МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра математического обеспечения ЭВМ

Лексический анализатор

ЛАБОРАТОРНАЯ РАБОТА

02.03.02 Фундаментальная информатика и информационные технологии

Инженерия программного обеспечения

Студент Брянцев В. А.

Воронеж 2024

1. Постановка задачи

Цель работы: разработка программы, реализующей конечный автомат для распознавания строк, которые начинаются и заканчиваются на символ двойных кавычек (“), между кавычками находятся восьмеричные константы и не идущие подряд двойные кавычки. Строки проверяются на соответствие заданной грамматике. Программа должна считывать строки с файла и выводить результаты проверки с описанием переходов по состояниям конечного автомата.

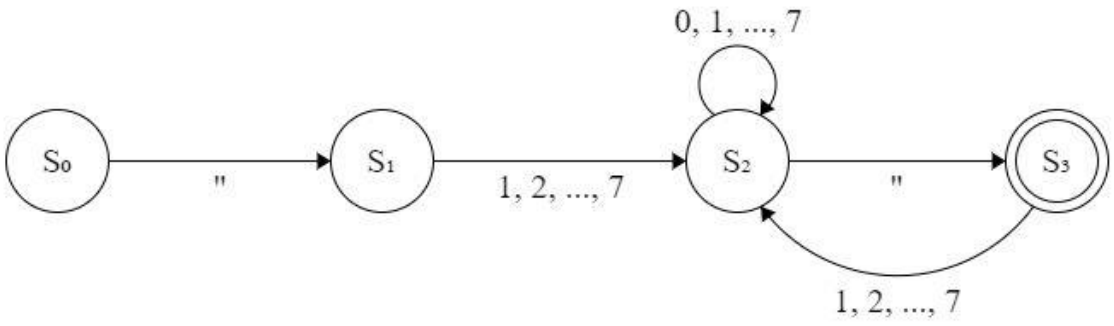
2. Разработка программы.

2.1. Модель и метод решения задачи

Для решения поставленной задачи используется модель конечного автомата. Автомат на основе считываемого символа переходит из одного состояния в другое. Моделируемый автомат имеет четыре состояния:

- *S0* – начальное состояние, принимающее символ двойных кавычек (“);
- *S1* – состояние, принимающее цифры от 1 до 7 для обработки начала восьмеричной константы;
- *S2* – состояние, принимающее либо цифры от 0 до 7, либо кавычки;
- *S3* – состояние, принимающее цифры от 1 до 7, в случае отсутствия входящего символа является конечным состоянием;
- *ERROR* – состояние ошибки, в которое автомат переходит при обнаружении символа, несоответствующего заданной грамматике.

Ниже представлена диаграмма конечного автомата и таблица переходов:



Состояние	0-7	1-7	“	ε
S0	∅	∅	S1	∅
S1	∅	S2	∅	∅
S2	S2	S2	S3	∅
S3	∅	S2	∅	∅

2.2. Описание структуры данных

Структура программы основана на перечислимом типе State, описывающем состояния конечного автомата:

- *S0* – начальное состояние;
- *S1* – состояние обработки “;
- *S2* – состояние обработки восьмеричных цифр;
- *S3* – конечное состояние;
- *ERROR* – состояние ошибки.

Входными данными для обработки являются строки, считываемые из файла. Каждая строка проверяется на соответствие допустимой лексике.

2.3. Алгоритм решения задачи

1) Программа считывает строку из файла.
2) Строка обрабатывается посимвольно, для каждого символа совершается переход из одного состояния в другое в соответствии с таблицей переходов:
    - В состоянии *S0* программа ожидает символ двойных кавычек (“). После обработки кавычек происходит переход в состояние *S1*.
    - В состоянии *S1* программа ожидает цифры от 1 до 7. После обработки цифры совершается переход в состояние *S2*.
    - В состоянии *S2* программа ожидает цифры от 0 до 7 или кавычки. При обработке цифры состояние остается тем же, а при обработке кавычек совершается переход в состояние *S3*.
    - В состоянии *S3* программа ожидает цифры от 1 до 7. При обработке цифры происходит переход в состояние *S2*.

- При несоответствии символа ожидаемому происходит переход в состояние ошибки *ERROR*.

3) Если в состоянии *S3* больше не остается символов для обработки, то это состояние становится конечным, а строка считается допустимой.
4) Вывод результата.

3. Руководство программиста

Программа состоит из основной функции *main()*, функции *transition()*, в которой происходит переход между состояниями конечного автомата, и функции *isValidString()*, которая проверяет входящую строку на корректность.

Программа написана на языке программирования C++. Она использует стандартные библиотеки:

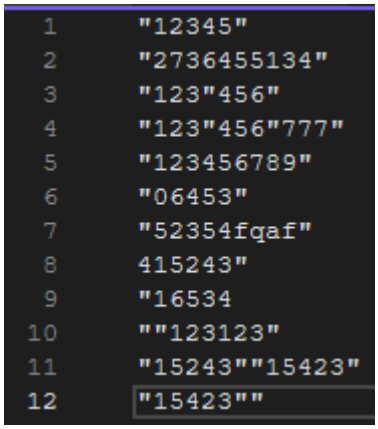
- *iostream* для консольного ввода и вывода;
- *fstream* для файлового ввода и вывода;
- *string* для работы со строками.

4. Руководство пользователя

Для работы программы пользователю необходимо подготовить файл *test_file.txt*, в котором должны содержаться строки. Данные строки будут считаны с файла программой и проверены на соответствие заданной грамматике. Программа будет выводить переходы по состояниям, а также результат проверки для каждой строки.

