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](#_Toc45968464)

[1 Неформальная постановка задачи 4](#_Toc45968465)

[2 Синтаксис входного языка 9](#_Toc45968466)

[3 Контекстные условия 15](#_Toc45968467)

[4 Таблица соответствия 17](#_Toc45968468)

[5 Проект лексического анализатора 21](#_Toc45968469)

[5.1 Таблица ключевых слов 21](#_Toc45968470)

[5.2 Типы лексем 23](#_Toc45968471)

[5.3 Ошибки лексического анализатора 24](#_Toc45968472)

[5.4 Конечный автомат лексического анализатора 25](#_Toc45968473)

[6 Проект синтаксического анализатора 31](#_Toc45968474)

[Заключение 42](#_Toc45968475)

[Список литературы 43](#_Toc45968476)

# 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.

1. 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*** |
| + | Складывает значения по обе стороны от оператора |
| - | Вычитает правый операнд из левого операнда |
| \* | Умножает значения по обе стороны от оператора |
| / | Оператор деления делит левый операнд на правый операнд |
| % | Делит левый операнд на правый операнд и возвращает остаток |
| ++ | Инкремент - увеличивает значение операнда на 1 |
| -- | Декремент - уменьшает значение операнда на 1 |

Таблица -Арифметические операторы

***Операторы сравнения (Таблица 2).***

|  |  |
| --- | --- |
| ***Оператор*** | ***Описание*** |
| == | Проверяет, равны или нет значения двух операндов, если да, то условие становится истинным |
| != | Проверяет, равны или нет значения двух операндов, если значения не равны, то условие становится истинным |
| > | Проверяет, является ли значение левого операнда больше, чем значение правого операнда, если да, то условие становится истинным |
| < | Проверяет, является ли значение левого операнда меньше, чем значение правого операнда, если да, то условие становится истинным |
| >= | Проверяет, является ли значение левого операнда больше или равно значению правого операнда, если да, то условие становится истинным |
| <= | Проверяет, если значение левого операнда меньше или равно значению правого операнда, если да, то условие становится истинным |

Таблица -Операторы сравнения

***Побитовые операторы (Таблица 3).***

|  |  |
| --- | --- |
| ***Оператор*** | ***Описание*** |
| & (побитовое и) | Бинарный оператор AND копирует бит в результат, если он существует в обоих операндах. |
| | (побитовое или) | Бинарный оператор OR копирует бит, если он существует в любом из операндов. |
| ^ (побитовое логическое или) | Бинарный оператор XOR копирует бит, если он установлен в одном операнде, но не в обоих. |
| ~ (побитовое дополнение) | Бинарный оператор дополнения и имеет эффект «отражения» бит. |
| << (сдвиг влево) | Бинарный оператор сдвига влево. Значение левых операндов перемещается влево на количество бит, заданных правым операндом. |
| >> (сдвиг вправо) | Бинарный оператор сдвига вправо. Значение правых операндов перемещается вправо на количество бит, заданных левых операндом. |
| >>> (нулевой сдвиг вправо) | Нулевой оператор сдвига вправо. Значение левых операндов перемещается вправо на количество бит, заданных правым операндом, а сдвинутые значения заполняются нулями. |

Таблица -Побитовые операторы

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

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

Таблица -Логические операторы

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

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

Таблица -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.

1. 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>,<ЦифрИд>,<Целое число>,<Вещественное число>,<Знак>,<Символ>,<Булево значение>,<Стандартные функции>,<Формула>,<Функция вывода>,<Вывод>,<Присваивание>,<Операторы присваивания>,<Унарные операторы>,<Часть формулы>,<Арифметические операторы>,<Цикл>,<Цикл for>,<Цикл while>,<Цикл do>,<Инициализация счётчика>,<Булево выражение>,<Условие>,<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 <Идентификатор>{<Главная функция>}

<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>|<Буква><Строка>

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

<Предложение><Объявить переменную >| <Функция вывода> |<Присваивание> | <Цикл> | <Оператор ветвления> | <Оператор останова>

<Список предложений><Предложение>; | <Предложение> ;<Список предложений>

<Объявить переменную> <Объявить одну переменную |<Объявить одну переменную>, <Объявить переменную>

<Объявить одну переменную> <Имя типа><Идентификатор | <Имя типа><Идентификатор> = <Выражение>

<Имя типа> byte|short|int|long|float|double|char|boolean|String

<Число>→<Вещественное число>|<Целое число>

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

<Вещественное число>→<Целое число>.<Целое число>

<Булево значение>→true|false

<Идентификатор>→\_<Символы ид-ра>|<Буква>< Символы ид-ра >|<Буква>

<Символы ид-ра> → <Цифра> | <Буква> | \_ |<Цифра><Символы ид-ра> | <Буква><Символы ид-ра> | \_ <Символы ид-ра>

<Стандартные функции>→Math.log(<Аргументы>)| Math.pow(<Аргументы>)| Math.sqrt(<Аргументы>)

<Аргументы>→<Выражение>,<Аргументы>|<Выражение>

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

<Присваивание>→<Идентификатор>=<Выражение>;|  
<Выражение><Унарные операторы>;

<Унарные операторы>→++|--

<Выражение>→<Арифметическое выражение>|<Логическое выражение>|<Символьное выражение>

<Знак операции символьного сложения>→+

<Знак операции символьного умножения>→\*

<Символьное выражение> →  
<Символьное слагаемое><Знак операции символьного сложения><Символьное выражение>|  
<Символьное слагаемое>

<Символьное слагаемое> →   
<Символьный множитель><Знак операции символьного умножения><Символьное слагаемое>|  
<Символьный множитель>

<Символьный множитель> → (<Символьное выражение >)|<Идентификатор>|‘<Строка>’ | “<Строка>”

