МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Дальневосточный федеральный университет»**

**ШКОЛА ЕСТЕСТВЕННЫХ НАУК**

**Кафедра прикладной математики, механики, управления и программного обеспечения**

РАЗРАБОТКА ТРАНСЛЯТОРА, ПЕРЕВОДЯЩЕГО ПОДМНОЖЕСТВО ЯЗЫКА JAVA В ЭКВИВАЛЕНТНОЕ ПОДМНОЖЕСТВО ЯЗЫКА GO

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Теория вычислительных процессов и структур»

по образовательной программе подготовки бакалавров по направлению 02.03.03 Математическое обеспечение и администрирование информационных систем

Выполнили студенты гр. Б8117-02.03.03

\_\_\_\_\_\_\_\_\_\_\_ Дейко А.О.

\_\_\_\_\_\_\_\_\_\_\_ Проскурин Д.А.

\_\_\_\_\_\_\_\_\_\_\_ Кокорин И.В..

Руководитель: старший преподаватель кафедры ПММУиПО Лемза А.О.

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

(подпись)

Защищён с оценкой \_\_\_\_\_\_\_\_

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

(подпись) И.О. Фамилия

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

г. Владивосток

2020 г.

**Оглавление**

[Введение 3](#_Toc505816078)

[1 Неформальная постановка задачи 4](#_Toc566121473)

[2 Синтаксис входного языка 9](#_Toc1040330581)

[3 Контекстные условия 14](#_Toc774233868)

[4 Таблица соответствия 17](#_Toc362782087)

[5 Проект лексического анализатора 21](#_Toc972410562)

[5.1 Таблица ключевых слов 22](#_Toc417999283)

[5.2 Типы лексем 24](#_Toc1893352259)

[5.3 Ошибки лексического анализатора 26](#_Toc1865073428)

[5.4 Конечный автомат лексического анализатора 27](#_Toc1300977302)

[6 Проект синтаксического анализатора 28](#_Toc409199050)

[Заключение 32](#_Toc1501547665)

[Список литературы 33](#_Toc1210815587)

# Введение

Программист, написавший программу на каком-либо языке программирования, не сможет сразу запустить свою программу. Одним из способов сделать это является перевод её на язык машины, на которой эта программа будет запущена, иначе говоря, эту программу нужно скомпилировать.

Таким образом компилятор – это программа, которая считывает текст программы, написанной на одном языке (исходном), и транслирует (переводит) его в эквивалентный текст на другом языке (целевом).

***Цель курсового проекта***: разработать проект транслятора, который преобразует программу, содержащую подмножество языка Java в программу, содержащую подмножество языка Go генерируя эквивалентный исходный код. Язык задаётся множеством цепочек, принадлежащих ему;

***Задачи:***

* разработать грамматику подмножества языка Java;
* описать контекстные условия входного языка;
* описать соответствие конструкций входного и выходного языков;
* разработать проект лексического анализатора;
* разработать проект синтаксического анализатора.

# Неформальная постановка задачи

Разработать проект транслятора из подмножества языка «Java» в эквивалентное подмножество языка «Go».

Подмножество языка «Java» включает:

1. Типы данных:***byte ,short, int,long,float,double,char,boolean.***
2. Операторы:

***Арифметические операторы(***Таблица 1***).***

|  |  |  |
| --- | --- | --- |
| Оператор | Описание | Пример |
| + | Складывает значения по обе стороны от оператора | A + B даст 30 |
| - | Вычитает правый операнд из левого операнда | A - B даст -10 |
| \* | Умножает значения по обе стороны от оператора | A \* B даст 200 |
| / | Оператор деления делит левый операнд на правый операнд | B / A даст 2 |
| % | Делит левый операнд на правый операнд и возвращает остаток | B % A даст 0 |
| ++ | Инкремент - увеличивает значение операнда на 1 | B++ даст 21 |
| -- | Декремент - уменьшает значение операнда на 1 | B-- даст 19 |

Таблица 1-Арифметические операторы

***Операторы сравнения(Таблица 2).***

|  |  |  |
| --- | --- | --- |
| Оператор | Описание | Пример |
| == | Проверяет, равны или нет значения двух операндов, если да, то условие становится истинным | (A == B) — не верны |
| != | Проверяет, равны или нет значения двух операндов, если значения не равны, то условие становится истинным | (A != B) — значение истинна |
| > | Проверяет, является ли значение левого операнда больше, чем значение правого операнда, если да, то условие становится истинным | (A > B) — не верны |
| < | Проверяет, является ли значение левого операнда меньше, чем значение правого операнда, если да, то условие становится истинным | (A < B) — значение истинна |
| >= | Проверяет, является ли значение левого операнда больше или равно значению правого операнда, если да, то условие становится истинным | (A >= B) — значение не верны |
| <= | Проверяет, если значение левого операнда меньше или равно значению правого операнда, если да, то условие становится истинным | (A <= B) — значение истинна |

Таблица 2-Операторы сравнения

***Побитовые операторы(Таблица 3).***

|  |  |  |
| --- | --- | --- |
| Оператор | Описание | Пример |
| & (побитовое и) | Бинарный оператор AND копирует бит в результат, если он существует в обоих операндах. | (A & B) даст 12, который является 0000 1100 |
| | (побитовое или) | Бинарный оператор OR копирует бит, если он существует в любом из операндов. | (A | B) даст 61 который равен 0011 1101 |
| ^ (побитовое логическое или) | Бинарный оператор XOR копирует бит, если он установлен в одном операнде, но не в обоих. | (A ^ B) даст 49, которая является 0011 0001 |
| ~ (побитовое дополнение) | Бинарный оператор дополнения и имеет эффект «отражения» бит. | (~ A) даст -61, которая является формой дополнением 1100 0011 в двоичной записи |
| << (сдвиг влево) | Бинарный оператор сдвига влево. Значение левых операндов перемещается влево на количество бит, заданных правым операндом. | A << 2 даст 240, который 1111 0000 |
| >> (сдвиг вправо) | Бинарный оператор сдвига вправо. Значение правых операндов перемещается вправо на количество бит, заданных левых операндом. | A >> 2 даст 15, который является 1111 |
| >>> (нулевой сдвиг вправо) | Нулевой оператор сдвига вправо. Значение левых операндов перемещается вправо на количество бит, заданных правым операндом, а сдвинутые значения заполняются нулями. | A >>> 2 даст 15, который является 0000 1111 |

Таблица 3-Побитовые операторы

***Логическое операторы(Таблица 4).***

|  |  |  |
| --- | --- | --- |
| Оператор | Описание | Пример |
| && | Называется логический оператор «И». Если оба операнда являются не равны нулю, то условие становится истинным | (A && B) — значение false |
| || | Называется логический оператор «ИЛИ». Если любой из двух операндов не равен нулю, то условие становится истинным | (A || B) — значение true |
| ! | Называется логический оператор «НЕ». Использование меняет логическое состояние своего операнда. Если условие имеет значение true, то оператор логического «НЕ» будет делать false | !(A && B) — значение true |

