;
6.  $[a - zA - Z0 - 9\backslash s.\#]^+$  – последовательность из одного или более буквенных символов английского алфавита, пробелов, цифрных символов, точки, решетки;
7.  $;$  – точка с запятой.

### 1.3 Недетерминированный конечный автомат

Недетерминированным конечным автоматом-распознавателем называется пятерка:

$$A = \langle S, X, S_0, \delta, F \rangle, \text{ где}$$

- $S$  - конечное непустое множество состояний;
- $X$  - конечное непустое множество входных сигналов (входной алфавит);
- $S_0 \in S$  - множество начальных состояний;
- $\delta : S \times \{X \cup \epsilon\} \rightarrow 2^S$  - функция переходов,  $2^S$  - булеан множества  $S$ ;
- $F \in S$  - множество заключительных (финальных) состояний.

Недетерминированный автомат отличается от детерминированного возможностью наличия пустых переходов и нескольких входных состояний. Кроме этого в недетерминированном автомате могут отсутствовать переходы по некоторым из символов алфавита.

По исходному регулярному выражению был построен следующий недетерминированный конечный автомат (рис. 1):

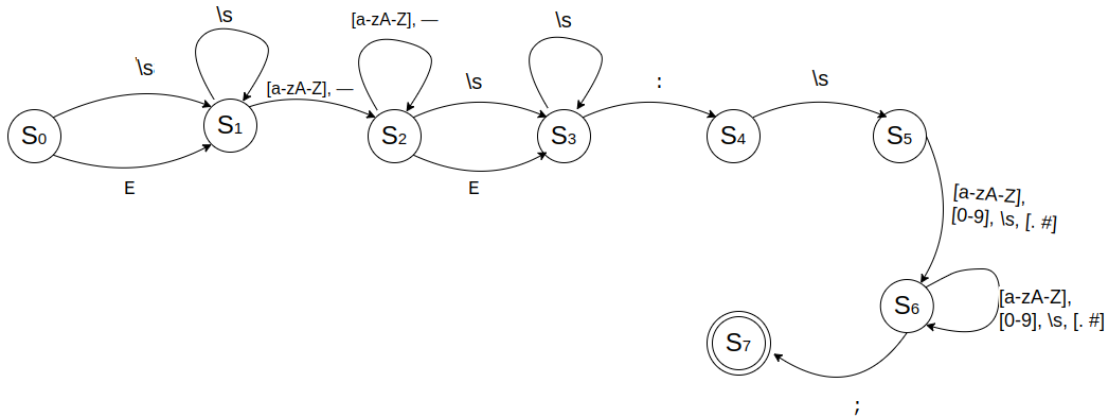


Рис. 1: Недетерминированный конечный автомат

Для моего недетерминированного автомата:

$S = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, \}$

$X$  - символы алфавита ASCII

$F = \{S_7\}$

$\delta : S \times X \rightarrow S$ , функция переходов представлена в виде таблице на рис. 2.

	E	\s	A-zA-Z	0-9	-	:	,	#	;	other
S0	S1	S1	-	-	-	-	-	-	-	-
S1	-	S1	S2	-	S2	-	-	-	-	-
S2	S3	S3	S2	-	S2	-	-	-	-	-
S3	-	S3	-	-	-	S4	-	-	-	-
S4	-	S5	-	-	-	-	-	-	-	-
S5	-	S6	S6	S6	-	-	S6	S6	-	-
S6	-	S6	S6	S6	Se	Se	S6	S6	S7	Se
S7	-	-	-	-	-	-	-	-	-	-

Рис. 2: Таблица переходов недетерминированного автомата

## 1.4 Детерминированный конечный автомат

**Детерминированный конечный автомат-распознаватель** - это пятерка объектов:

$A = \langle S, X, s_0, \delta, F \rangle$ , где:

- $S$  - конечное непустое множество состояний;
- $X$  - конечное непустое множество входных сигналов (входной алфавит);
- $s_0 \in S$  - начальное состояние;
- $\delta : S \times X \rightarrow S$  - функция переходов;
- $F \in S$  - множество заключительных(финальных) состояний.