<Знак операции типа сложение>→+|-

<Знак операции типа умножение>→\*|/|%

<Арифметическое выражение> →  
<Арифметическое слагаемое><Знак операции типа сложение><Арифметический выражение>|  
<Арифметическое слагаемое>

<Арифметическое слагаемое> →   
<Арифметический множитель><Знак операции типа умножение><Арифметическое слагаемое>|  
<Арифметический множитель>

<Арифметический множитель> → (<Арифметическое выражение >)|<Идентификатор>|<Число>|<Стандартные функции>

<Знак операции логического сложения> → ||

<Знак операции логического умножения> → &&

<Знак операции сравнения >→==|!=|>|<|>=|<=

<Логическое выражение> →  
<Логическое слагаемое><Знак операции логического сложения><Логическое выражение>|  
<Логическое слагаемое>

<Логическое слагаемое> →   
<Логический множитель><Знак операции логического умножения><Логическое слагаемое>|  
<Логический множитель>

<Логический множитель> →   
<Арифметическое выражение><Знак операции сравнения>< Арифметическое выражение>|  
<Логическое выражение><Знак операции сравнения>< Логическое выражение>|  
(<Логическое выражение>)|  
<Идентификатор>|  
<Булево значение>

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

<Цикл for>→for (<Инициализация счётчика>;<Условие>;<Изменение счётчика>){<Список предложений>}|for (<Инициализация счётчика>;<Условие>;<Изменение счётчика>)< Предложение >

<Инициализация счётчика>→|<Объявить переменную>|<Присваивание>

<Изменение счётчика>→|<Присваивание>

<Условие>→<Логическое выражение>

<Цикл while>→While (<Условие>){< Список предложений >}

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

<Оператор ветвления>→if (<Условие>) {< Список предложений >}|if (<Условие>){< Список предложений >}else{< Список предложений >}|if (<Условие>){< Список предложений >}else <Предложение>|if(<Условие>)<Предложение>|if(<Условие>)<Предложение> else {< Список предложений>}|if(<Условие>)<Предложение> else <Предложение>}

<Операторы останова>→break|continue

1. Контекстные условия

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

1. Все используемые в программах идентификаторы должны быть описаны до их использования в программе.
2. Каждый из идентификаторов, используемых в программе, должен быть описан один раз.

***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. Размер массива обязательно указывается при создании. Размерность массива может быть указана при помощи числа или числовой константы.
5. При инициализировании массива собственными значениями, тип элементов из списка значений должен соответствовать объявленному типу массива.
6. 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.
6. Таблица соответствия

В данной главе представлена таблица соответствия (Таблица 6).

|  |  |
| --- | --- |
| **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]} |

Таблица -Таблица соответствия Java-Go

Продолжение таблицы 6 соответствия Java-Go

|  |  |  |
| --- | --- | --- |
| **Java** | **Go** | |
| ***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*** | | |
| +  -  \*  /  %  ++  -- | | +  -  \*  /  %  ++  -- |

Продолжение таблицы 6 соответствия Java-Go

|  |  |  |
| --- | --- | --- |
| **Java** | **Go** | |
| ***Поразрядные операции*** | | |
| &  |  ^  ~  <<  >>  >>> | | &  |  ^  &^  <<  >>  Нет аналога |
| ***Условные выражения*** | | |
| ==  !=  <  >  <=  >= | | ==  !=  <  >  <=  >= |
| ***Операторы присваивания*** | | |
| =  +=  -=  \*=  /=  %= | | =  [Идентификатор]=[Идентификатор] +[Выражение]  [Идентификатор]=[Идентификатор] -[Выражение]  [Идентификатор]=[Идентификатор] \*[Выражение]  [Идентификатор]=[Идентификатор] /[Выражение]  [Идентификатор]=[Идентификатор] %[Выражение] |

Окончание таблицы соответствия Java-Go

|  |  |  |
| --- | --- | --- |
| **Java** | **Go** | |
| ***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]) |

1. Проект лексического анализатора

# Таблица ключевых слов

Ниже представлена таблица ключевых слов (Таблица 7).

|  |  |
| --- | --- |
| ***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 -Таблица ключевых слов

Ниже представлена таблица зарезервированных имён (Таблица 8).

|  |  |
| --- | --- |
| ***Зарезервированное имя*** | ***Значение*** |
| boolean | R1 (имя типа данных) |
| byte — | R2 (имя типа данных) |
| char | R3 (имя типа данных) |
| double | R4 (имя типа данных) |
| float | R5 (имя типа данных) |

Таблица -Таблица зарезервированных имён

Окончание таблицы 8 зарезервированных имён

|  |  |
| --- | --- |
| ***Зарезервированное имя*** | ***Значение*** |
| int | R6 (целочисленный тип;) |
| long | R7 (целочисленный тип;) |
| short | R8 (целочисленный тип;) |
| min | R9 (встроенная функция) |
| log | R10 (встроенная функция) |
| pow | R11 (встроенная функция) |
| sqrt | R12 (встроенная функция) |
| time | R13 (встроенная функция) |
| System.out.println | R14 (встроенная функция) |
| True | R15 (логический тип) |
| False | R16 (логический тип) |

# Типы лексем

Ниже представлена таблица типов лексем (Таблица 9).

|  |  |
| --- | --- |
| Идентификатор | 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) |

Таблица -Типы лексем

Окончание таблицы 9 Типы лексем

|  |  |
| --- | --- |
|  | %=(О23) |
| <<(О24) |
| >>(О25) |
| &(О26) |
| |(О27) |
| ^(О28) |
| >>>(О29) |
| Разделитель | . (D1) |
| , (D2) |
| ; (D3) |
| { (D4) |
| } (D5) |
| ( (D6) |
| ) (D7) |
| Число | Целое, вещественное |
| Строка | Символ (‘ ‘) |

