МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра вычислительной техники

**ОТЧЁТ**

**по лабораторной работе №2**

***«*** *Лексика языков программирования. Конечные автоматы без памяти для обнаружения слов в тексте программы* ***»***

**по дисциплине: «** *Теория формальных языков и компиляторов***»**

Вариант: 24443144

Выполнила:Проверил:

Студентка гр. АВТ-709 *« к.т.н., доцент »*

*« Васильева Н.В. » « Малявко А.А. »*

«\_\_\_» \_\_\_\_\_\_ 2020г.«\_\_\_» \_\_\_\_\_\_ 2020 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск

2020

Цели работы

Изучение конечных автоматов (КА) без памяти, способов определения КА –канонического, графового и табличного, методов построения недетерминированного КА по системе регулярных выражений, методов эквивалентных преобразований недетерминированных КА в оптимальные полностью определенные КА – лексические акцепторы.

**Вариант: 24443144**

1. *Идентификаторы*

|  |
| --- |
| **2** |
| <бБ><пЦ><бБ>  ( d23U, N1q, x15y, …) |

1. *Константы*

|  |
| --- |
| **4** |
| целые по основанию 10 и указанному в константе;  вещественные;  символьные |

1. *Объявления примитивных типов (целое, вещественное, символьное):*

|  |
| --- |
| **4** |
| **card[inal][\_u]**  **double**  **litera** |

1. *Оператор присваивания:*

|  |
| --- |
| **4** |
| **put**<В> **to** <И> ; |

1. *Условный оператор:*

|  |
| --- |
| **3** |
| **when** <ЛВ> **then** <ОБ> **[ other** <ОБ> **]** |

1. *Оператор цикла:*

|  |
| --- |
| **1** |
| **loop** <ОБ> **until** <ЛВ> |

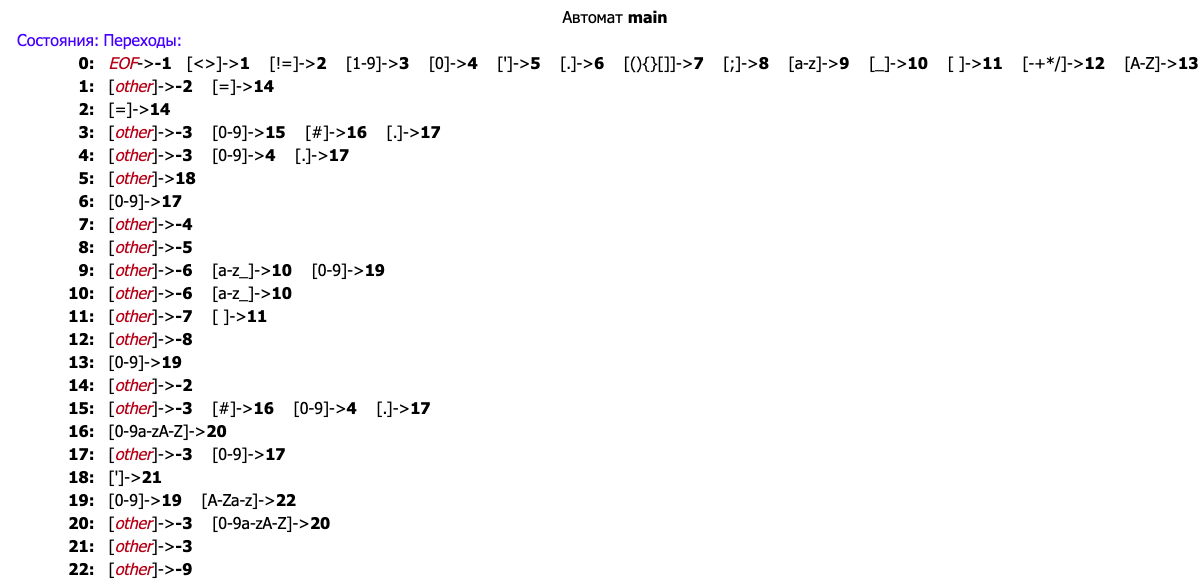
1. *Оператор переключателя*

|  |
| --- |
| **4** |
| **??** <В> **{ ?** <К> **:** <ОБ> **[gout;]** … **[ ?~** **:** <ОБ> **]** **}** |

1. *Формат псевдокода*

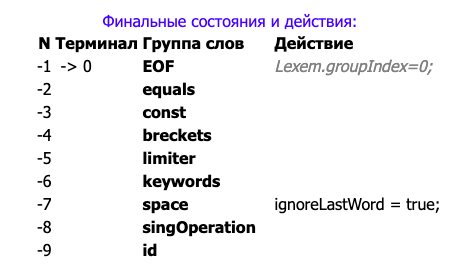
|  |
| --- |
| **4** |
| Триады <Код> <ОП> <Р> |

