Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра информационных технологий и систем

**Проектирование лексического анализатора**

Лабораторная работа

по дисциплине:

«Теория языков программирования и методы трансляции»

Разработал:

Студент группы 8091

Шаклеин В. В. \_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2020г.

Проверил:

Макаров В. А. \_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2020г.

**Великий Новгород**

**2020**

# Постановка задачи

## Формулировка

Разработать программу, решающую задачу лексического разбора исходного текста программы.

Результат работы лексического анализатора выводятся в файл в виде таблицы распознанных лексем. Таблица должна следующие поля содержать: номер строки и атрибуты лексемы (имя лексемы, класс лексемы (подкласс), значение лексемы).

## Представление входных данных

Грамматика языка была составлена путём выборочного смешения 17 и 19 вариантов из документа “Варианты грамматик 2020”.

С добавлением конструкции for (… of …) по аналогии с for (… in …) и отказом от дифференциации целых и вещественных чисел, поскольку таковая отсутствует в реальном языке JavaScript.

Алфавит:**if, for**, **in**, **of**, **return**, ‘**“**’, ‘**=**’, ‘**\*=**’, ‘**/=**’, ‘**+=**’,  ‘**-=**’, ‘**&**’, ‘**|**’, ‘**&&**’, ‘**||**’, ‘**(**‘, ‘**)**’, ‘**{**’, ‘**}**’,  ‘**;**’, ‘**.**’, ‘**\_**’, ‘**$**’, [**a-z]**, [**A-Z**], [**0-9**]

Классы лексем:

1. Ключевые слова языка (**if**, **for**, **in, of**, **return**);
2. Разделители и знаки операций (‘**>**’, ‘**<**’, ‘**>=**’, ‘**<=**’, ‘**&**’, ‘**|**’, ‘**&&**’, ‘**||**’, ‘**(**‘, ‘**)**’, ‘**{**’, ‘**}**’,  ‘**;**’) ;
3. Знаки операции присваивания (‘**=**’, ‘**\*=**’, ‘**/=**’, ‘**+=**’, ‘**-=**’);
4. Идентификаторы;
5. Числовая константа;
6. Строка в двойных кавычках;

## Описание выбранного метода решения задачи лексического анализа

В качестве основного метода лексического анализа был выбран конечный автомат, поскольку это наиболее оптимальный метод для обработки комментариев и сложных констант. Однако для оптимизации разработки было принято решение прибегнуть к некоторым ухищрениям, позволяющим распознавать лексемы, состоящие из двух символов без создания дополнительных состояний.

Для хранения таблицы ключевых слов и идентификаторов я решил использовать словарь std::map, поскольку от него можно ожидать высокой производительности.

# Описание реализации

Из-за использования ухищрений позволяющих проще распознавать двухсимвольные лексемы приведение полной схемы конечного автомата представляется крайне затруднительной задачей, поэтому привожу упрощённую схему. Кроме того, опущены подробности по распознаванию сложных численных констант, вместо этого напишу на основе какого регулярного выражения оно построено: **[-+]?[0-9]\*\.?[0-9]+([eE][-+]?[0-9]\*)?**

При обнаружении лексемы класса **ID** проверяется, является ли она ключевым словом, известным идентификатором или требует создания новой записи в таблице идентификаторов.

