**Содержание**

[**Введение** 4](#_Toc27411005)

[**1. Спецификация языка программирования** 5](#_Toc27411006)

[**1.1.** **Характеристика языка программирования** 5](#_Toc27411007)

[**1.2.** **Алфавит языка** 5](#_Toc27411008)

[**1.3.** **Символы сепараторы** 5](#_Toc27411009)

[**1.4.** **Применяемые кодировки** 6](#_Toc27411010)

[**1.5.** **Типы данных** 7](#_Toc27411011)

[**1.6.** **Преобразование типов данных** 7](#_Toc27411012)

[**1.7.** **Идентификаторы** 7](#_Toc27411013)

[**1.8.** **Литералы** 8](#_Toc27411015)

[**1.9.** **Область видимости идентификаторов** 8](#_Toc27411016)

[**1.10.** **Инициализация данных** 8](#_Toc27411017)

[**1.11.** **Инструкции языка** 8](#_Toc27411018)

[**1.12.** **Операции языка** 9](#_Toc27411019)

[**1.13.** **Выражения и их вычисления** 9](#_Toc27411020)

[**1.14.** **Программные конструкции языка** 9](#_Toc27411021)

[**1.15.** **Область видимости** 10](#_Toc27411022)

[**1.16.** **Семантические проверки** 10](#_Toc27411023)

[**1.17.** **Распределение оперативной памяти на этапе выполнения** 11](#_Toc27411024)

[**1.18.** **Стандартная библиотека и её состав** 11](#_Toc27411025)

[**1.19.** **Точка входа** 11](#_Toc27411027)

[**1.20.** **Препроцессор** 11](#_Toc27411028)

[**1.21.** **Соглашения о вызовах** 11](#_Toc27411029)

[**1.22.** **Объектный код** 11](#_Toc27411030)

[**1.23.** **Классификация сообщений транслятора** 12](#_Toc27411031)

[**1.24.** **Контрольный пример** 12](#_Toc27411032)

[**2. Структура транслятора** 13](#_Toc27411033)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 13](#_Toc27411034)

[**2.2 Перечень входных параметров транслятора** 14](#_Toc27411035)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 14](#_Toc27411036)

[**3. Разработка лексического анализатора** 15](#_Toc27411037)

[**3.1 Структура лексического анализатора** 15](#_Toc27411038)

[**3.2 Контроль входных символов** 15](#_Toc27411039)

[**3.3 Удаление избыточных символов** 16](#_Toc27411040)

[**3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов** 16](#_Toc27411041)

[**3.5 Основные структуры данных** 17](#_Toc27411042)

[**3.6 Принцип обработки ошибок** 17](#_Toc27411043)

[**3.7 Структура и перечень сообщений лексического анализатора** 17](#_Toc27411044)

[**3.8 Параметры лексического анализатора** 18](#_Toc27411045)

[**3.9 Алгоритм лексического анализа** 18](#_Toc27411046)

[**3.10 Контрольный пример** 18](#_Toc27411047)

[**4. Разработка синтаксического анализатора** 19](#_Toc27411048)

[**4.1 Структура синтаксического анализатора** 19](#_Toc27411049)

[**4.2 Контекстно-свободная грамматика, описывающая синтаксис языка** 19](#_Toc27411050)

[**4.3 Построение конечного магазинного автомата** 21](#_Toc27411051)

[**4.4 Основные структуры данных** 22](#_Toc27411052)

[**4.5 Описание алгоритма синтаксического разбора** 22](#_Toc27411053)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 22](#_Toc27411061)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 23](#_Toc27411062)

[**4.8 Принцип обработки ошибок** 23](#_Toc27411063)

[**5. Разработка семантического анализатора** 24](#_Toc27411064)

[**5.1 Структура семантического анализатора** 24](#_Toc27411065)

[**5.2 Функции семантического анализатора** 24](#_Toc27411066)

[**5.3 Перечень сообщений семантического анализатора** 24](#_Toc27411067)

[**5.4 Принцип обработки ошибок** 24](#_Toc27411068)

[**5.5 Контрольный пример** 24](#_Toc27411069)

[**Глава 6. Преобразование выражений** 25](#_Toc27411070)

[**6.1 Выражения, допускаемые языком** 25](#_Toc27411071)

[**6.2 Польская запись** 25](#_Toc27411072)

[**6.3 Программная реализация обработки выражений** 26](#_Toc27411073)

[**6.4 Контрольный пример** 26](#_Toc27411074)

[**7. Генерация кода** 27](#_Toc27411075)

[**7.1 Структура генератора кода** 27](#_Toc27411076)

[**7.2 Представление типов данных в оперативной памяти** 27](#_Toc27411077)

[**7.3 Алгоритм работы генератора кода** 28](#_Toc27411078)