Таблица 4-Логические операторы

***Операторы присваивания(Таблица 5).***

|  |  |  |
| --- | --- | --- |
| Оператор | Описание | Пример |
| = | Простой оператор присваивания, присваивает значения из правой стороны операндов к левому операнду | C = A + B, присвоит значение A + B в C |
| += | Оператор присваивания «Добавления», он присваивает левому операнду значения правого | C += A, эквивалентно C = C + A |
| -= | Оператор присваивания «Вычитания», он вычитает из правого операнда левый операнд | C -= A, эквивалентно C = C - A |
| \*= | Оператор присваивания «Умножение», он умножает правый операнд на левый операнд | C \* = A эквивалентно C = C \* A |
| /= | Оператор присваивания «Деление», он делит левый операнд на правый операнд | C /= A эквивалентно C = C / A |
| %= | Оператор присваивания «Модуль», он принимает модуль, с помощью двух операндов и присваивает его результат левому операнду | C %= A, эквивалентно C = C % A |

Таблица 5-Операторы присваивания

1. Операторы циклов: цикл for, цикл do, цикл while.
2. Операторы ветвления: if else
3. Стандартные функции:

double min( a, b ) – возвращает минимум из a и b

double log( a ) – возвращает натуральный логарифм аргумента a

double pow( a, b ) – возвращает значение a, возведенное в степень b

double sqrt( a ) – возвращает квадратный корень из a

double time() – возвращает текущее значение модельного времени (в единицах модельного времени)

System.out.println(Строка/переменная)– функция вывода в стандартный поток вывода.

# Синтаксис входного языка

G=(N,,P,<Программа>)

N={<Программа>,<Имя класса>,<Главная функция>,<Буква>,<Цифра>,<Знак>,<Строка>,<Предложение>,<Функция вывода>,<Арифметическое выражение>,<Цикл>,<Оператор ветвления>,<Объявить переменную>,<Целый тип>,<Вещественный тип>,<Символьный тип>,<Логический тип>,<Переменные целые>,<Вещественные переменные>,<Символьные переменные>,<Логические переменные>,<Идентификатор>,<Целое число>,<Вещественное число>,<Знак>,<Символ>,<Булево значение>,<Стандартные функции>,<Формула>,<Функция вывода>,<Вывод>,<Арифметическое выражение>,<Операторы присваивания>,<Унарные операторы>,<Часть формулы>,<Арифметические операторы>,<Побитовые операторы>,<Цикл>,<Цикл for>,<Цикл while>,<Цикл do>,<Инициализация счётчика>,<Булево выражение>,<Изменение счётчика>,<Счётчик>,<Выражение счётчика>,<Логические операторы>,<Операторы сравнения>,<Оператор ветвления>}

={A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,1,2,3,4,5,6,7,8,9,0,+,\_,|-,[,],’,;,-,%,/,\*,~,<,>,&,^,|,!,.,=,(,),{,}}

P={

<Программа> public class <Имя класса>{<Главная функция>}

<Имя класса><Буква>|<Буква><Имя класса>

<Буква>A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<Цифра>→1|2|3|4|5|6|7|8|9|0

<Знак>→+|-|

<Строка>→<Цифра>|<Знак>|<Буква>|<Цифра><Строка>|<Знак><Строка>|<Строка>|<Буква><Строка>

<Главная функция>public static void main(String[] args){<Предложение>}

<Предложение><Объявить переменную><Предложение>|<Функция вывода><Предложение>|<Арифметическое выражение><Предложение>|<Цикл><Предложение>|<Оператор ветвления><Предложение>|

<Объявить переменную><Целый тип>| <Вещественный тип>  
<Символьный тип>|<Логический тип>

<Целый тип>byte <Переменные целые>| short <Переменные целые>|int <Переменные целые>|long <Переменные целые>

<Вещественный тип>→float <Вещественные переменные>|double <Вещественные переменные>

<Символьный тип>→char<Символьные переменные>

<Логический тип>→boolean<Логические переменные>

<Переменные целые><идентификатор>;|<идентификатор>,<Переменные целые>|<идентификатор>=<Знак><целое число>;|<идентификатор>=<Знак><целое число>, <Переменные целые>

<Целое число>→<Цифра>|<Цифра><целое число>

<Переменные вещественные>→<идентификатор>;|<идентификатор>,<Переменные вещественные>|<идентификатор>=<Знак><Вещественное число>;|<идентификатор>=<Знак><Вещественное число>, <Переменные вещественные>

<Вещественное число>→<Цифра>|<Цифра><Вещественное число>|<Цифра>.<Вещественное число>

<Переменные символьные>→<идентификатор>;|<идентификатор>,<Переменные символьные>|<идентификатор>=<Символ>;|<идентификатор>=<Символ>, <Переменные символьные>

<Символ>→’<Буква>’|’<Цифра>’|’<Знак>’|<Целое число>

<Логические переменные>→<идентификатор>;|<идентификатор>,<Логические переменные>|<идентификатор>=<Булево значение>;|<идентификатор>=<Булево значение>, <Логические переменные>

<Булево значение>→true|false

<Идентификатор>→<Буква>|<Цифра>|\_<Идентификатор>

<Стандартные функции>→min(<Формула>,<Формула>)|log(<Формула>), pow(<Формула>,<Формула>),sqrt(<Формула>);

<Функция вывода>→System.out.print(<Вывод>);

<Вывод>→<Идентификатор>|<Идентификатор>+<Вывод>|<Строка>|<Стандартные функции>

<Арифметическое выражение>→<Идентификатор><Операторы присваивания><Формула>;|<Унарные операторы><Формула>;|<Формула><Унарные операторы>;

<Формула>→<Часть формулы>|<Часть формулы><Арифметические операторы><Часть формулы>|<Часть формулы><Побитовые операторы><Часть формулы>

<Часть формулы>→<Операнд>|<Унарные операторы><Операнд>|<Операнд><Унарные операторы>|(<Формула>)|<Стандартные функции>|~(<Формула>)

<Операнд>→<Знак><Идентификатор>|<Знак><Вещественное число>|<Знак><Стандартные функции>

<Унарные операторы>→++|--

<Арифметические операторы>→+|-|%|/

<Побитовые операторы>→**<<**|**>>**|**>>>**|&|**|**|^

<Операторы присваивания>→=|-=|+=|%=|/=|\*=

<Цикл>→<Цикл for>|<Цикл while>|<Цикл do>

<Цикл for>→for (<Инициализация счётчика>;<Булево выражение>;<Изменение счётчика>){<Предложение>}|for (<Инициализация счётчика>;<Условие>;<Изменение счётчика>)<Арифметическое выражение>