По представленному выше недетерминированному автомату был построен сле-

дующий детерминированный автомат (рис. 3):

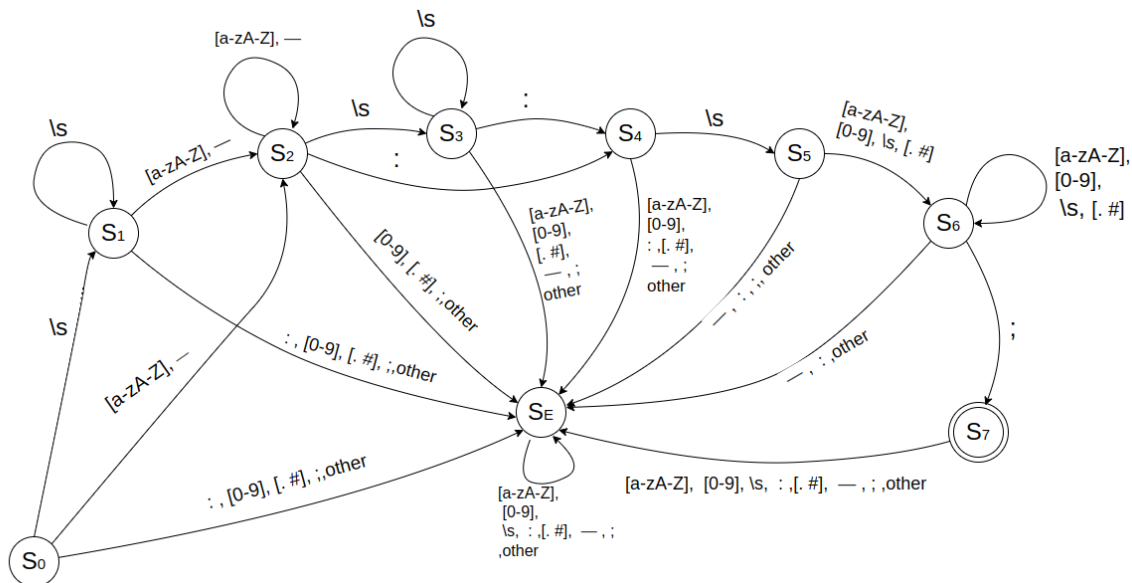


Рис. 3: Детерминированный конечный автомат

Для моего детерминированного автомата:

$$S = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_E\}$$

$X$  - символы алфавита ASCII

$$F = \{S_7\}$$

$\delta : S \times X \rightarrow S$ , функция переходов представлена в виде таблице на рис. 4.

	<b>\s</b>	<b>A-a-Z</b>	<b>0-9</b>	<b>-</b>	<b>:</b>	<b>.</b>	<b>#</b>	<b>;</b>	<b>other</b>
<b>S0</b>	S1	S2	Se	S2	Se	Se	Se	Se	Se
<b>S1</b>	S1	S2	Se	S2	Se	Se	Se	Se	Se
<b>S2</b>	S3	S2	Se	S2	Se	Se	Se	Se	Se
<b>S3</b>	S3	Se	Se	Se	S4	Se	Se	Se	Se
<b>S4</b>	S5	Se	Se	Se	Se	Se	Se	Se	Se
<b>S5</b>	S6	S6	S6	Se	Se	S6	Se	Se	Se
<b>S6</b>	S6	S6	S6	Se	Se	S6	S6	S7	Se
<b>S7</b>	Se	Se	Se	Se	Se	Se	Se	Se	Se
<b>Se</b>	Se	Se	Se	Se	Se	Se	Se	Se	Se

Рис. 4: Таблица переходов детерминированного автомата

## 2 Особенности реализации

В рамках данной лабораторной был реализован класс RegExprAnalyze, который содержит блок перечисления состояний автомата, а так же следующие основные методы:

```

void CreateTable();
bool Analyse(string);
pair<string,string> GenerateOutput();
string GetRandomStr()

```

В функции main реализован ввод с консоли и вывод на консоль, вызовы методов класса RegExpAnalyze.