# Ошибки лексического анализатора

В данном проекте лексический анализатор распознает следующие синтаксические ошибки:

1. Ошибка в образовании вещественного числа;
2. Неправильное формирование символа;
3. Неправильное формирование оператора;
4. Неверное образование имен переменных;
5. Недопустимый грамматикой символ;

# Конечный автомат лексического анализатора

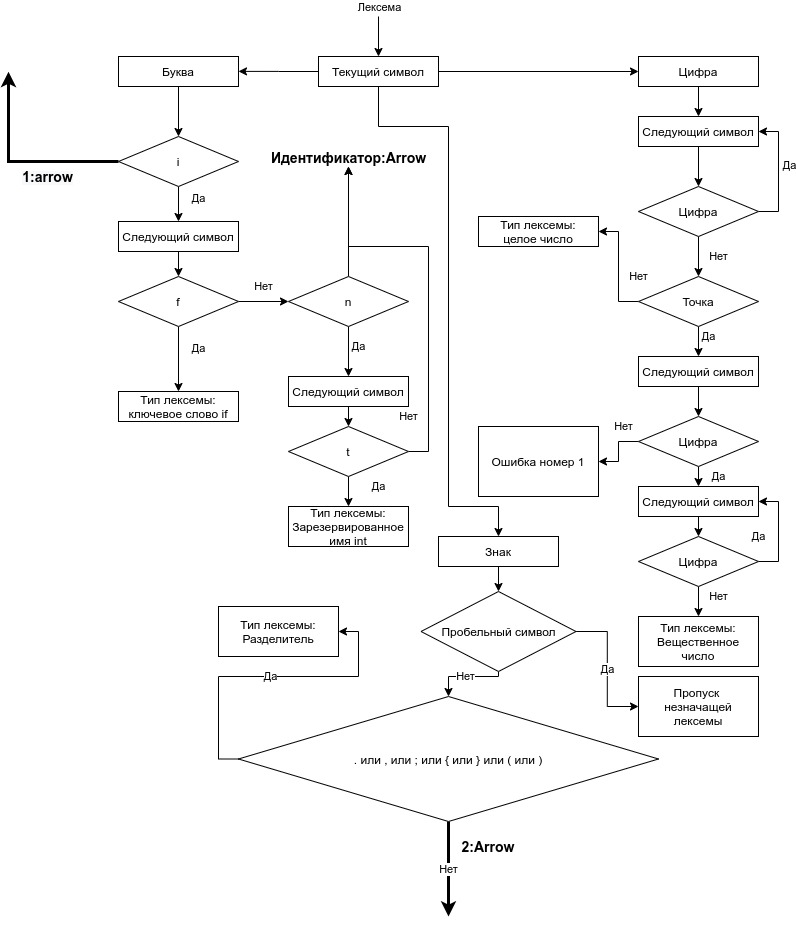


Рисунок - Конечный автомат 1 часть

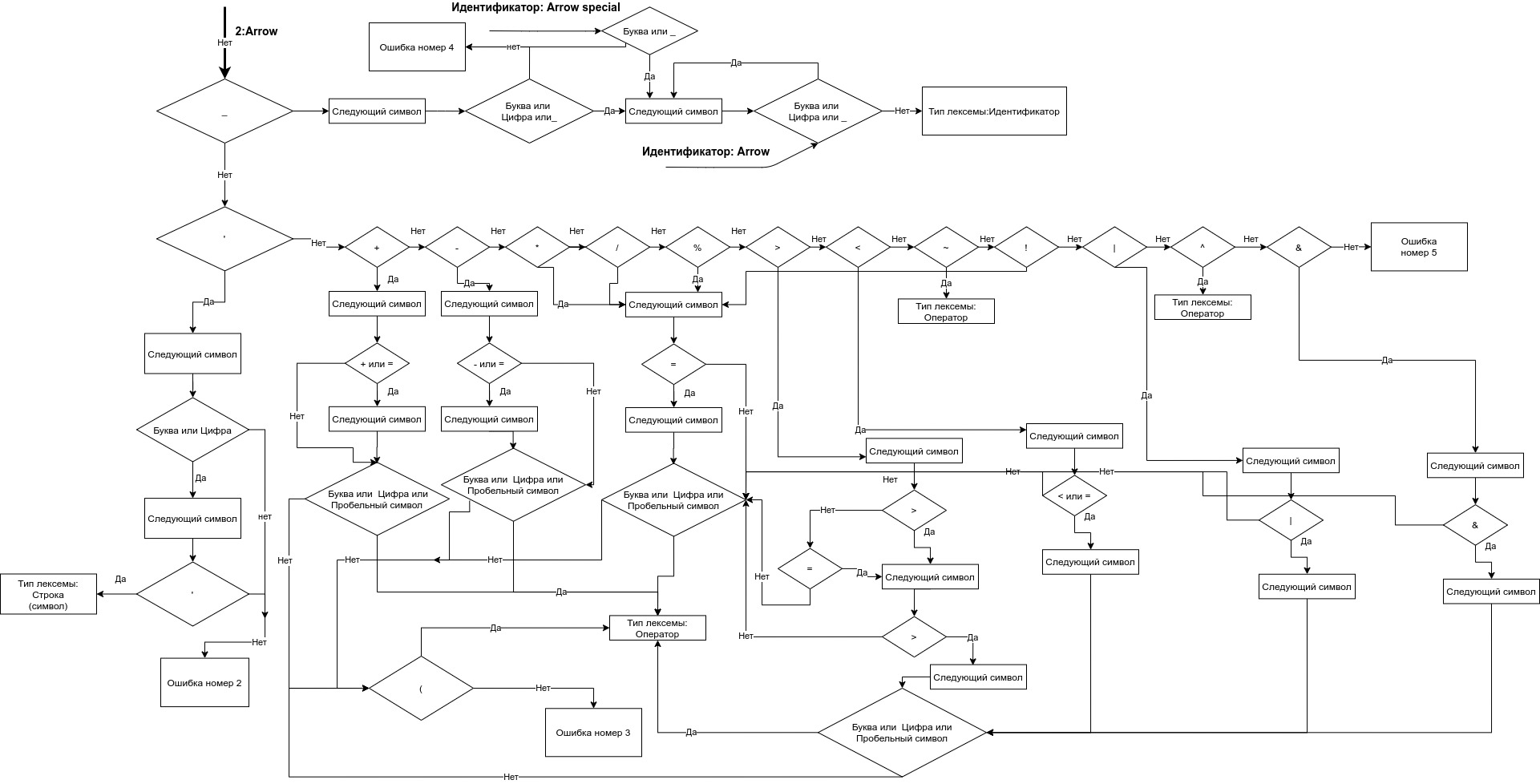