# Описание структуры программных модулей конечных автоматов, управляемых таблично и графом состояний и переходов

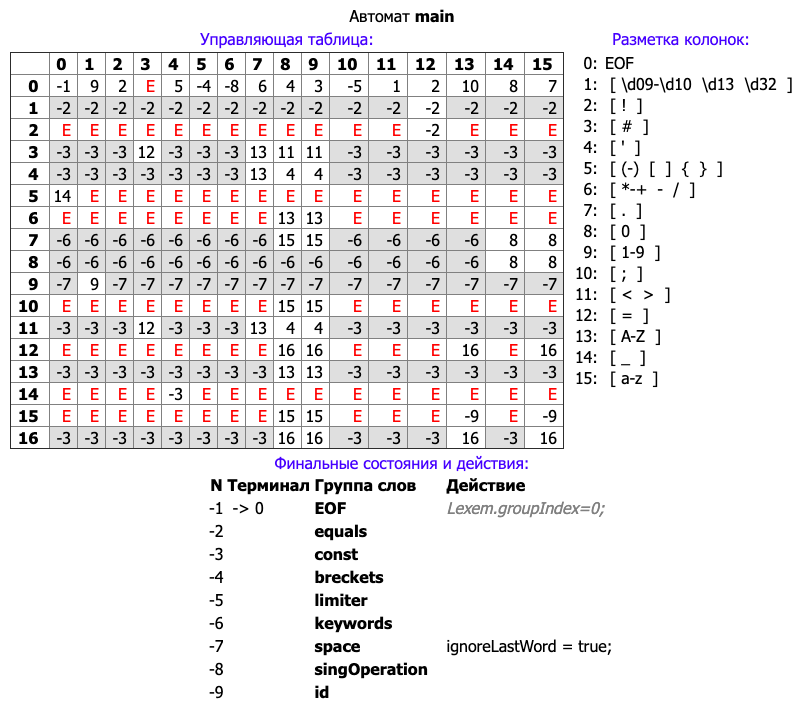


*Рисунок 1. Граф состояний и переходов*

Список состояний и возможных переходов из них. На вход анализатора подается символ и обрабатывается в соответствии с графом состояний. Например, из состояния 0 можно перейти в одно из 13 (1-13), далее либо в промежуточное (рабочее), либо в финальное состояние (рисунок 2).



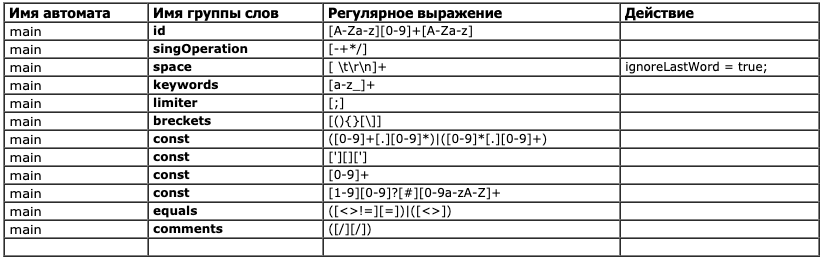
*Рисунок 2. Финальные состояния и действия*



*Рисунок 3. Управляющая таблица*

Если число в ячейке таблицы красное Е – это останов по ошибке, больше 0 – это переход в рабочее состояние, если число меньше 0 – переход в финальное состояние, если число меньше 0 на сером фоне – переход в финальное состояние с возвратом литеры на вход. Каждая строка таблицы соответствует своему состоянию.

# Система регулярных выражений



# Описание языка

**Файлы:**

Программа располагается в одном файле, может состоять из одной(main) или нескольких функций, где главное является main(), с нее и начинается выполнение.

Функции могут принимать параметры, т.е. некоторые значения, передаваемые функции для того, чтобы она что-либо сделалас ними. Эти параметры похожи на переменные, за исключением того, что значение этих переменных указывается при вызове функции, и во время работы функции им уже присвоены их значения. Параметры указываются в скобках при объявлении функции и разделяются запятыми. Аналогично мы передаём значения, когда вызываем функцию.

Константы имеют область видимости. В случае определения константы в теле функции/цикла область ее видимости - функция/цикл, в котором она была определена – *локальная* переменная (в случае, если это не глобальная переменная). Переменные, объявленные вне функции, но доступные внутри функции, называются ***глобальными***.

**Строки и отступы:**

Программа представляет собой последовательность логических строк, каждая из которых может состоять из одной или нескольких физических строк. Каждая физическая строка может заканчиваться комментарием.

Строка, содержащая лишь пробелы и, возможно, комментарии, называется пустой: такие строки игнорируются. В конец физической строки в большинстве случаев означает конец инструкции.

Для организации блочной структуры программ используются отступы. Отступы создаются путем смещения начала логической строки блока от левого края на определенное количество пробелов(обычно 4 пробела или Tab). *Блок* - это последовательность смежных логических строк, имеющих одинаковые отступы; логическая строка с меньшим отступом означает конец блока, также можно указывать блок в фигурных скобках, чтобы не будет являться ошибкой.