[**8. Тестирование транслятора** 29](#_Toc27411079)

[**8.1 Тестирование фазы проверки на допустимость символов** 29](#_Toc27411080)

[**8.2** **Тестирование лексического анализатора** 29](#_Toc27411081)

[**8.3 Тестирование синтаксического анализатора** 29](#_Toc27411082)

[**8.4 Тестирование семантического анализатора** 29](#_Toc27411083)

[**Заключение** 30](#_Toc27411084)

[**Приложения** 31](#_Toc27411085)

[**Контрольный пример** 31](#_Toc27411086)

[**Приложение А** 32](#_Toc27411087)

[**Приложение Б** 41](#_Toc27411088)

[**Приложение В** 42](#_Toc27411089)

[**Приложение Г** 43](#_Toc27411090)

[**Приложение Д** 46](#_Toc27411091)

[**Приложение Е** 50](#_Toc27411092)

# **Введение**

Основной целью данной курсовой работы является разработка транслятора для языка программирования HKV-2019. Главная задача транслятора заключается в том, чтобы сделать программу, написанную языке программирования HKV-2019, понятной компьютеру. В данном курсовом проекте трансляция будет осуществляться в код на языке Assembly.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбратка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка выражений;

– генерация кода на язык Assembly;

– тестирование транслятора.

Язык программирования HKV-2019 предназначен для работы с консолью, выполнения простейших арифметический действий, операций над строками и логическими переменными.

# **1. Спецификация языка программирования**

## **Характеристика языка программирования**

Язык HKV-2019 ­– это процедурный, универсальный, строго типизированный, компилируемый язык. He является объектно-ориентированным.

* 1. **Алфавит языка**

Исходный код HKV-2019 может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9. Русские символы запрещены.

## **Символы сепараторы**

Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ;  ‘ ‘ (пробел)  ,  >  <  =  +  -  \* | Разделение инструкций |
| {  } | Программный блок инструкций |
| (  ) | Параметры функций / приоритетность операций (в выражениях) |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования HKV-2019 используется кодировка Windows-1251, представленная на рисунке ниже:



## **Типы данных**

В языке HKV-2019 есть 3 типа данных: целочисленный, строковый, символьный. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка HKV-2019

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| integer | Фундаментальный тип данных. Предусмотрен для объявления целочисленных положительных данных (4 байта). Максимальное значение 4294967295 или 0hffffffff.  Автоматически инициализируется нулевым значением. |
| string | Фундаментальный тип данных. Предусмотрен для объявления строк. Автоматическая инициализация строкой нулевой длины. Максимальное количество символов в строке – 48; |
| char | Фундаментальный тип данных. Предусмотрен для объявления символьных переменных. |

## **Преобразование типов данных**

В языке программирования HKV-2019 преобразование типов данных не поддерживается.

## **Идентификаторы**

## В имени идентификатора допускаются символы латинского алфавита нижнего регистра. Также имя идентификатора может содержать цифры (цифра не должна находиться на первой позиции). Максимальная длина идентификатора 15 (в случае превышения идентификатором максимальной длины, имя идентификатора урезается).

## **Литералы**

В языке существует 3 типа литералов: целого, символьного, логического типов. Краткое описание литералов представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа  (десятичные)  (шестнадцатеричные) | Целочисленные неотрицательные литералы, инициализируются 0. Литералы только rvalue.  Шестнадцатеричные вида 0h123abcd  Максимальное значение числового литерала - 4294967295 и 0hffffffff в десятичной и шестнадцатеричной форме соответственно. |
| Строковые литералы | Символы, заключённые в “ “(двойные кавычки), инициализируются пустой строкой, строковые переменные. Только rvalue. Максимальная длина строкового литерала – 48. |
| Символьные литералы | Символы, заключённые в ‘ ’ (одинарные кавычки).  Только rvalue. |

## **Область видимости идентификаторов**

Область видимости «сверху вниз». В языке HKV-2019 требуется обязательное объявление переменной перед её использованием. Имеется возможность объявления одинаковых переменных в разных блоках.

## **Инициализация данных**

Таблица 1.4 – Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| <тип данных> create <идентификатор>; | Автоматическая инициализация: переменные типа integer инициализируются нулём, переменные типа string – пустой строкой; |
| <идентификатор> = <значение>; | Присваивание переменной значения. |

## **Инструкции языка**

Все возможные инструкции языка программирования HKV-2019 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования HKV-2019

|  |  |
| --- | --- |
| Инструкция | Запись на языке HKV-2019 |
| Объявление переменной | <тип данных> create <идентификатор>; |
| Присваивание | <идентификатор> = <значение> | <идентификатор>; |
| Объявление внешней функции | <тип данных> function <идентификатор>  (<тип данных> <идентификатор>, …) {…} |
| Блок инструкций | {  …  } |
| Возврат из подпрограммы | rback <идентификатор> | <литерал>; |
| Вывод данных | wout (<идентификатор> | <литерал>); |
| Оператор цикла | loop(<идентификатор> | <литерал>) (числовой)  {  } |

## **Операции языка**

Язык программирования HKV-2019 может выполнять арифметические операции, представленные в таблице 1.6.