<Инициализация счётчика>→|<Целый тип>|<Вещественный тип>

<Изменение счётчика>→|<Выражение счётчика>

<Выражение счётчика>→<Счётчик>|<Счётчик><Выражение счётчика>

<Счётчик>→<Идентификатор><Операторы присваивания><Формула>|<Унарные операторы><Формула>|<Формула><Унарные операторы>

<Булево выражение>→<Условие>

<Условие>→!<Булево выражение>|(<Булево выражение>)|<Булево выражение><Логические операторы><Булево выражение>|<Формула><Операторы сравнения><Формула>

<Логические операторы>→&&|**||**|

<Операторы сравнения>→==|!=|>|<|>=|<=

<Цикл while>→While (<Условие>){<Предложение>}

<Цикл do>→do {<Предложение>} while (<Условие>);

<Оператор ветвления>→if (<Условие>) {<Предложение>}|if (<Условие>){<Предложение>}else{<Предложение}|if (<Условие>){<Предложение>}else <Арифметическое выражение>|if(<Условие>)<Арифметическое выражение>|if(<Условие>)<Арифметическое выражение> else {<Предложение>}|if(<Условие>)<Арифметическое выражение> else <Арифметическое выражение>

}

# Контекстные условия

***Контекстные условия о правилах описания идентификаторов в программах:***

1. Идентификатор – это последовательность символов, состоящая из букв, цифр, знака подчёркивания (\_) и знака доллара ($), он не может начинаться с цифры.
2. Идентификатором не может быть зарезервированное слово, а также значения true, false или null.
3. Все используемые в программах идентификаторы должны быть описаны до их использования в программе.
4. Каждый из идентификаторов, используемых в программе, должен быть описан один раз.
5. Если требуется объявить несколько переменных заданного типа, это можно сделать в виде разделенного запятыми списка имен переменных.
6. Переменная может быть инициализирована при объявлении или позже.

***Контекстные условия о правилах использования идентификаторов в своей области действия:***

Две основные области видимости в Java определяются классом и методом, хотя такое их разделение несколько искусственно.

1. Все используемые в программах идентификаторы должны иметь тот же регистр что и при объявлении.
2. Все используемые в программах идентификаторы должны быть объявлены в этой области видимости, либо в более глобальной, включающую эту область видимости.
3. Во внутреннем блоке кода нельзя объявлять переменные с тем же именем, что и во внешней области действия.

***Контекстные условия, определяющие правила соответствия видов величин, входящих в синтаксические конструкции программ:***

1. Соответствие фактических и формальных параметров идёт в порядке перечисления (т.е. первый фактический параметр соответствует первому формальному, второй фактический- второму формальному и т.д).
2. Фактические параметры должны быть совместимы с формальными или быть приводимыми (автоматически или с помощью операторов «кастования»).
3. В операторе присваивания типы переменной и выражения должны совпадать, либо быть приводимыми (автоматически или с помощью операторов «кастования»).
4. В условном операторе и в операторе цикла в качестве условия возможно только логическое выражение.
5. Размер массива обязательно указывается при создании. Размерность массива может быть указана при помощи числа или числовой константы.
6. При создании массива указываются квадратные скобки [] в обеих частях выражения. Их можно указать до названия переменной-ссылки или после.
7. После объявления переменной массива, можно определить сам массив с помощью ключевого слова new с указанием типа и размера.
8. При объявлении массива можно сразу инициализировать массив собственными значениями, когда он объявляется и определить количество элементов. При инициализировании массива собственными значениями, тип элементов из списка значений должен соответствовать объявленному типу массива.
9. Обращение к элементу массива происходит по имени массива, за которым следует значение индекса элемента, заключённого в квадратные скобки. При операции присваивания и т.д типы данных должны соответствовать.

***Контекстные условия, задающие различные количественные ограничения:***

1. При достаточном размере выделенной памяти, массив может иметь длину *Integer.MAX\_VALUE - 8* значений
2. Идентификатор может быть любой длины.
3. Количество идентификаторов не более 1000.
4. Количество параметров в функциях, процедурах не более 1000.
5. Вложенность скобок в выражении не более 5 уровней.

# Таблица соответствия

|  |  |
| --- | --- |
| **Java** | **Go** |
| ***Структура программы*** | |
| public class <Имя Класса> {  public static void main(String[] args) {  [Объявление переменных]  [Тело программы]  } | func main() {  [Объявление переменных]  [Тело программы]  } |
| ***Циклы*** | |
| for ([инициализация счетчика]; [условие]; [изменение счетчика])  {  // действия  }  do{  // действия  }  while ([условие]);  while ([условие]){  // действия  } | for [инициализация счетчика]; [условие]; [изменение счетчика]{  // действия  }  for {; [условие];{  // действия  } |
| ***Оператор ветвления*** | |
| If([условие]){[блок]}  If([условие]){[блок]}else{[блок]} | If [условие] {[блок]}  If [условие] {[блок]}else{[блок]} |
| ***Типы данных*** | |
| Byte  Short  Int  Long  Double  Float  Char  Boolean | Int8  Int16  Int32  Int64  Float32  Float64  String  Bool |
| ***Логический тип данных*** | |
| True  False | True  False |
| ***Логические операции*** | |
| !  &&  || | !  &&  || |
| ***Математические операции*** | |
| +  -  \*  /  %  ++  -- | +  -  \*  /  %  ++  -- |
| ***Поразрядные операции*** | |
| &  |  ^  ~  <<  >>  >>> | &  |  ^  &^  <<  >>  Нет аналога |
| ***Условные выражения*** | |
| ==  !=  <  >  <=  >= | ==  !=  <  >  <=  >= |
| ***Операторы присваивания*** | |
| =  +=  -=  \*=  /=  %= | =  [Идентификатор]=[Идентификатор] +[Выражение]  [Идентификатор]=[Идентификатор] -[Выражение]  [Идентификатор]=[Идентификатор] \*[Выражение]  [Идентификатор]=[Идентификатор] /[Выражение]  [Идентификатор]=[Идентификатор] %[Выражение] |
| ***Встроенные функции языка*** | |
| min( a, b )  log( a )  pow( a, b )  sqrt( a )  time()  System.out.println([Выражение]) | Нет аналога  log( a )  pow( a, b )  sqrt( a )  time.Now()  fmt.Println([Выражение]) |

Таблица 6-Соответствие Java-Go

# Проект лексического анализатора

# Таблица ключевых слов

|  |  |
| --- | --- |
| **Ключевое слово** | **Ключ** |
| break | К1 |
| class | К2 |
| const | К3 |
| continue | К4 |
| do | К5 |
| else | К6 |
| for | К7 |
| goto; | К8 |
| if | К9 |
| static | К10 |
| void | К11 |
| while | К12 |
| public | К13 |
| return | К14 |