5. Тестирование программы и его результаты

Входные данные:



Выходные данные:

```
S0 -> S1
S1 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S3
Line "12345" is valid
```

```
S0 -> S1
S1 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S3
Line "2736455134" is valid
```

```
S0 -> S1
S1 -> S2
S2 -> S2
S2 -> S2
S2 -> S3
S3 -> S2
S2 -> S2
S2 -> S2
S2 -> S3
Line "123"456" is valid
```

```
S0 -> S1
S1 -> S2
S2 -> S2
S2 -> S2
S2 -> S3
S3 -> S2
S2 -> S2
S2 -> S2
S2 -> S3
S3 -> S2
S2 -> S2
S2 -> S2
S2 -> S3
Line "123"456"777" is valid
```

```
S0 -> S1
S1 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
Error in S2
Line "123456789" is invalid
```

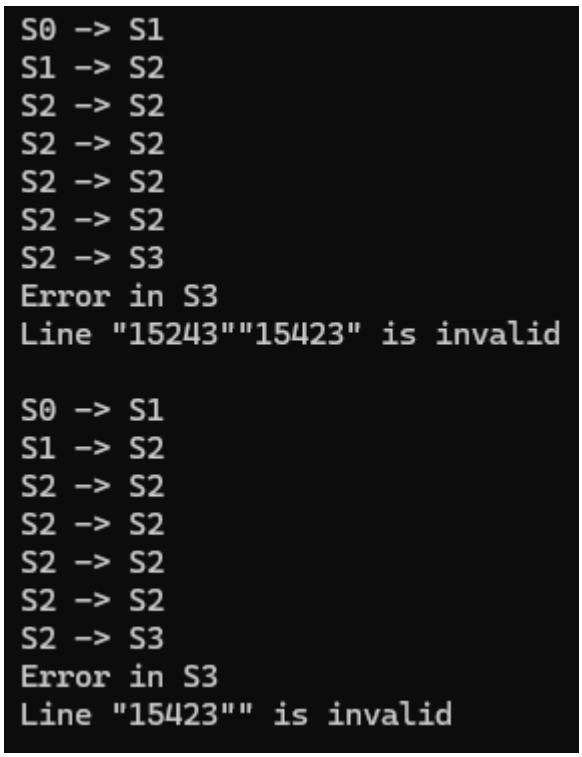
```
S0 -> S1
Error in S1
Line "06453" is invalid

S0 -> S1
S1 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
Error in S2
Line "52354fqaf" is invalid

Error in S0
Line 415243" is invalid

S0 -> S1
S1 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
S2 -> S2
Line "16534 is invalid

S0 -> S1
Error in S1
Line ""123123" is invalid
```



6. Результаты работы программы и их анализ

Программа успешно считала и распознала строки, соответствующие заданной грамматике, и корректно обработала недопустимые строки.

7. Выводы

Разработанная программа успешно решает задачу распознавания строк, состоящих из кавычек и восьмеричных констант. Программа выполнена в виде конечного автомата, что делает ее легко расширяемой для более сложных грамматик. Тестирование показало корректность работы программы для различных тестовых входных данных.

Приложение

```
// 10. "........\123...\321...\..."..."
// Строковая константа, содержащая восьмеричные константы и кавычки

#include <iostream>
#include <fstream>
#include <string>

enum State
{
	S0, // начальное состояние
	S1, // переход по открывающим кавычкам
	S2, // переход по цифрам 0, 1, 2, 3, 4, 5, 6, 7
	S3, // переход по закрывающим кавычкам к конечному состоянию
	ERROR // ошибка
};

State transition(State currentState, char symbol)
{
	switch (currentState)
	{
	case S0:
		if (symbol == '"')
		{
			std::cout << "S0 -> S1\n";
			return S1;
		}
		std::cout << "Error in S0\n";
		return ERROR;
	case S1:
		if (symbol == '1' || symbol == '2' || symbol == '3' || symbol == '4'
|| symbol == '5' || symbol == '6' || symbol == '7')
		{
			std::cout << "S1 -> S2\n";
			return S2;
		}
		std::cout << "Error in S1\n";
		return ERROR;
	case S2:
		if (symbol == '0' || symbol == '1' || symbol == '2' || symbol == '3'
|| symbol == '4' || symbol == '5' || symbol == '6' || symbol == '7')
		{
			std::cout << "S2 -> S2\n";
			return S2;
		}
		if (symbol == '"')
		{
			std::cout << "S2 -> S3\n";
			return S3;
		}
		std::cout << "Error in S2\n";
		return ERROR;
	case S3:
		if (symbol == '1' || symbol == '2' || symbol == '3' || symbol == '4'
|| symbol == '5' || symbol == '6' || symbol == '7')
		{
			std::cout << "S3 -> S2\n";
			return S2;
		}
		std::cout << "Error in S3\n";
		return ERROR;
	default:
		std::cout << "Unexpected error\n";
		return ERROR;
	}
```

```
}

bool isValidString(const std::string& line)
{
	State currentState = S0; // Начальное состояние
	for (char symbol : line)
	{
		currentState = transition(currentState, symbol);
		if (currentState == ERROR)
		{
			return false;
		}
	}
	return currentState == S3; // Конечное состояние
}

int main()
{
	std::ifstream test_file("tests.txt");
	std::string line;

	while (std::getline(test_file, line))
	{
		if (isValidString(line))
		{
			std::cout << "Line " << line << " is valid\n\n";
		}
		else
		{
			std::cout << "Line " << line << " is invalid\n\n";
		}
	}
	test_file.close();
}
```