## 2.1 Реализация состояний конечного автомата

Состояния автомата описанны в виде блока перечисления:

```

enum State{
    S0,
    S1,
    S2,
    S3,
    S4,
    S5,
    S6,
    S7,
    Se
};

```

## 2.2 Реализация таблицы переходов конечного автомата

Таблица переходов создается в методе CreateTable() и хранится в виде матрицы.

Программная реализация соответствует таблице представленной на рис. 4.

Исходный код функции не приведен, поскольку он громоздок и тривиален.

## 2.3 Реализация анализа строки на соответствие регулярному выражению

Анализ строки происходит в методе Analyse(string) класса RegExpAnalyze.

**Входные данные:** Строка для анализа.

**Выходные данные:** True / False.

В теле функции реализован проход по всем символам строки и переход состояний автомата, в соответствии с таблицей.

После обработки всей строки, в случае, если автомат оказался в конечном состоянии - программа вернет True, иначе - False.

**Исходный код функции:**

```

bool RegExpAnalyze::Analyse(string str){
    curr_state=S0;
    next_state=S0;
    for(int i=0;i<str.size();i++){

```



```

        curr_state=table_state[curr_state][str[i]];
    }
    if(curr_state==S7){
        return true;
    }
    else{
        return false;
    }
}

```

## 2.4 Реализация автоматической генерации строки

Автоматическая генерация строки, соответствующей регулярному выражению реализована в методе GetRandomStr класса RegExpAnalyze.

**Входные данные:** Таблица переходов конечного автомата.

**Выходные данные:** Строка, соответствующая регулярному выражению.

В теле функции генерируются символы из алфавита ASCII в случайном порядке.

Если в результате входа обработки сгенерированного символа автомат переходит в состояние ошибки - переход отменяется, символ генерируется снова.

Иначе, генерация продолжается, пока автомат не окажется в финальном состоянии.

Для того, чтобы генерируемая строка не становилась неуместно длинной, в состоянии S3 генерируются только пробел и двоеточие;

В состоянии S4 - только пробел.

**Исходный код функции:**

```

QString RegExpAnalyze::GetRandomStr(){
    QString ans="";
    curr_state=S0;
    while(curr_state!=S7){
        int simbol=rand()%128;

        if(curr_state==S3){
            int r=rand()%2;
            if(r)
                simbol=' ';
            else
                simbol=': ';
            next_state=table_state[curr_state][simbol];
        }
        else if(curr_state==S4){
            simbol=' ';
            next_state=table_state[curr_state][simbol];
        }
    }
}

```

```

    else{
        next_state=table_state[curr_state][simbol];
        while(next_state==Se){
            simbol=rand()%128;
            next_state=table_state[curr_state][simbol];
        }
    }
    curr_state=next_state;
    ans+=simbol;
}
return ans;
}

```

## 2.5 Функция main

В функции main в бесконечном цикле запрашивается ввод пользователя строки для анализа.

При вводе символа «G» - программа генерирует новую строку;

При вводе символа «Q» - выход из программы.

В функции main так же вызывается функция CheckSimvols(string), которая проверяет наличие в строке символов не из алфавита. В случае, если в строке присутствуют символы не из алфавита, выводится сообщение об ошибке, анализ строки не происходит.

**Исходный код функции main:**

```

int main(int argc, char *argv[])
{
    srand(time(0));
    QCoreApplication a(argc, argv);
    string str="";
    RegExpAnalyze analyzator;
    while(true){
        str="";
        qDebug()<<"Enter <<G>> for auto generation string";
        qDebug()<<"Enter <<Q>> for quit";
        qDebug()<<"Or enter the string for check";
        std::getline(std::cin, str);
        if(str=="Q")
            break;
        else if(str=="G"){
            qDebug()<<analyzator.GetRandomStr();
        }
        else{
            if(!CheckSimvols(str))
                qDebug()<<"There are characters not from the alphabet!";
        }
    }
}