Рисунок -Конечный автомат 2 часть

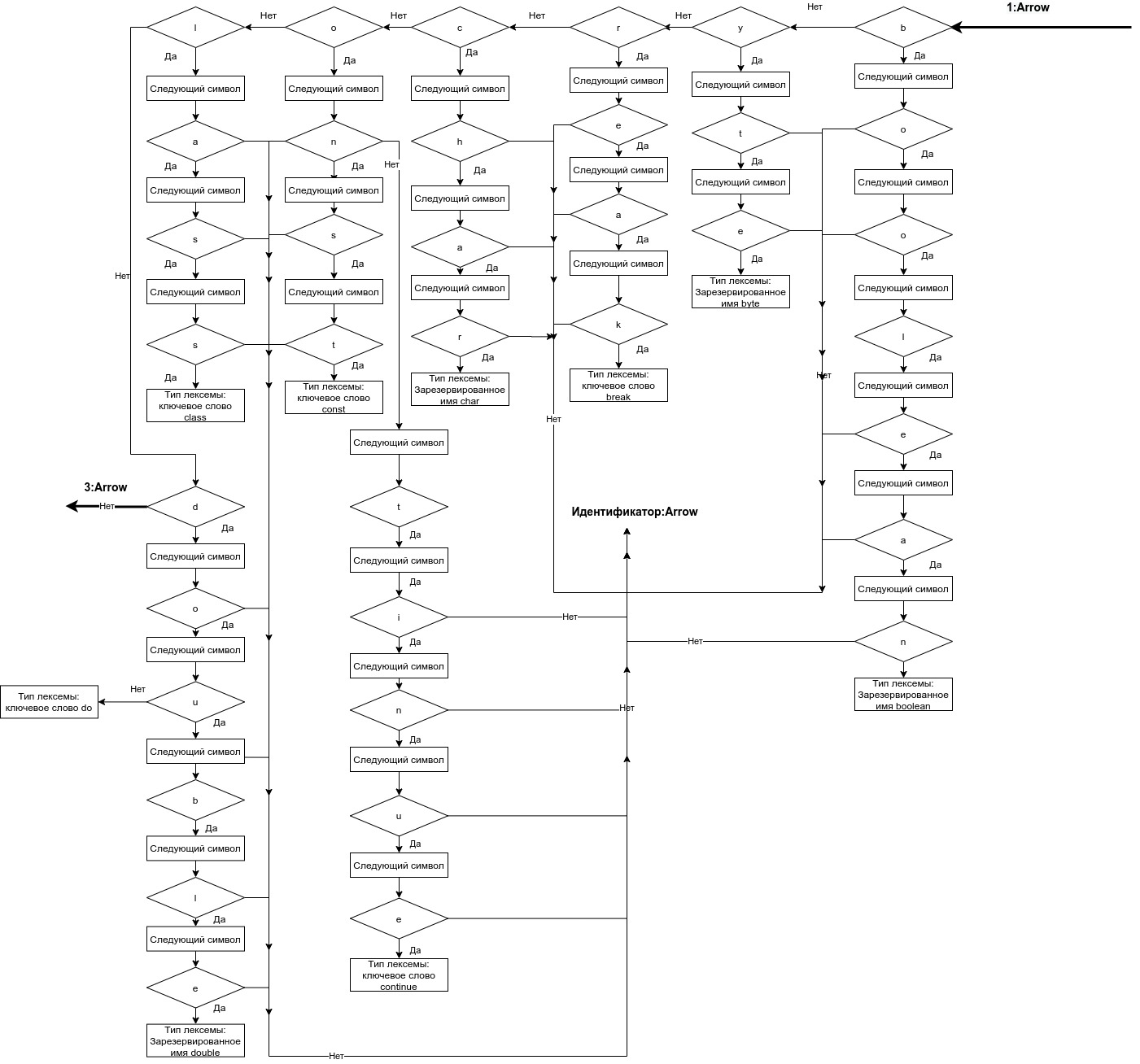


Рисунок -Конечный автомат 3 часть

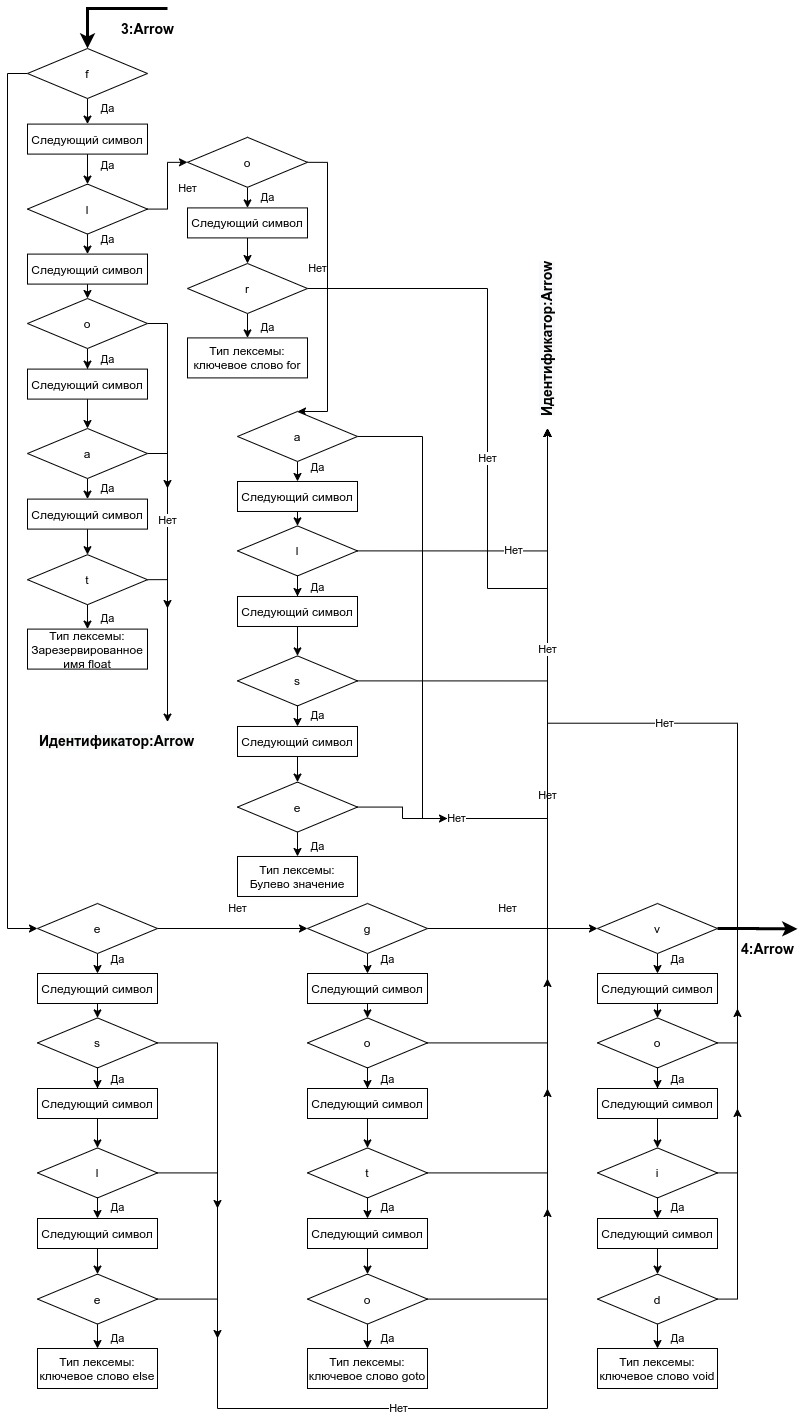


Рисунок -Конечный автомат 4 часть

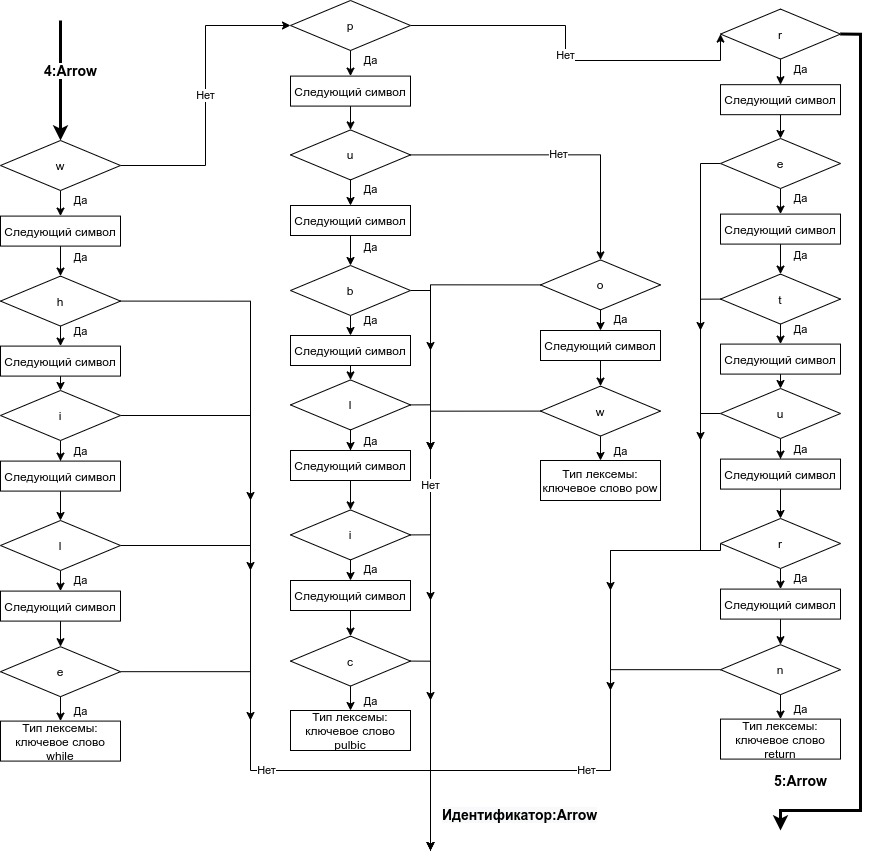


Рисунок -Конечный автомат часть 5