Таблица 1.6 – Приоритетности операций языка программирования HKV-2019

|  |  |
| --- | --- |
| Операция | Приоритетность операции |
| (  ) | 0 |
| , | 1 |
| +  - | 2 |
| \* | 3 |
| > (сдвиг вправо)  < (сдвиг влево) | 4 |

Максимальным значением приоритетности является “4”, минимальным “0” соответственно.

## **Выражения и их вычисления**

Круглые скобки могут использоваться для записи математических выражений, обязаны использоваться при объявлении функции. Фигурные скобки используются для составления блоков кода функций, операторов.

## **Программные конструкции языка**

Ключевые программные конструкции языка программирования HKV-2019 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка HKV-2019

|  |  |
| --- | --- |
| Главная функция (точка входа в приложение) | main  {  } |
| Функция | <тип> function <идентификатор>(<тип> <идентификатор>, …)  {  …  rback <идентификатор> | <литерал>;  } |

## **Область видимости**

В языке HKV-2019 переменные могут только внутри программного блока функций. В языке HKV-2019 переменные обязаны находиться внутри программного блока функций. Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

## **Семантические проверки**

Таблица с перечнем семантических проверок, предусмотренных языком, приведена в таблице 1.8.

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы не должны повторяться. |
| 2 | Тип данных переменной должен совпадать с типом значения, которое присваивается этому типу. |
| 3 | Идентификатор должен быть объявлен до его использования. |
| 4 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении. |
| 5 | Тип данных возвращаемой функции должен совпадать с типом при ее объявлении. |
| 6 | Тип данных передаваемых значений в функцию стандартной библиотеки должен соответствовать заявленному. |
| 7 | Операнды в арифметическом выражении не могут быть разных типов. |
| 8 | Невозможно использование строк в выражениях. |

## **Распределение оперативной памяти на этапе выполнения**

Все переменные размещаются в стеке. Таблица лексем и таблица идентификаторов сохраняются в структуры с выделенной под них динамической памятью, которая очищается по окончанию работы транслятора.

## **Стандартная библиотека и её состав**

Функции стандартной библиотеки с описанием представлены в таблице 1.9. Стандартная библиотека написана на языке Assembly.

Таблица 1.9 – Состав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| rand | integer | integer min  integer max | Возвращает сгенерированное рандомно число (от min до max или от 1 до 100000, если диапазон не задан) |
| strlen | integer | string str | Возвращает длину строки str |

## Так же, в стандартную библиотеку входит функция преобразования числа в строку для последующего вывода в консоль intToString.

**Ввод и вывод данных**

wout (<идентификатор или литерал>); – вывод в стандартный поток вывода.

## **Точка входа**

Точкой входа является функция main.

## **Препроцессор**

Препроцессор в языке программирования HKV-2019 не предусмотрен.

## **Соглашения о вызовах**

В языке вызов функций происходит по соглашению о вызовах stdcall. Особенности stdcall:

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## **Объектный код**

HKV-2019 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке HKV-2019 и выявления её транслятором в текущий файл протокола выводится сообщение. Их классификация сообщений приведена в таблице 1.10.

Таблица 1.10. – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-99 | Системные ошибки |
| 100-199 | Ошибки параметров, открытия и чтения файлов |
| 300-399 | Ошибки лексического анализа |
| 500-599 | Ошибки семантического анализа |
| 700-799 | Ошибки синтаксического анализа |

## **Контрольный пример**

Контрольный пример представлен в главе Приложения.

# **2. Структура транслятора**

## **2.1 Компоненты транслятора, их назначение и принципы взаимодействия**

Транслятор преобразует программу, написанную на языке HKV-2019 в программу на языке ассемблера. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.