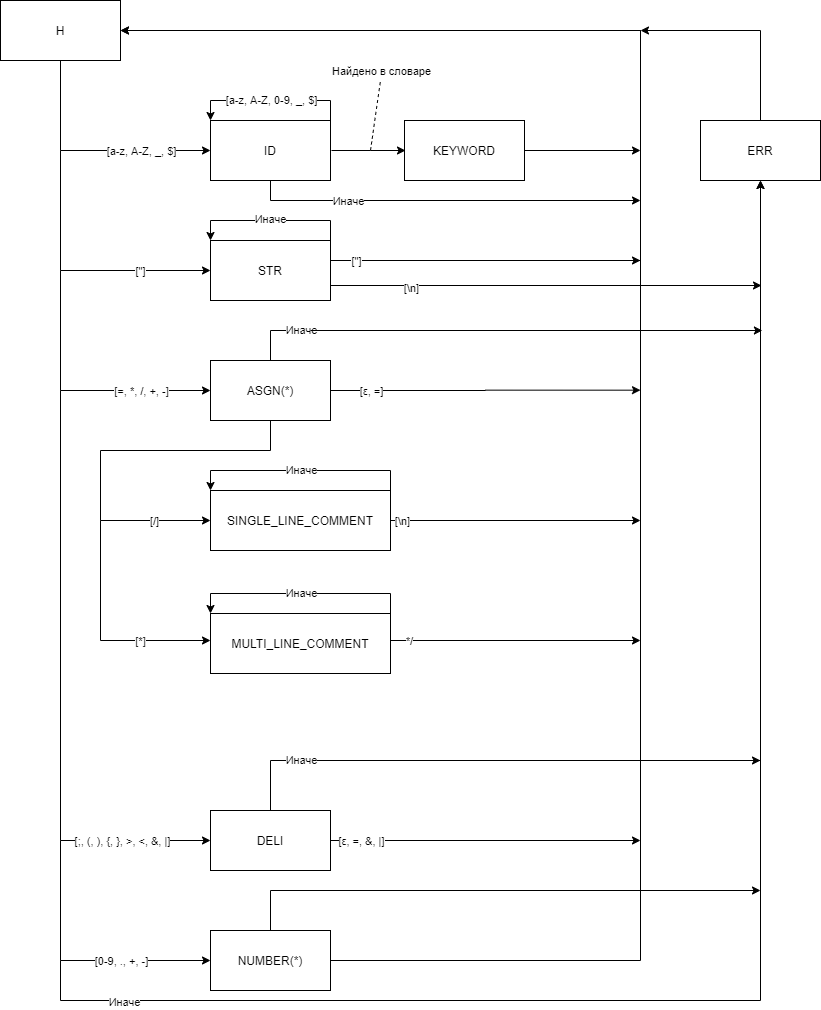


Рисунок 1 Упрощенная схема конечного автомата

На рисунке \* отмечены “умные” состояния

# Обработка ошибок

Лексема помечается как ошибочная если следующий символ является недопустимым продолжением для неё или в случае, если ожидается начало новой лексемы, но входной символ не соответствует началу ни одной из известных лексем.

После обнаружения ошибочной лексемы сканирование файла продолжается.

# Методика тестирования

Для проверки работы анализатора, разработаны следующие тестовые файлы

## Генератор выбранного числа Фибоначи

**n = 10;**

**a = 0;**

**b = 1;**

**for (i = 1;i <= n;i += 1)**

**{**

**c = b;**

**b += a;**

**a = c;**

**}**

**return a;**

## Набор типовых лексем

**//i am comment**

**/\***

**big weird multiline comment**

**" \* /**

**///**

**/\***

**\*/**

**"12345" //string**

**";re// /\* " //string with comment symbols**

**"fsd\"" //string with escaped "**

**if (a>b && c < b || b >= c) {}**

**for (i = 0;i < 1e10;i += 1) {**

**a = a|b;**

**c = a & b;**

**d \*= c;**

**c /= d;**

**if (c > d)**

**{**

**c += d;**

**}**

**}**

**if**

**for**

**return 1234**

**a<=b**

**//Weird stuff**

**.**

**..**

**.e**

**.e1**

**1.e**

**12fasg**

**12.2.45**

**1e3e5**

**"123**

**456**

**"**

**"123**

**/////////**

## Тест распознавания чисел

Так как тест имеет огромный размер, в отчёт вставлено только его начало.

**+287**

**54**

**+0e+8**

**47E4**

**.89**

**+36.8E+5**

**-06.29E+7**

**-5**

**-49e3**

**86.2**

**-81.5E-5**

**+5**

**855**

**0.5**

**+91.8E+5**

**08E3**

**983e1**

**-11**

**…**

# Выводы

Мной была собрана грамматика, на основе которой возможно написание минимально полезных программ и не имеющая отклонений от настоящей грамматики JavaScript. Также мной был разработан лексический анализатор для этой грамматики, способный различать комментарии, а так же сложные числовые и строчные константы.