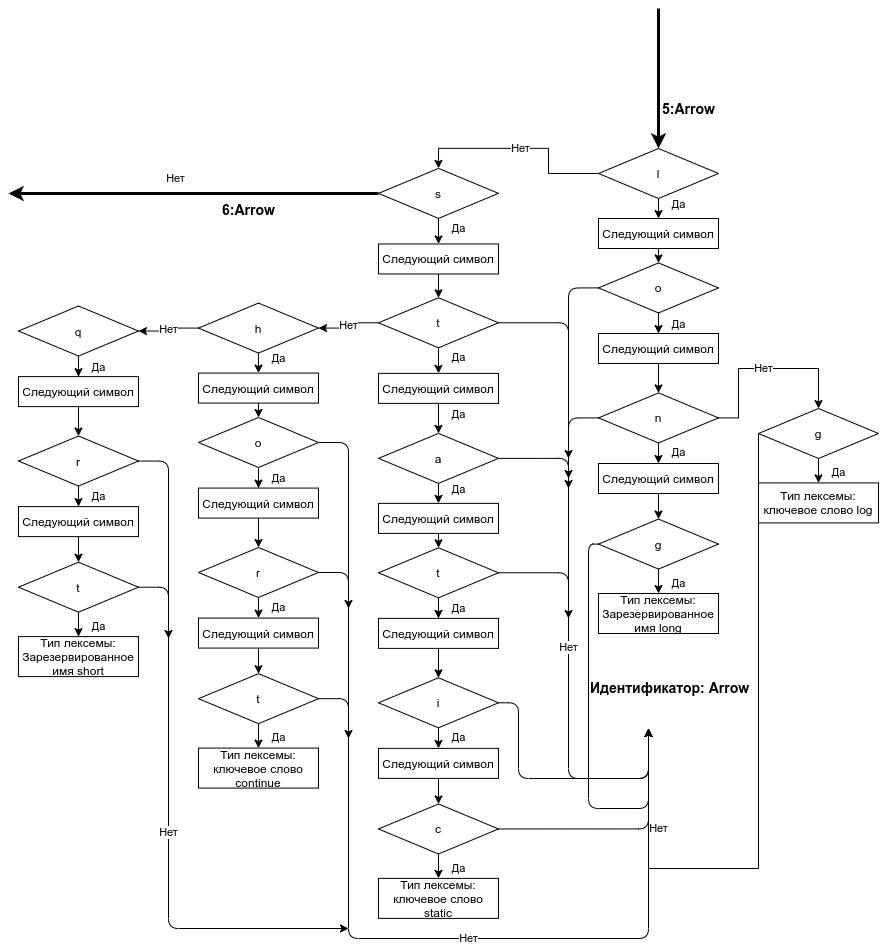


Рисунок -Конечный автомат 6 часть

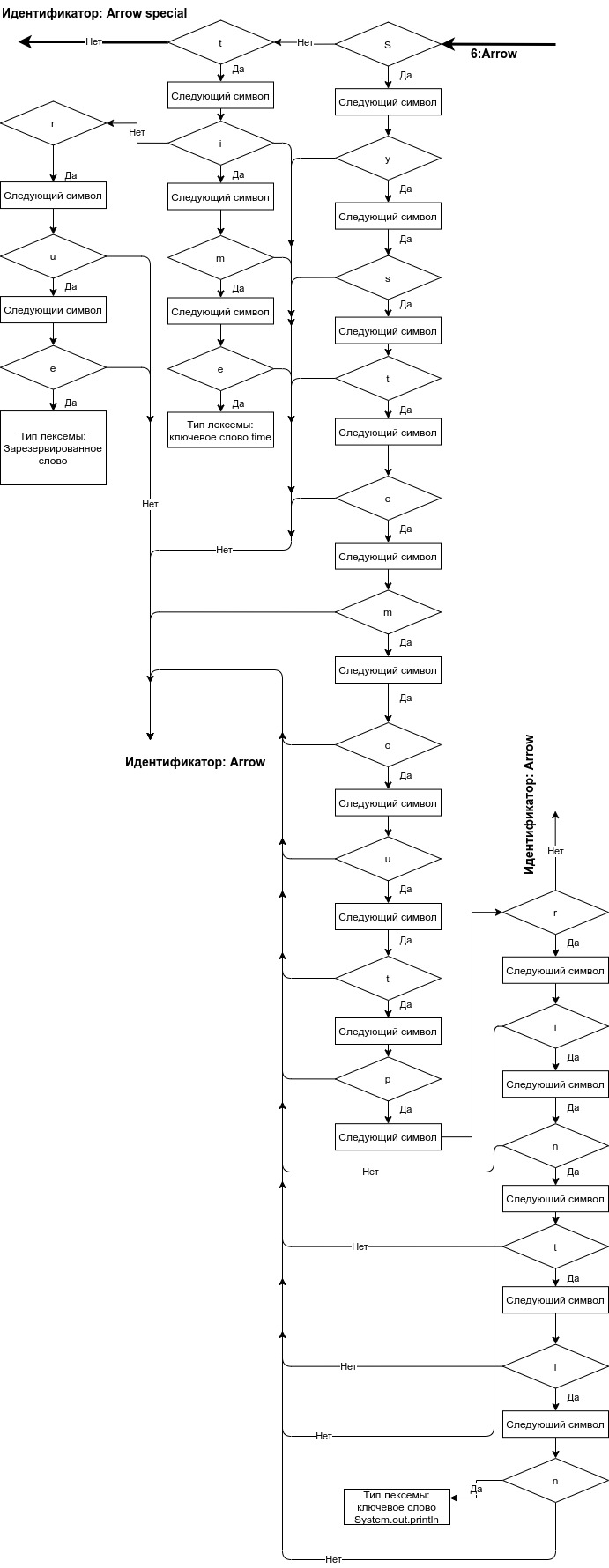


Рисунок -Конечный автомат 7 часть

1. 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 алгоритм Эрли восходящего разбора.