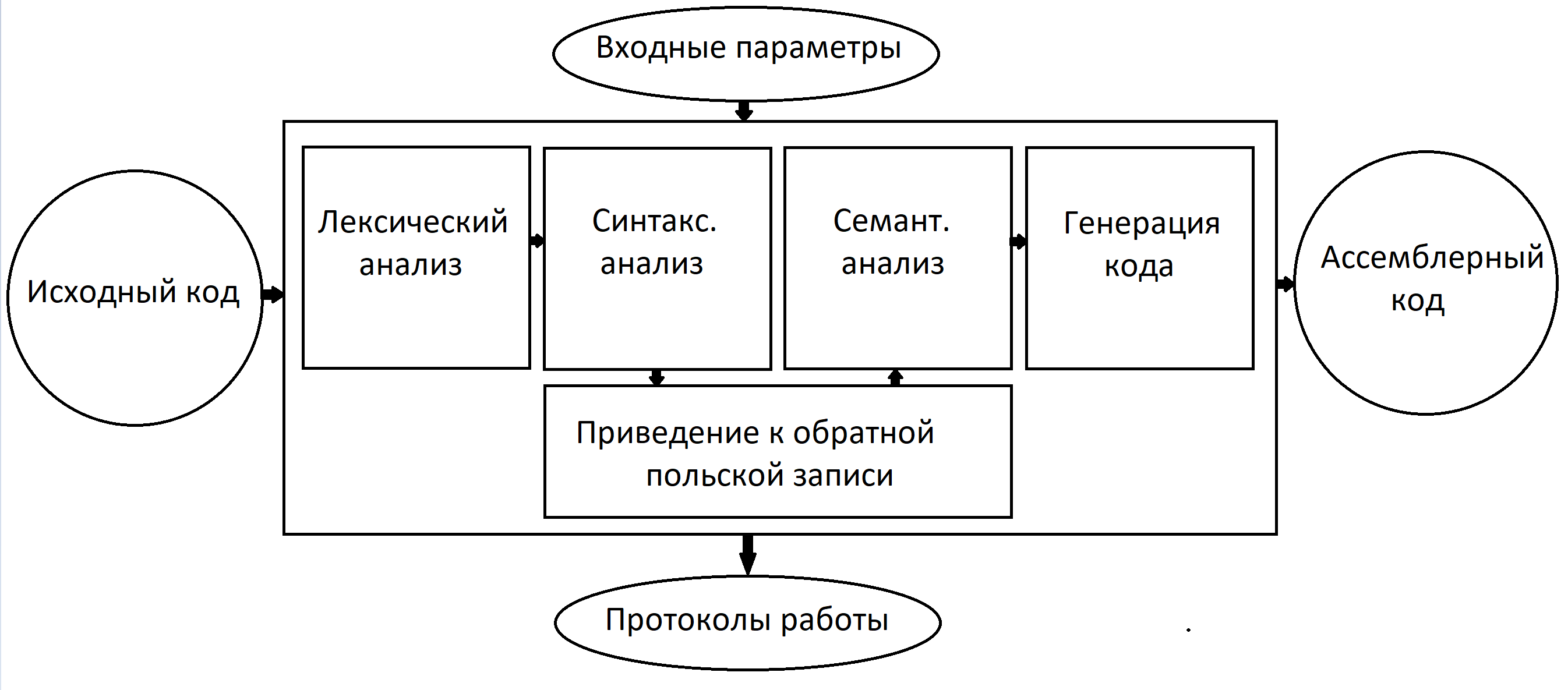


Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка и формирование таблицы лексем и таблицы идентификаторов.

Семантический анализ в свою очередь является проверкой исходной программы на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций. Входным параметром для синтаксического анализа является таблица лексем. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора.

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке HKV-2019 в код на языке Ассемблера.

## **2.2 Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка HKV-2019

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с расширением .txt, в котором содержится исходный код на HKV-2019 | Не предусмотрено |
| -log:<имя\_файла> | Будет опрелелять файлы, содержащие результат работы программы. | <имя\_in\_файла>.log |
| -out:<имя\_файла> | Файл для записи результата работы транслятора | <имя\_in\_файла>.out.asm |

Стоит заметить, что имена файлов необходимо записывать с расширением.

## **2.3 Перечень протоколов, формируемых транслятором и их содержимое**

Таблица с перечнем протоколов, формируемых транслятором языка

HKV-2019 и их назначением представлена в таблице 2.2

Таблица 2.2 – Протоколы, формируемые транслятором языка HKV-2019

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала с параметром <log> | Содержит информацию о входных параметрах в приложения, этапе проверки символов на допустимость, результат работы лексического и синтаксического анализаторов. |
| Выходной файл c параметром <out> | Содержит сгенерированный код на языке Ассемблера. |

# **3. Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке HKV-2019. На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1