Таблица 7-Таблица ключевых слов

|  |  |
| --- | --- |
| **Зарезервированное имя** | **Значение** |
| boolean — имя типа данных | R1 |
| byte — имя типа данных | R2 |
| char — имя типа данных | R3 |
| double — имя типа данных | R4 |
| float — имя типа данных | R5 |
| int — целочисленный тип; | R6 |
| long — целочисленный тип; | R7 |
| short — целочисленный тип; | R8 |
| min– встроенная функция | R9 |
| log – встроенная функция | R10 |
| pow– встроенная функция | R11 |
| sqrt– встроенная функция | R12 |
| time – встроенная функция | R13 |
| System.out.println– встроенная функция | R14 |

Таблица 8-Таблица зарезервированных имён

# Типы лексем

|  |  |
| --- | --- |
| Типы лексем | Ключевое слово,Зарезервированное имя |
| Оператор | +(О1)  -(О2)  \*(О3)  /(О4)  %(О5)  ++(О6)  --(О7)  >(О8)  <(О9)  >=(О10)  <=(О11)  ==(О12)  !=(О13)  &&(О14)  ||(О15)  !(О16)  ~(О17)  =(О18)  +=(О19)  -=(О20)  \*=(О21)  /=(О22)  %=(О23)  <<(О24)  >>(О25)  &(О26)  |(О27)  ^(О28)  >>>(О29) |
| Разделитель | . (D1)  , (D2)  ; (D3)  { (D4)  } (D5)  ( (D6)  ) (D7) |
| Число | Целое, вещественное |
| Строка | Символ (‘ ‘) |
| Булево значение | true, false |