**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 EarleyAlgorithm. Основной метод класса – разбор цепочки – получает на вход цепочку символов, результатом работы метода является последовательность номеров правил – правый вывод входной цепочки.

Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок – Диаграмма потока данных

**Класс SyntaxAnalyzer**

getParsed(inputText) // метод возвращает цепочку правил – правый 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.

getRuleNumber(rule) // метод возвращает номер правила.

**Класс LexicalAnalyzer**

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

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

**Класс EarleyAlgorithm**

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

isGrammarWithoutCycles() // метод проверяет грамматику на отсутствие циклов.

createParseList(tokenList) // метод возвращает список разбора входной цепочки.

getParseTree(parseList, tokenList) // метод строит правый вывод входной цепочки по списку разбора.

getParsed(tokenList) // метод возвращает правый вывод входной цепочки.

**Класс State**

setSituation(rule, dotAfterPos, index) // метод добавляет ситуацию с правилом rule, положением точки после dotAfterPos позиции, с индексом index.

getSituation(situationIndex) // метод возвращает ситуацию с индексом situationIndex.

getSituations() // метод возвращает список всех ситуаций.

getSize() // метод возвращает количество ситуаций в состоянии.

getDotAtEnd() // метод возвращает список всех ситуаций с точкой на конце.

getDotBefore(symbol) // метод возвращает список всех ситуаций с точкой перед символом symbol.

getDotBeforeNonterm() / метод возвращает список всех ситуаций с точкой перед нетерминальным символом.

**Класс Situation**

getDotAfter() // метод возвращает позицию точки в ситуации.

getIndex() // метод возвращает индекс ситуации.

getRule() // метод возвращает правило вывода ситуации.

setDotAfter(position) // метод позицию точки в ситуации.

setIndex(index) // метод устанавливает индекс ситуации.

setRule(rule) // метод устанавливает правило вывода ситуации.

isDotAtEnd() // метод возвращает true, если точка находится на конце.

isDotBeforeNonterm() // метод возвращает true, если точка находится перед нетерминальным символом.

isDotBefore(symbol) // метод возвращает true, если точка находится перед символом symbol.

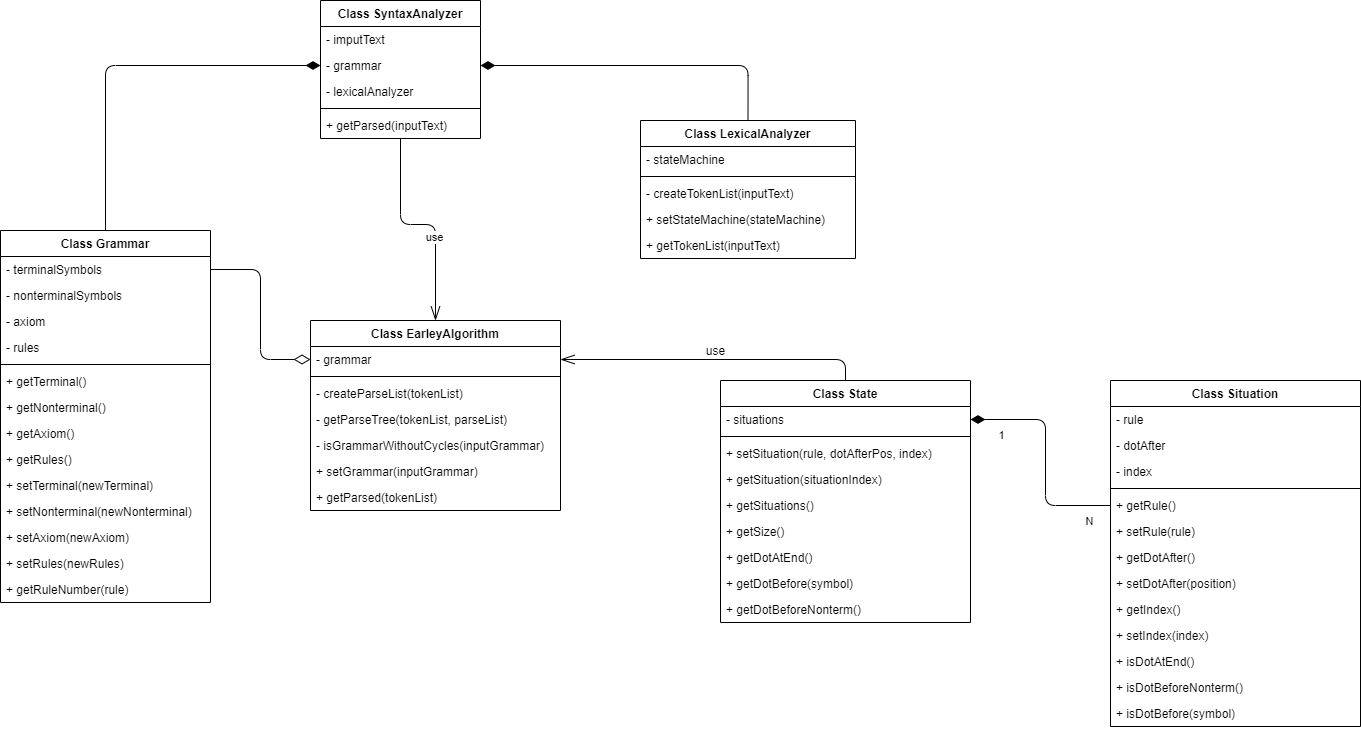


Рисунок – Диаграмма классов синтаксического анализатора

**Проект алгоритма Эрли**

Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок – Проект алгоритма разбора