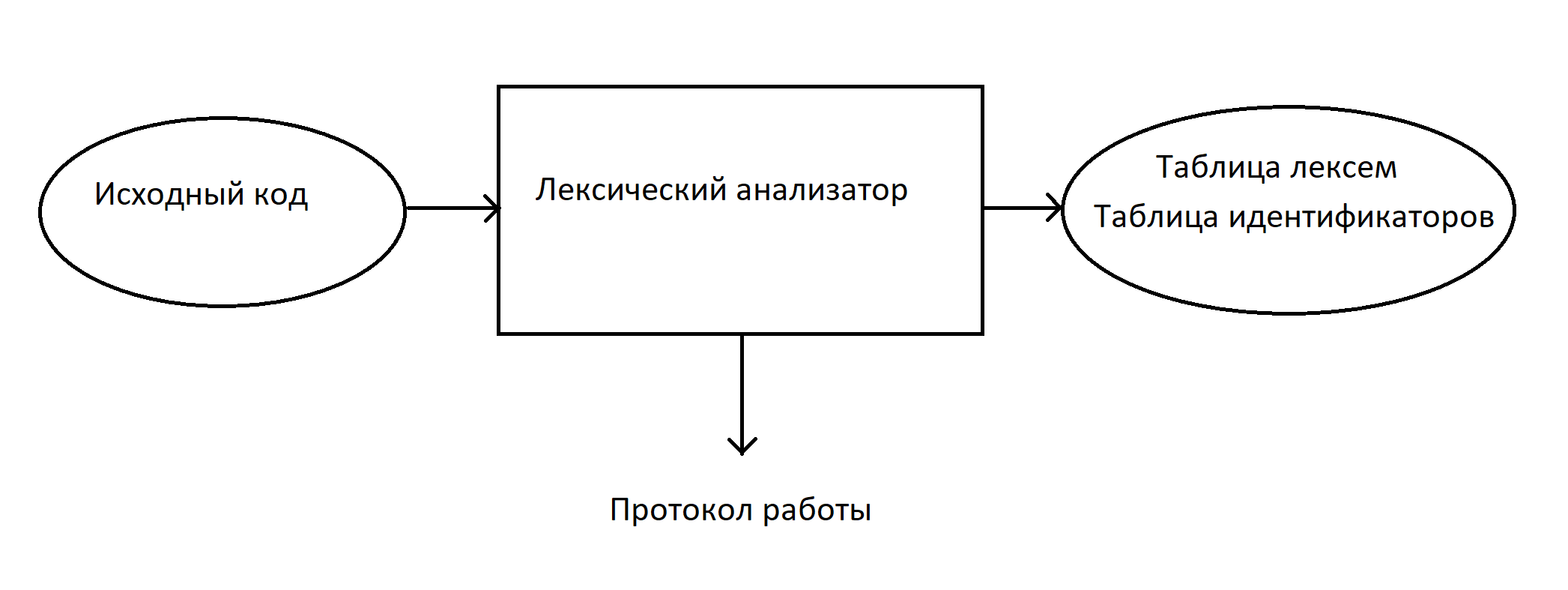


Рисунок 3.1 – Структура лексического анализатора HKV-2019

## **3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2

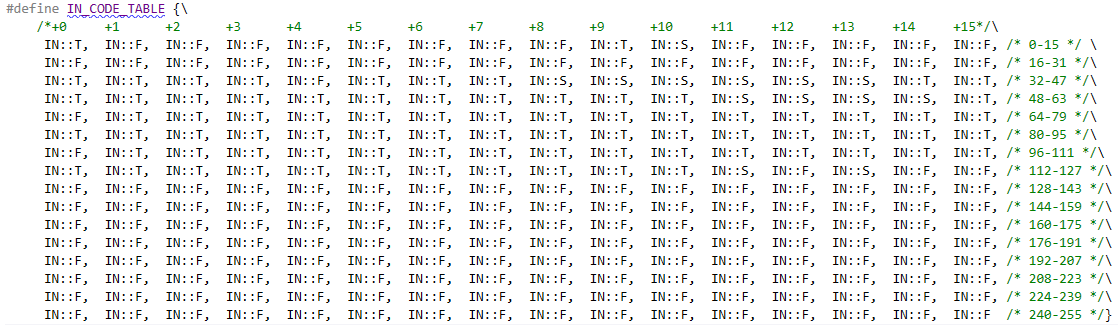


Рисунок 3.2. – Таблица контроля входных символов

Принцип работы таблицы заключается в соответствии значения каждого элемента значению в таблице ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, I – игнорируемый символ, S – сепаратор.

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на лексемы.

Описание алгоритма удаления избыточных символов:

1. посимвольно считываем файл с исходным кодом программы;
2. встреча пробела или знака табуляции является своего рода встречей символа-сепаратора;
3. в отличие от других символов-разделителей, не записываем их в таблицу лексем;
4. в цикле проверяем: если следующий символ в потоке будет знаком табуляции или пробела – пропускаем этот символ до тех пор, пока следующим символом будет не \t и не ‘ ‘.

## **3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов**

Лексемы – это символы, соответствующие ключевым словам, символам операций и сепараторам, необходимые для упрощения дальнейшей обработки исходного кода программы. Данное соответствие описано в таблице 3.1.