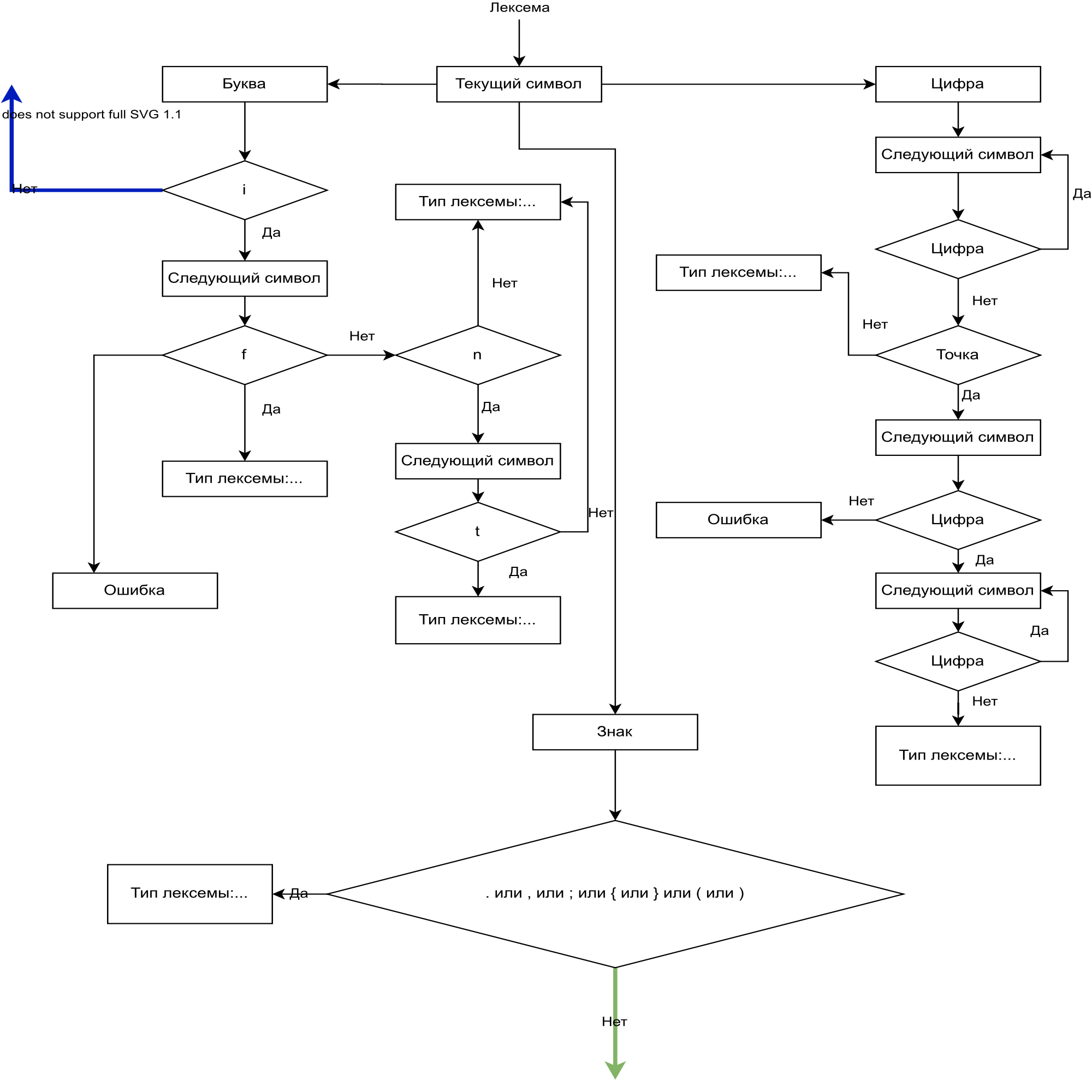
Таблица 9-Типы лексем

# Ошибки лексического анализатора

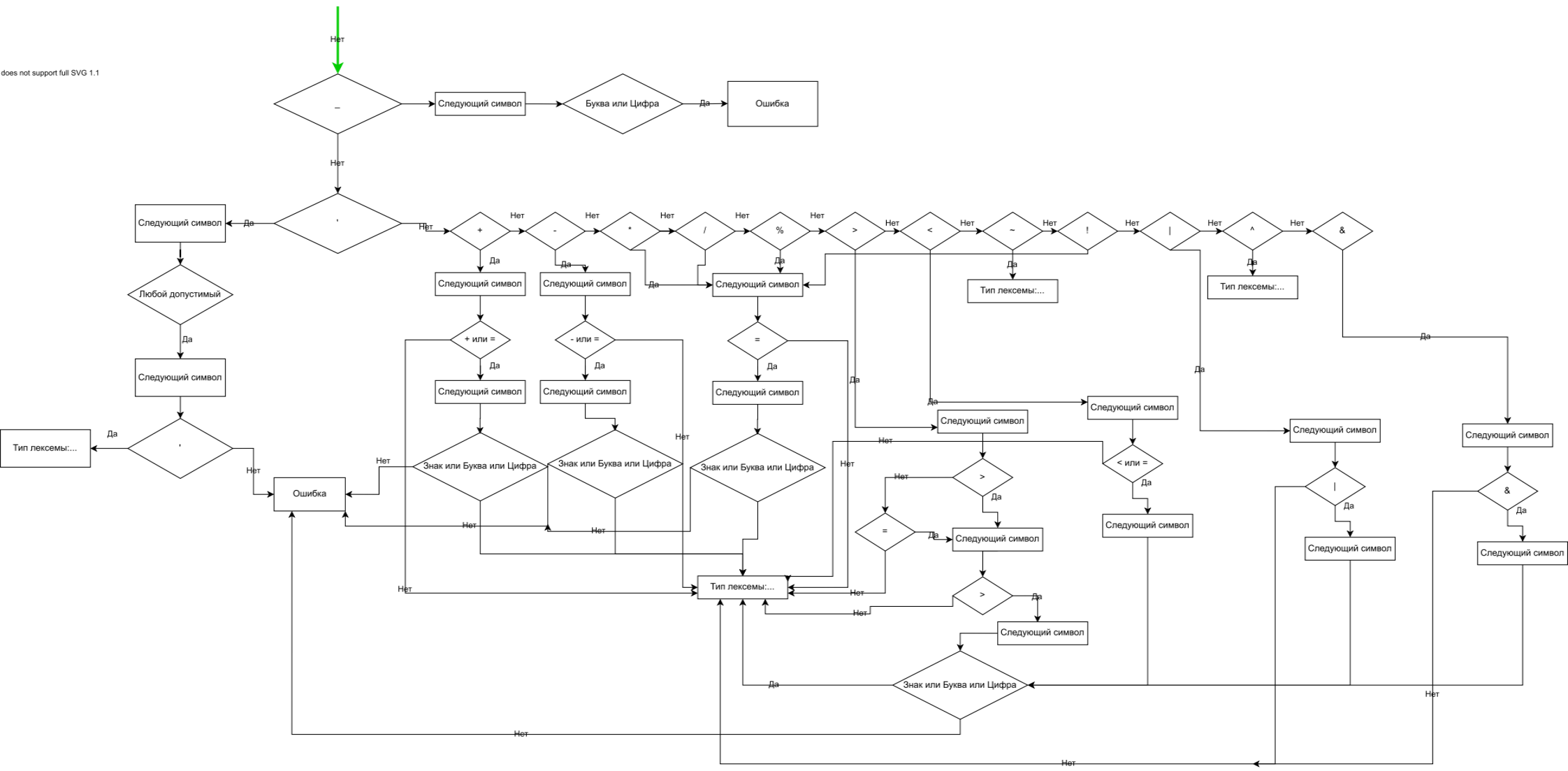
В данном проекте лексический анализатор распознает следующие синтаксические ошибки:

1. Ошибка в образовании вещественного числа;
2. Неправильное формирование строки (символа);
3. Неправильное формирование оператора;
4. Неверное образование имен переменных;
5. Неверное написание служебных слов;

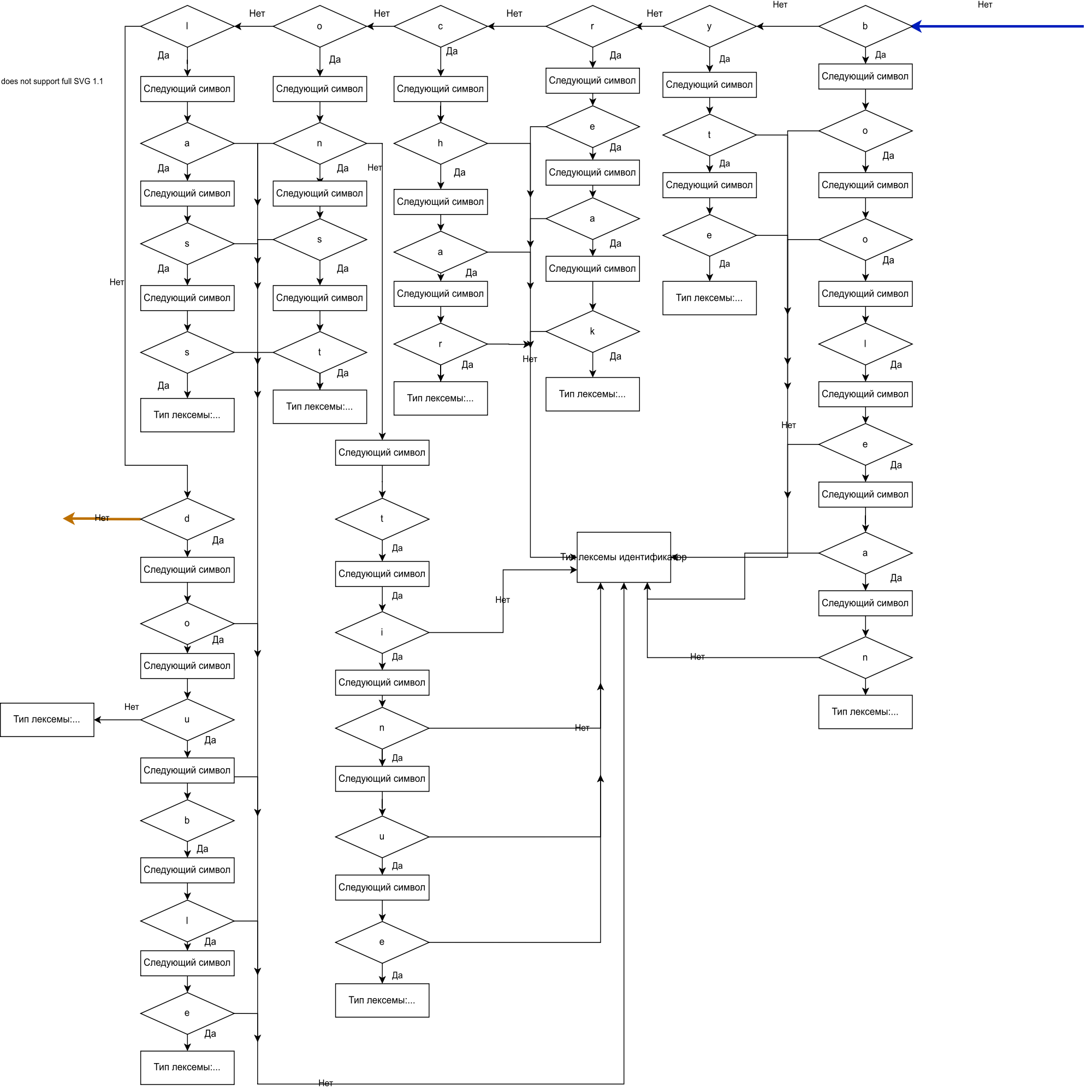
# Конечный автомат лексического анализаторау