```

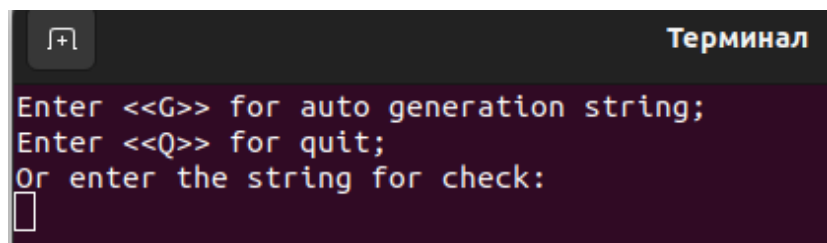
```

        Try again.";
    else{
        bool result=analyzer.Analyse(str);
        if(result)
            qDebug()<<"The string is correct.";
        else{
            qDebug()<<"The string is incorrect.";
        }
    }
}
qDebug()<<"\n\n\n";
}
return 0;
}

```

### 3 Результаты работы

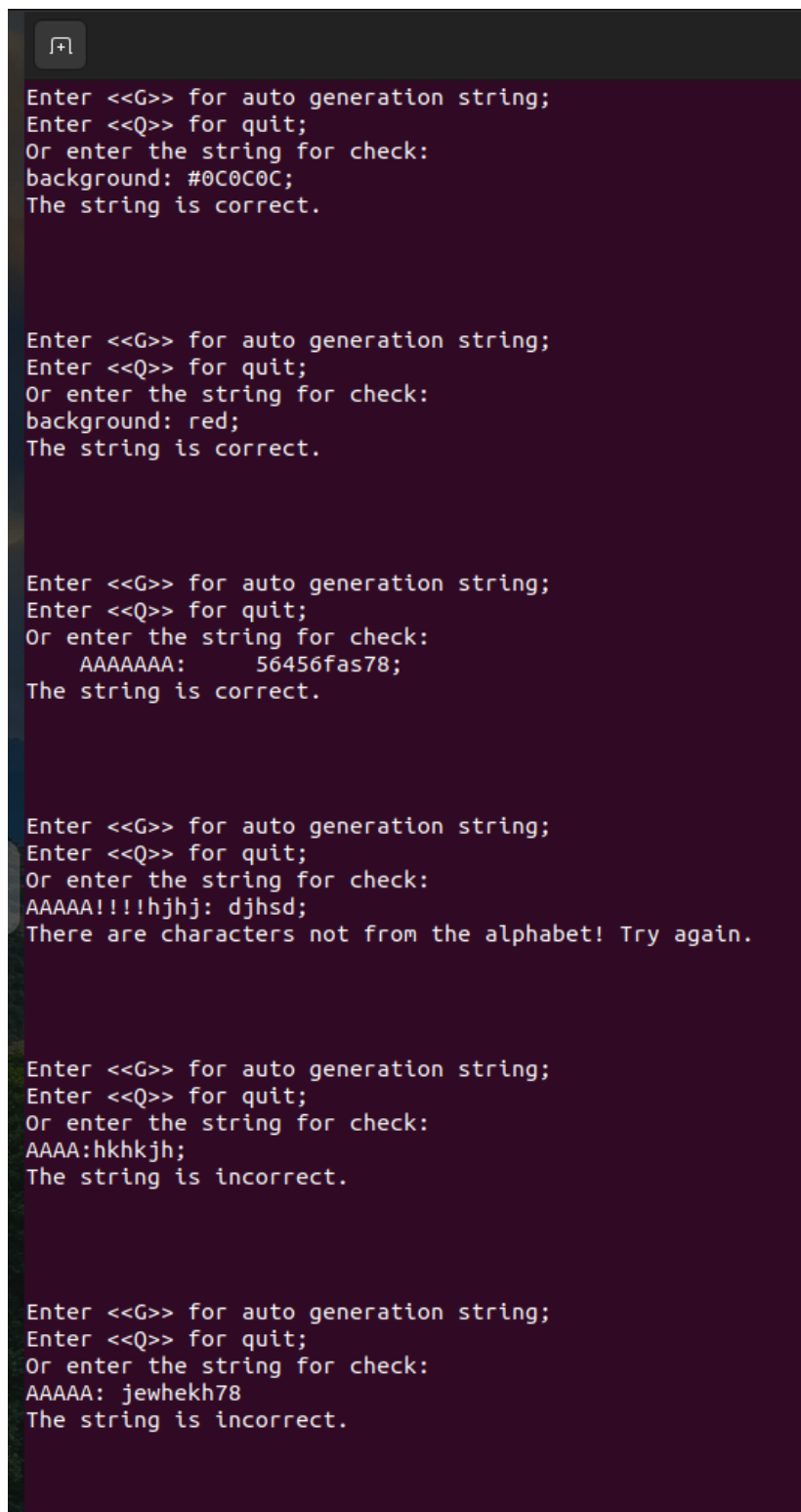
На рис. 5 показано пользовательское меню программы:



```
Enter <<G>> for auto generation string;  
Enter <<Q>> for quit;  
Or enter the string for check:  
□
```

Рис. 5: Пользовательское меню.