Таблица 3.1 Соответствие ключевых слов, символов операций и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | integer | t |
| string | t |
| char | t |
| main | m |
| create | c |
| function | f |
| rback | r |
| wout | w |
| strlen | e |
| rand | g |
| loop | l |
| Иное | Идентификатор | i |
| Литерал | l |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| = | = |
| Операторы | - | - |
| \* | \* |
| + | + |
| > | > |
| < | < |

Пример реализации таблицы лексем представлен в приложении А.

Также в приложении А находятся конечные автоматы, соответствующие лексемам языка HKV-2019.

## **3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка HKV-2019, используемых для хранения, представлены в приложении А.

В таблице лексем содержится лексема, её номер строки в исходном коде, номер в таблице идентификаторов, тип оператора и область видимости лексемы.

В таблице идентификаторов содержится имя идентификатора, его номер строки в исходном коде, тип (функция, переменная…), тип данных, система исчисления, значение и область видимости.

## **3.6 Принцип обработки ошибок**

При возникновении ошибки работа транслятора прекращается.

## **3.7 Структура и перечень сообщений лексического анализатора**

## 

## **3.8 Параметры лексического анализатора**

Входным параметром лексического анализатора является исходный текст программы, написанный на языке HKV-2019.

## **3.9 Алгоритм лексического анализа**

Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором. Цель лексического анализа — выделение и классификация лексем в тексте исходной программы. Лексический анализатор распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. Регулярные выражения — аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа.

Алгоритм работы:

1. считывание текста из исходного файла в буфер;
2. далее – посимвольное считывание из буфера. Если текущий символ конец строки – пункт 8;
3. проверка символа на допустимость;
4. если текущий символ сепаратор – запись в таблицу лексем. Если нет – дозапись в строку-буфер до тех пор, пока сепаратор не встретится;
5. проверка строки-буфера на одно из зарезервированных слов. Если такое есть – запись в таблицу лексем и переход к пункту 2;
6. проверка строки-буфера на соответствие одному из конечных автоматов. Если нет – пункт 8, выход из программы с ошибкой;
7. запись в таблицу лексем соответствующего значения. Запись в таблицу идентификаторов того или иного идентификатора или литерала. Переход к пункту 2;
8. конец работы анализатора.

Все регулярные выражения, имеющиеся в языке, представлены в приложении А.

## **3.10 Контрольный пример**

Результат работы лексического анализатора – таблицы лексем и идентификаторов – представлен в приложении А.

# **4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций. Входом для синтаксического анализа является таблица лексем, полученная после фазы лексического анализа. Выходом – дерево разбора. Структура синтаксического анализатора представлена на рисунке 4.1.