Для создания строки, располагающейся на двух строках, можно использовать символ перевода строки. Пример:

'Не очень длинная строка, которая выводится\n

на двух строках' // использование комментария в предыдущей

// строке запрещено

**Лексемы:**

Каждая логическая строка разбирается на ряд элементарных лексических компонентов, так называемых *лексем*. Каждая лексема соответствует подстроке логической строки. К числу обычных типов лексем относятся идентификаторы, ключевые слова, операторы, разделители. Для разделения лексем можно свободно использовать пробелы.

**Литералы:**

*Литерал* - это значение, указанное непосредственно в программе. Ниже приведены примеры:

35 // Целочисленный литерал

3.14 // Литерал числа с плавающей точкой

‘g’ // Символьный литерал

**Идентификаторы:**

Идентификатор должен начинаться с буквы (это буквы от А до z и от а до z;) за которыми могут следовать или не следовать одна или несколько букв и цифры.

**Выполняемые операторы:**

Оператор присваивания:

**put<В> to <И>**

<В> - произвольное выражение

<И> - идентификатор

Условный оператор:

**when <ЛВ> then <ОБ> [ other <ОБ> ]**

<ЛВ> - логическое выражение

<ОБ> - оператор или блок

Оператор Цикла:

**loop <ОБ> until <ЛВ>**

<ЛВ> - логическое выражение

<ОБ> - оператор или блок

Оператор переключателя:

**?? <В> { ? <К> : <ОБ> [gout;] … [ ?~ : <ОБ> ] }**

<В> - произвольное выражение

<К> - константа

<ОБ> - оператор или блок

*Выражение* - это "фраза" кода, которую язык вычисляет для получения значения. Простейшими выражениями являются литералы и идентификаторы. Другие выражения строятся посредством объединения подвыражений с помощью операторов, выполняющих соответствующие операции.

В качестве операторов используются символы, не являющиеся алфавитно-цифровыми, а также сочетания символов. Список операторов: < > <= >= != == + - \* /

**Ключевые слова:**

Ключевые слова - идентификаторы, зарезервированные для специального использования. Ключевые слова включают только строчные буквы. Их нельзя использовать в качестве обычных идентификаторов.

card[inal][\_u] if else put

double when then to

litera loop until

**Управляющие последовательности:**

Управляющие последовательности это комбинации символов, состоящие из разделительного символа и не графических символов. Они обычно используются для задания каких-либо действий или для литерального представления неграфических символов и символов, имеющих специальное значение. Список управляющих последовательностей:

\n Перевод строки

\r Возврат каретки

\t Горизонтальная табуляция

# Вывод

Были изучены способы определения конечных автоматов (КА) – графового и табличного методов построения недетерминированного КА по системе регулярных выражений. Описан заданный язык программирования и написаны тестовые программы.

# Приложения

Приложение 1. Автомат, реализованного таблично

1. //Построено 19.3.2020 13:21:58 по исходному файлу/шаблону lr1.xml/lexAsTableSyntAsSingleFSM\_to\_js
2. var lCnt = 0;
3. var cCnt = 0;
5. //входной поток литер
6. var TextReader = {
7. text: "^",
8. len: -1,
9. backedChar: 0,
10. pos: 0,
12. setText: function(t){
13. this.text = t;
14. this.len = t.length - 1;
15. this.pos = 0;
16. },
17. //возврат заданной литеры во входной поток
18. back: function(bc){
19. if(this.pos > 0){    //если нет возвращенной, но не прочитанной литеры
20. this.pos = -this.pos;   //устанавливаем признак наличия возвращенной литеры
21. this.backedChar = bc;       //и запоминаем ее код
22. }
23. },
24. //чтение одной литеры
25. read: function() {
26. if(this.pos >= 0){   //если нет возвращенной литеры
27. //и возвращаем -1, если достигнут конец входного потока (или входного потока просто нет), или очередную литеру из него
28. return this.pos <= this.len? this.text[this.pos++]:null;
29. }
30. else    //если есть возвращенная литера
31. {
32. this.pos = -this.pos;   //сбросим признак ее наличия
33. return this.backedChar; //и вернем ее вновь
34. }
35. }
36. };

Приложение 2. Текст программы на описанном языке программирования. Представить натуральное число в виде простых сомножителей или найти сумму вещественных чисел.

F1(){

put N1n to card(input()) //считываем с клавиатуры

loop {

card put I1i to 2

    card put F2f to 0

loop{

when (N1n%I1i == 0) then {

            N1n = N1n / I1i

            print("%d ", i)

            F1f = 1

             break

  }

         other{

            I1i += 1

}

} until 1

when (F1f == 1) then {

    continue

}

} until N1n > 1

print()

}

F2(){

double X1x

double Y1y

double S1um

put S1um to X1x + Y1y

print(S1um)

}

main(){

put A1a to litera(input())

switch (A1a){

case 'a': F1(); break;

case 'b': F2(); break;

default: print("error\n");

}