На рис. 6 показана проверка бти введенных пользователем строк:



```
Enter <<G>> for auto generation string;
Enter <<Q>> for quit;
Or enter the string for check:
background: #0C0C0C;
The string is correct.

Enter <<G>> for auto generation string;
Enter <<Q>> for quit;
Or enter the string for check:
background: red;
The string is correct.

Enter <<G>> for auto generation string;
Enter <<Q>> for quit;
Or enter the string for check:
AAAAAAA: 56456fas78;
The string is correct.

Enter <<G>> for auto generation string;
Enter <<Q>> for quit;
Or enter the string for check:
AAAAA!!!hjhj: djhsd;
There are characters not from the alphabet! Try again.

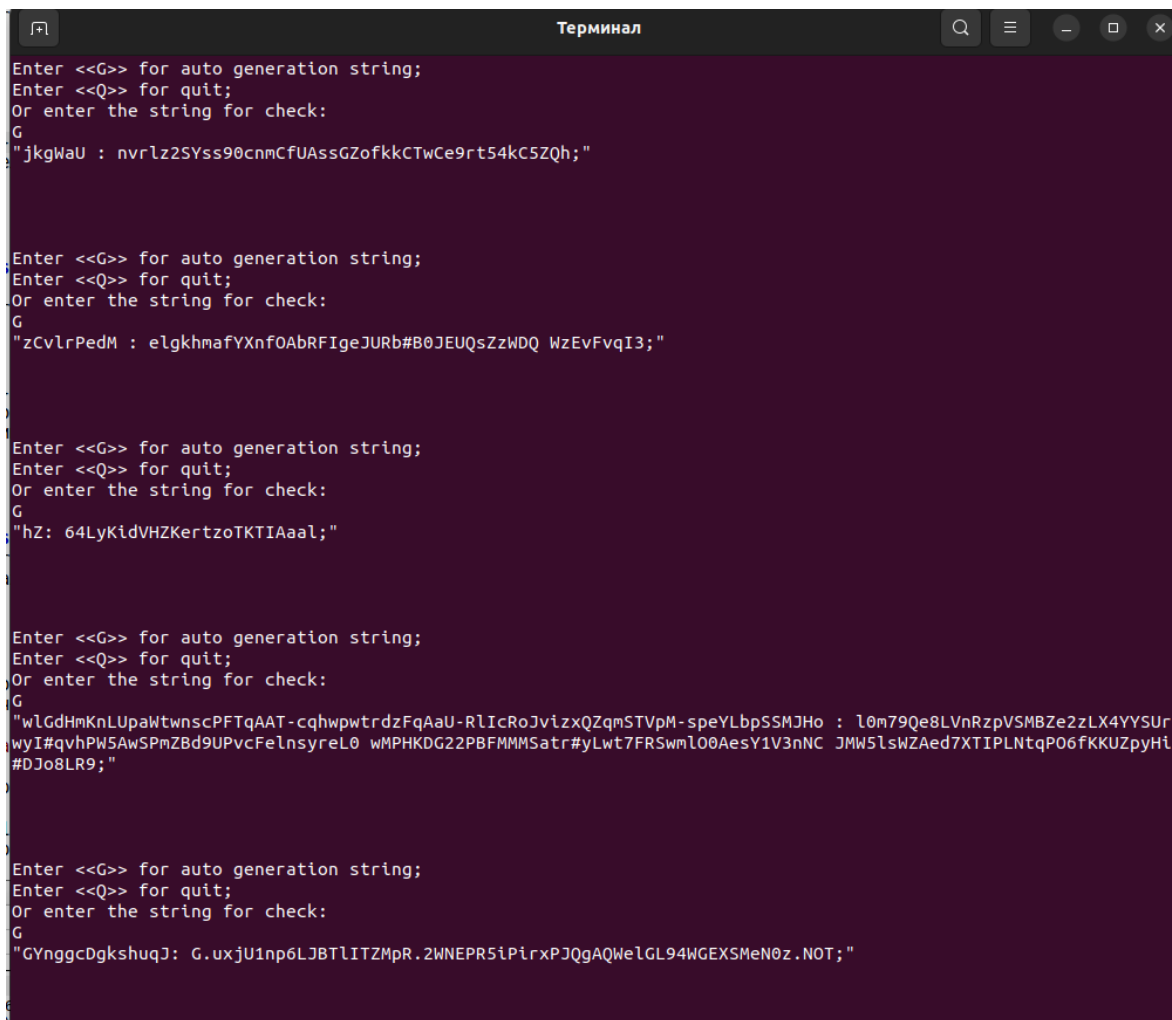
Enter <<G>> for auto generation string;
Enter <<Q>> for quit;
Or enter the string for check:
AAAA:hkhkjh;
The string is incorrect.

Enter <<G>> for auto generation string;
Enter <<Q>> for quit;
Or enter the string for check:
AAAAA: jewhekh78
The string is incorrect.
```