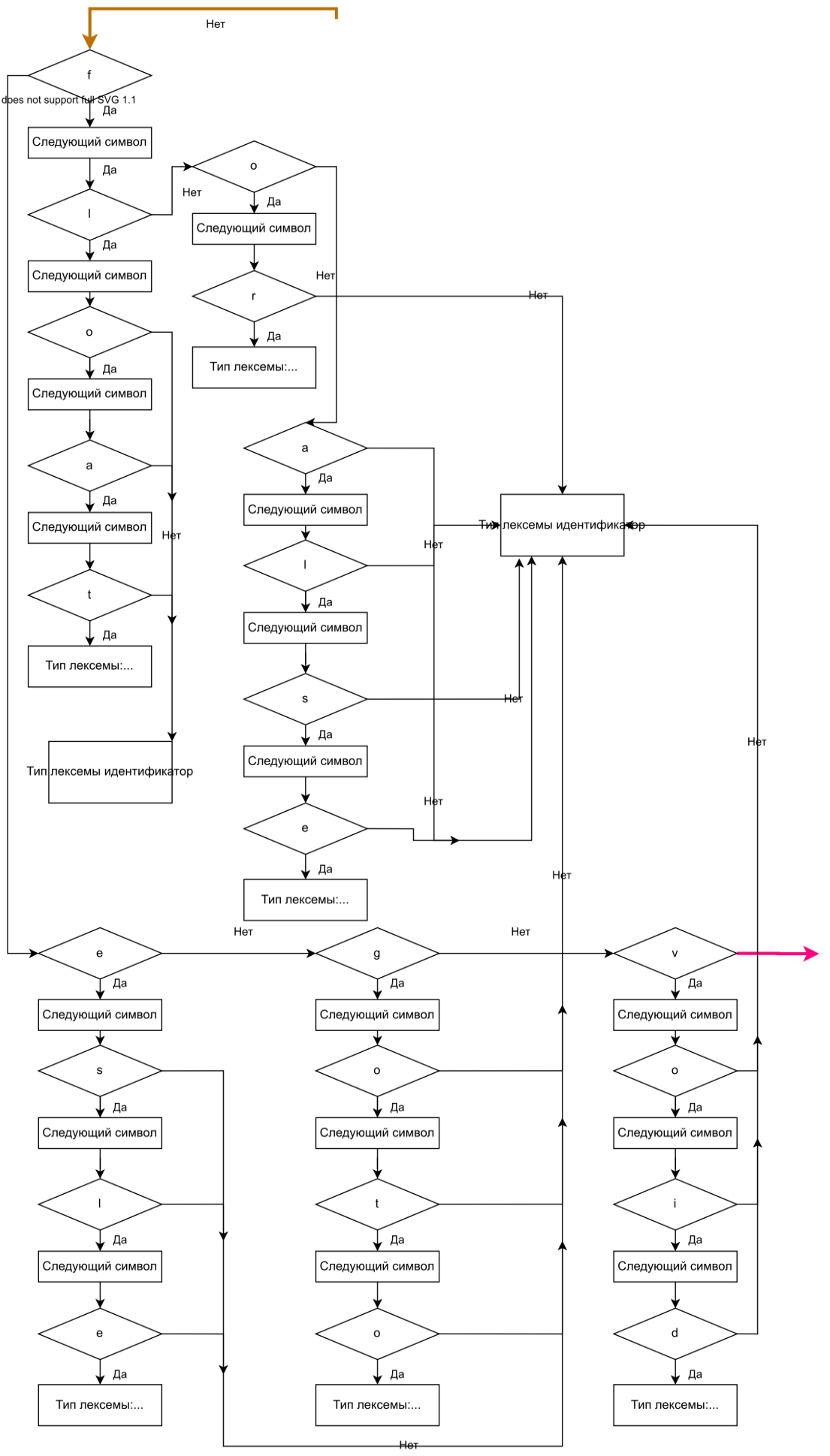
Изображение 1-Конечный автомат 1 часть



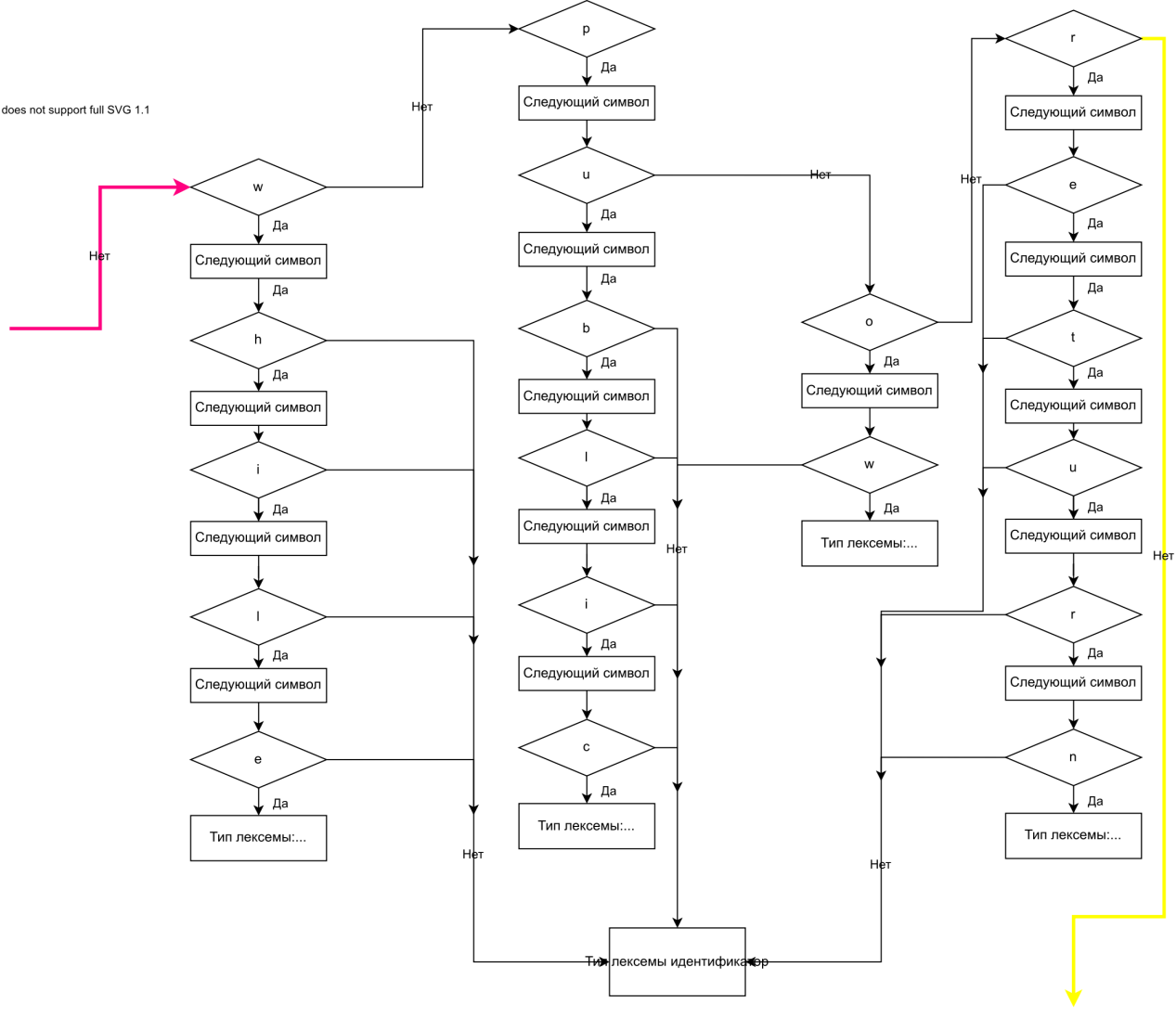
Изображение 2-Конечный автомат 2 часть



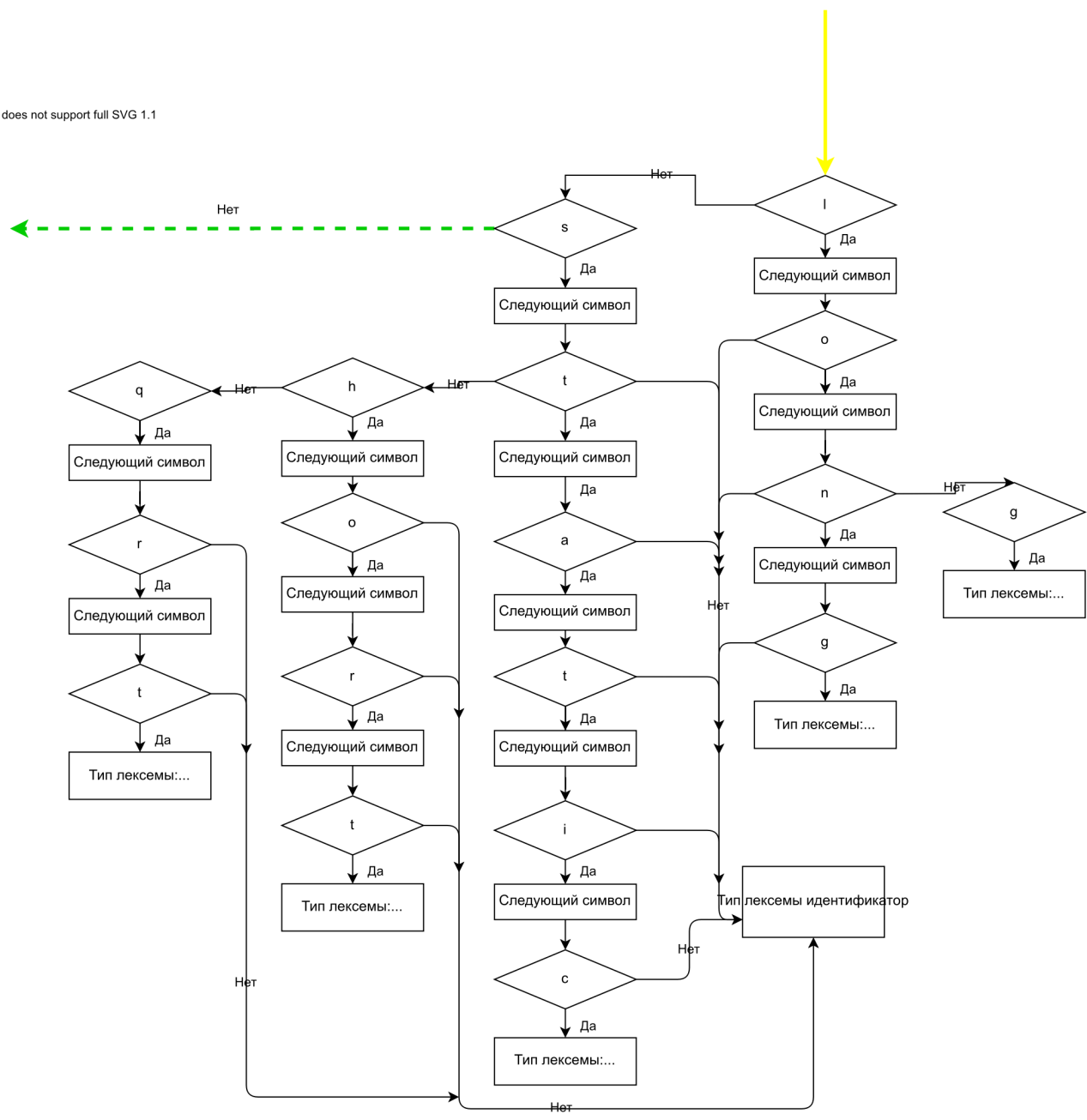
Изображение 3-Конечный автомат 3 часть



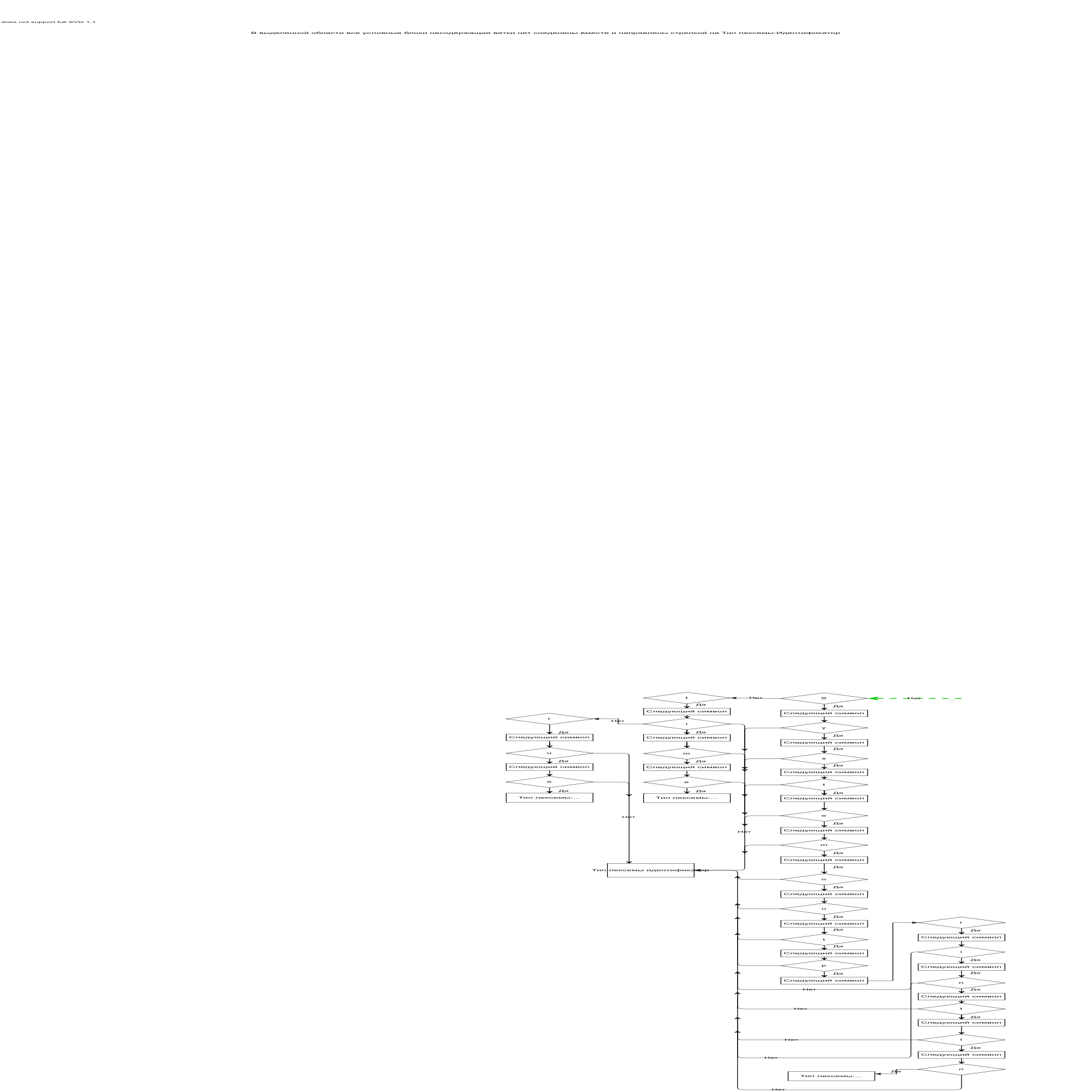
Изображение 4-Конечный автомат 4 часть



Изображение 5-Конечный автомат часть 5



Изображение 6-Конечный автомат 6 часть



Изображение 7-Конечный автомат 7 часть

# Проект синтаксического анализатора

Задача синтаксического анализа – проверка правильности структуры программы, описываемой контекстно-свободной грамматикой. Класс контекстно-свободных грамматик шире класса автоматных грамматик, поэтому и синтаксический анализатор сложнее лексического.

Для синтаксического анализатора был выбран LL(1)-предсказывающий алгоритм нисходящего разбора для LL-грамматик.

**Модель данных**

Язык реализации – C++. Синтаксический анализатор реализован в классе SyntaxAnalyzer. По умолчанию задана грамматика класса Grammar и лексический анализатор класса LexicalAnalyzer. Алгоритм разбора реализован в классе LLAlgorithm. Основной метод класса – разбор цепочки – получает на вход цепочку символов, результатом работы метода является последовательность номеров правил – левый разбор входной цепочки.

Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 1 – Диаграмма потока данных