Изображение выглядит как текст

Автоматически созданное описание

Рисунок – Проект алгоритма составления списка разбора

Изображение выглядит как текст, карта

Автоматически созданное описание

Рисунок – Проект алгоритма построения разбора

Изображение выглядит как текст, карта

Автоматически созданное описание

Рисунок – Проект процедуры R(правый разбор)

**Пример дерева разбора для следующей входной цепочки**

public class Main{

public static void main(String[] args){

System.out.print(“Hello”);

}

}

Пример дерева её разбора приведён на рисунке 7:

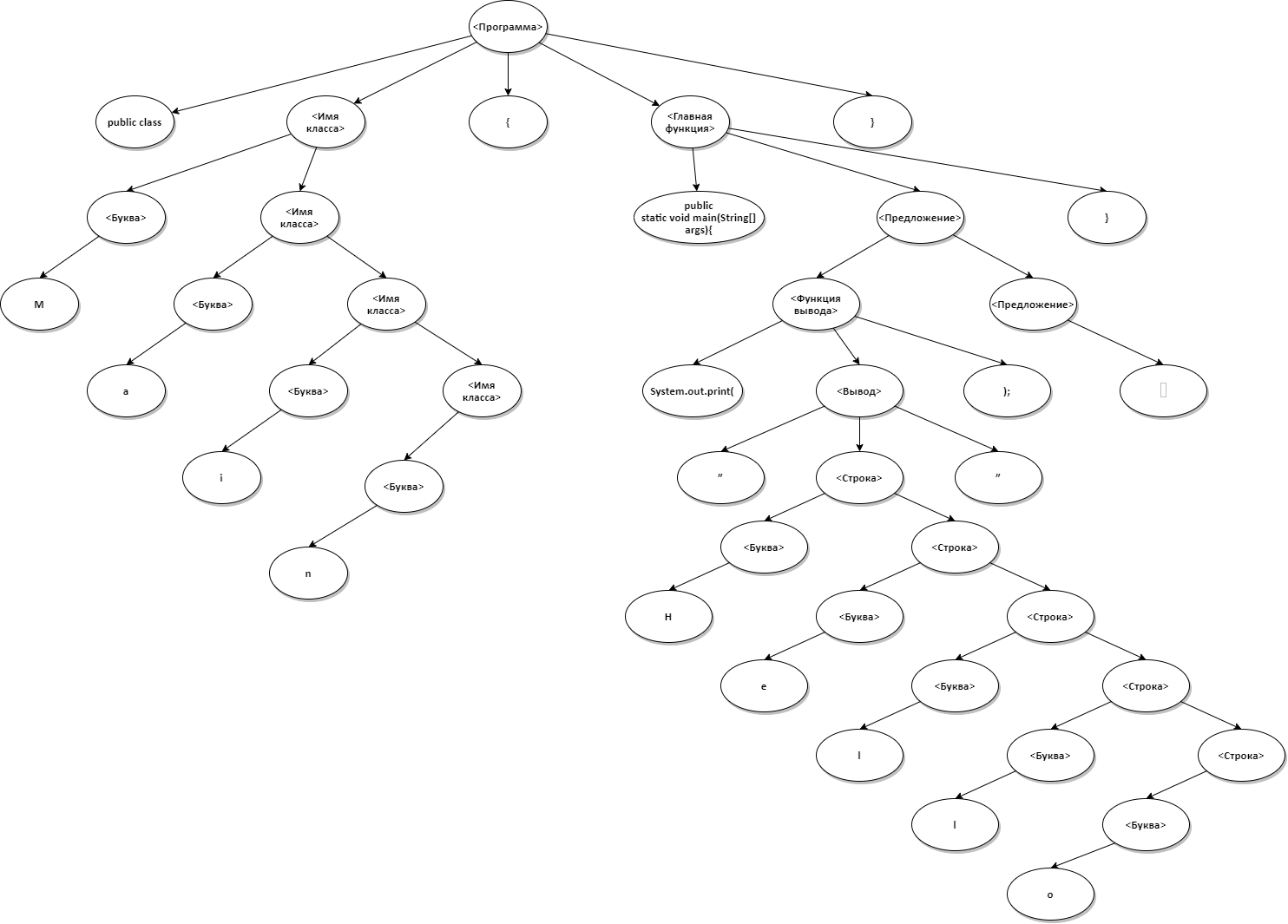


Рисунок -Дерево разбора

**Ошибки синтаксичекского анализатора**

В данном проекте синтаксический анализатор распознает следующие синтаксические ошибки:

1. Наличие цикла в исходной грамматике;
2. Разбор цепочки, не принадлежащей языку исходной грамматики.

# Заключение

В ходе курсового проекта были решены следующие задачи:

1. Сформулирована неформальная постановка задачи.
2. Разработан синтаксис входного языка.
3. Описаны контекстные условия.
4. Создан проект лексического анализатора, включающий в себя таблицы соответствий, ключевых и зарезервированных слов, типов лексем, конечный автомат.
5. 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 [Текст]: в 2-х т. / К.С. Хорстманн. – М.: Вильямс, 2014. – 864 с. – 1 т.
2. Донован, А.А. Язык программирования Go. [Текст] / А.А. Донован, Б.У. Керниган. – М.: Вильямс, 2016. – 432 с.
3. Ахо, А. Теория синтаксического анализа, перевода и компиляции [Текст]: в 2-х т. / А. Ахо, Д. Ульман. – М.: Мир, 1978 – 384 с. – 1 т.
4. Серебряков, В.А. Основы конструирования компиляторов [Текст] / В.А. Серебряков, М.П. Галочкин. – М.: УРСС, 2001 – 192 с.
5. Грис, Д. Конструирование компиляторов для цифровых вычислительных машин [Текст] / Д. Грис. - М. : Мир, 1975 – 575 с.