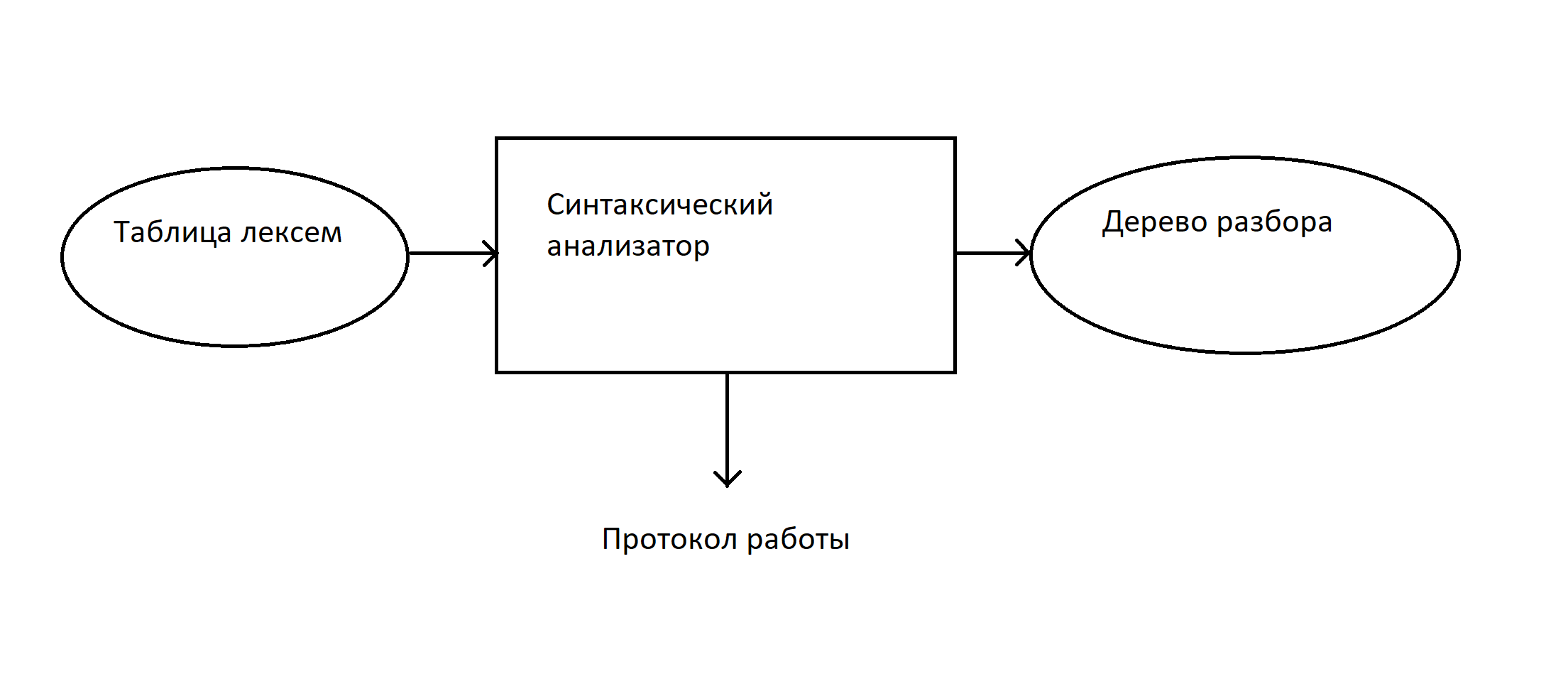


Рисунок 4.1 – Структура синтаксического анализатора

## **4.2 Контекстно-свободная грамматика, описывающая синтаксис языка**

В синтаксическом анализаторе транслятора языка HKV-2019 используется контекстно-свободная грамматика , где

* T – множество терминальных символов (было описано в разделе 1.2 данной пояснительной записки),
* N – множество нетерминальных символов (первый столбец таблицы 4.1),
* P – множество правил языка (второй столбец таблицы 4.1),
* S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика имеет ослабленную нормальную форму Грейбах:

1. , где ; (или , или )
2. , где — начальный символ, при этом если такое правило существует, то нетерминал  не встречается в правой части правил.

Грамматика языка HKV-2019 представлена в приложении Б.

* TS – терминальные символы, которыми являются сепараторы, знаки арифметических операций и некоторые строчные буквы.
* NS – нетерминальные символы, представленные заглавными буквами латинского алфавита.

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов HKV-2019

|  |  |
| --- | --- |
| Нетерминал | Цепочки правил |
| S | tfiFS  m{B} |
| N | tci;B  tci;  tci=E;B  tci=E;  i=E;B  i=E;  i(C);B  i(C);  rE;  o(i){B}B  o(l){B}B  o(i){B}  o(l){B}  w(i)B  w(l)B  w(i);  w(l); |
| B | t  f  iO  i  (E)O  (E)  i(W)O  i(W)  i()O  i() |
| E  E | i  l  (E)vE  (E)  ivE  lvE  i(C)vE  i(C)  i()  i()vE  e(C)  e(C)vE  g()  g()vE  g(C)  g(C)vE |
| P | ti  ti,P |
| C | i  l  i,C  l,C |
| F | (){B}  (P){B} |

## **4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку, описание которой представлено в таблице 4.2. Структура данного автомата показана в приложении В.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ S) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

## **4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающей правила языка HKV-2019. Данные структуры представлены в приложении В.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

## В магазин записывается стартовый символ.

## На основе полученной ранее таблицы лексем формируется входная лента.

## Запускается автомат.

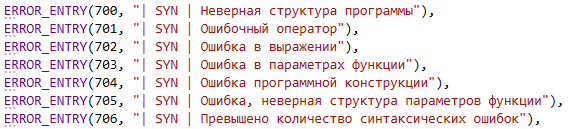
## Выбирается цепочка по первому символу, соответствующая нетерминальному символу, записывается в магазин в обратном порядке.

## Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и магазина. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другое правило нетерминала.

## Если в правиле встретился нетерминал – пункт 4.

## Если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

## **4.6 Структура и перечень сообщений синтаксического анализатора**



## **4.7 Параметры синтаксического анализатора и режимы его работы**

Входным параметром синтаксического анализатора является таблица лексем, полученная на этапе лексического анализа, а также правила контекстно-свободной грамматики в форме Грейбах.

Выходными параметрами являются трассировка прохода таблицы лексем и правила разбора, которые записываются в файл протокола данного этапа обработки.

## **4.8 Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

* синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем;
* если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка, которая записывается в протокол работы и программа останавливается.

**4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке HKV-2019 представлен в приложении Г. Дерево разбора исходного кода также представлено в приложении Г.