**Класс SyntaxAnalyzer**

getParsed(inputText) // метод возвращает цепочку правил – левый вывод цепочки inputText.

**Класс Grammar**

getTerminal() // метод возвращает множество терминальных символов грамматики.

getNonterminal() // метод возвращает множество нетерминальных символов грамматики.

getAxiom() // метод возвращает начальный символ грамматики.

getRules() // метод возвращает множество правил вывода грамматики.

setTerminal(newTerminal) // метод устанавливает множество терминальных символов грамматики.

setNonterminal(newNonterminal) // метод устанавливает нетерминальных символов грамматики.

setAxiom(newAxiom) // метод устанавливает начальный символ грамматики.

setRules(newRules) // метод устанавливает множество правил вывода грамматики.

**Класс LexicalAnalyzer**

setStateMachine(stateMachine) // метод устанавливает конечный автомат лексического анализатора.

getTokenList(inputText) // метод возвращает множество токенов входной цепочки.

**Класс LLAlgorithm**

setGrammar(inputGrammar) // метод устанавливает грамматику с которой работает алгоритм.

getParsed(inputText) // метод возвращает цепочку правил – левый вывод цепочки inputText.

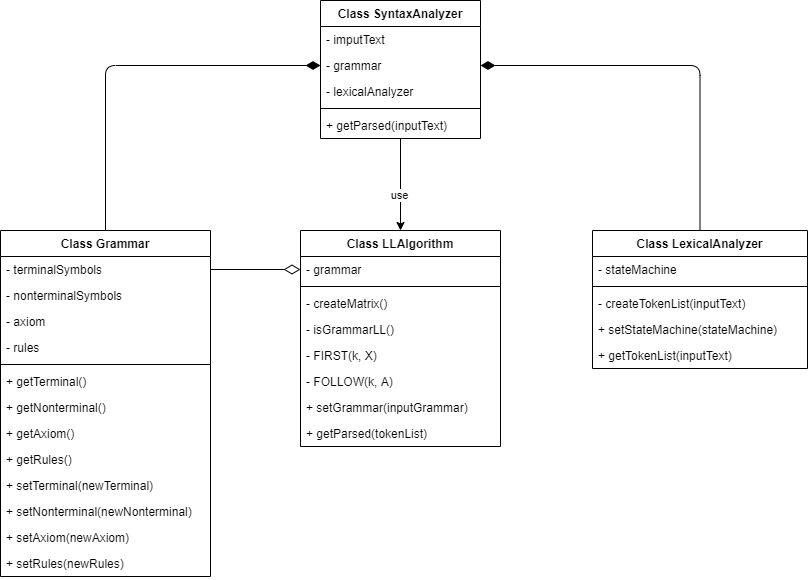


Рисунок 2 – Диаграмма классов синтаксического анализатора