Рис. 6: Проверка введенных пользователем строк.

На рис. 7 показаны 5 случайно сгенерированных программой строк, соответ-

свующих исходному регулярному выражению:



```
Терминал
Enter <G> for auto generation string;
Enter <Q> for quit;
Or enter the string for check:
G
"jkgWau : nvrlz2SYss90cnmCfUAssGZofkkCTwCe9rt54kCSZQh;"

Enter <G> for auto generation string;
Enter <Q> for quit;
Or enter the string for check:
G
"zCvLrPedM : elgkhmafYXnf0AbRFIgeJURb#B0JEUQsZzWDQ WzEvFvqI3;"

Enter <G> for auto generation string;
Enter <Q> for quit;
Or enter the string for check:
G
"hZ: 64LyKidVHZKertzoTKTIAaal;"

Enter <G> for auto generation string;
Enter <Q> for quit;
Or enter the string for check:
G
"wLGdHmKnLUpaWtwnscPFTqAAT-cqhwptwtrdzFqAaU-RlIcRoJvizxQZqmSTVpM-speYlbpSSMJHo : l0m79Qe8LVnRzpVSMBZe2zLX4YYSUr
wyI#qvhPW5AwSPmZBd9UPvcFelnsyreL0 wMPHKDG22PBFMMMSatr#yLwt7FRSwmL00AesY1V3nNC JMW5lSWZAed7XTIPLNtqP06fKKUZpyHl
#DJo8LR9;"

Enter <G> for auto generation string;
Enter <Q> for quit;
Or enter the string for check:
G
"GYnggcDgkshuqJ: G.uxju1np6LJBTLITZMpR.2WNEPR5iPirxPJQgAQWGL94WGEYSMeN0z.NOT;"
```

Рис. 7: Автоматически сгенерированные выражения.

## 4 Заключение

Итак, в результате работы была реализована программа, позволяющая проверять строки на их соответствие регулярному выражению и для автоматической генерации строк, по нему.

Регулярное выражение:

$\backslash s * [a - zA - Z \backslash -] + \backslash s * [:] \{1\} \backslash s [a - zA - Z 0 - 9 \backslash s . \#] + [;] \{1\}$

**Достоинства реализации:**

1. Реализация переходов автомата с помощью таблицы( а не блока условий), что позволяет легче масштабировать программу.

**Недостатки реализации:**

1. Примитивный пользовательский интерфейс.

**Масштабирование:**

Программу можно масштабировать путем добавления новых состояний конечного автомата. Для этого нужно только изменить таблицу переходов и блок перечисления состояний.

## Список использованных источников

- [1] Ф.А. Новиков. Дискретная математика для программистов: Учебник для вузов. 3-е изд. — СПб: Питер, 2008. —384 с.
- [2] А.В. Востров. Лекция №6 по дисциплине «Математическая логика и теория автоматов».