**Проект LL(1)-предсказывающего алгоритма**

Изображение выглядит как текст, карта

Автоматически созданное описание

Рисунок 3 - Проект LL(1)-предсказывающего алгоритма

# Заключение

1. В ходе курсового проекта были решены следующие задачи:
2. Сформулирована неформальная постановка задачи.
3. Разработан синтаксис входного языка.
4. Описаны контекстные условия.
5. Создан проект лексического анализатора, включающий в себя таблицы соответствий, ключевых и зарезервированных слов, типов лексем, конечный автомат.
6. Создан проект синтаксического анализатора, включающий в себя модель данных, архитектуру классов синтаксического анализатора, проект алгоритма синтаксического анализа.

# Список литературы

1. Хорстманн.К.С., Java. Библиотека профессионала, том 1 Основы 10-е изд.: Пер. с англ. — М .: ООО "И.Д. Вильямс", 2014. — 864 с.
2. Донован, А.А., Керниган.Б.У., Язык программирования Go. Пер. с англ. — М.: ООО “И.Д. Вильямс”, 2016. — 432 с.
3. Ахо А., Ульман Д., Теория синтаксического анализа, перевода и компиляции. Синтаксический анализ: Том 1. М.: Мир, 1978 - 384 с.
4. Серебряков В.А., Галочкин.М.П., Основы конструирования компиляторов. М.: УРСС, 2001—192 с. Режим доступа: https://ru.scribd.com/doc/49153216(дата обращения: 30.01.2020)
5. Грис.Д., Конструирование компиляторов для цифровых вычислительных машин. М.: Мир, 1975 -575с.