# **5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

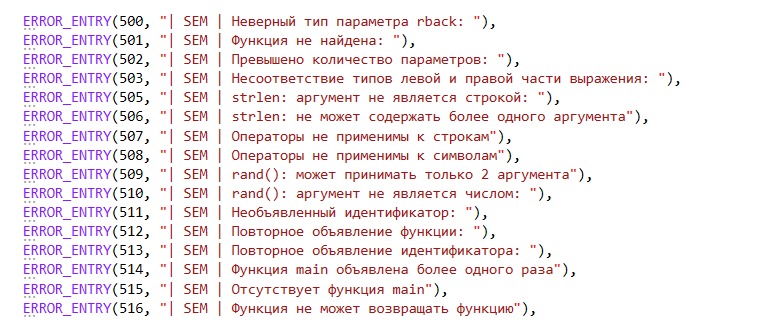
Часть ошибок семантического анализа обрабатываются на этапе лексического анализа. Но ошибки, требующие более сложной обработки (например, несоответствие типов операндов) вынесены в отдельный этап, следующий после синтаксического анализа и преобразования к польской записи, так как с таблицей лексем после в форме ПОЛИЗ работать намного удобнее.

## **5.2 Функции семантического анализатора**

Семантический анализатор выполняет проверку на основные правила языка (семантики языка), которые описаны в разделе 1.16.

## **5.3 Перечень сообщений семантического анализатора**

Сообщения, формируемые семантическим анализатором, представлены ниже:



## **5.4 Принцип обработки ошибок**

Принцип обработки ошибок идентичен принципу обработки ошибок на этапе лексического анализа (раздел 3.6).

## **5.5 Контрольный пример**

Результат работы контрольного примера расположен в приложении А, где показан результат лексического анализатора, т.к. некоторые ошибки семантического анализа обрабатываются еще на этапе формирования таблицы лексем и идентификаторов.

## **Глава 6. Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке HKV-2019 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, > (сдвиг вправо), < (сдвиг влево) и (), и вызовы функций как операнды арифметических выражений

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке HKV-2019

|  |  |
| --- | --- |
| Приоритет | Операция |
| 0 | ( |
| 0 | ) |
| 1 | , |
| 2 | + |
| 2 | - |
| 3 | \* |
| 4 | < |
| 4 | > |
| 5 | ( – скобка параметров функции |
| 5 | ) – скобка параметров функции |

## **6.2 Польская запись**

Выражения в языке HKV-2019 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок.

Обратная польская запись – это форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. 

Алгоритм построения:

– исходная строка: выражение;

– результирующая строка: польская запись;

– стек: пустой;

– результирующая строка: польская запись;

– исходная строка просматривается слева направо;

– операнды переносятся в результирующую строку в порядке их следования;

– операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;

– операция выталкивает все операции с большим или равным приоритетом в результирующую строку;

– запятая не помещается в стек, если в стеке операции, то все выбираются в строку;

– отрывающая скобка помещается в стек;

– закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;

– закрывающая скобка с приоритетом, равным 4, выталкивает все до открывающей с таким же приоритетом и генерирует @ – специальный символ, в которого записывается информация о вызываемой функции, а в поле приоритета для данной лексемы записывается число параметров вызываемой функции;

– по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| b\*2 - n(i) |  |  |
| \*2 - n(i) | b |  |
| 2 - n(i) | b | \* |
| - n(i) | b2 | \* |
| n(i) | b2\* | - |
| (i) | b2\* | - |
| i) | b2\* | - |
| ) | b2\*i | - |
|  | b2\*i@1- |  |

## **6.3 Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений к польской записи представлена в приложении Д.

## **6.4 Контрольный пример**

Пример преобразования выражения к польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи в нашем случае необходимо для построения более простых алгоритмов при последующей обработки таблицы лексем.

В приложении Д приведены изменённые таблицы лексем и идентификаторов, отображающие результаты преобразования выражений в польский формат.

# **7. Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы в цепочку символов выходного языка. На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом.



Рисунок 7.1 Структура генератора кода

## **7.2 Представление типов данных в оперативной памяти**

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка HKV-2019 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке HKV-2019 и на языке ассемблера приведены в таблице 7.1.

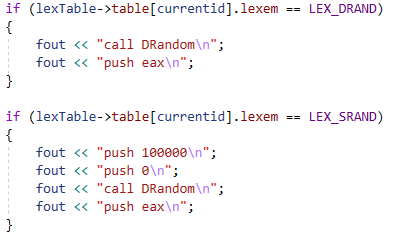
Таблица 7.1 – Соответствия типов идентификаторов языка HKV-2019 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке HKV-2019 | Тип идентификатора на языке ассемблера | Пояснение |
| integer | DWORD | Хранит целочисленный тип данных без знака. |
| string | DB – для литералов  DWORD – для переменных | Литерал хранит последовательность байтов.  Переменные хранят указатель на начало строки литерала. |
| char | DB – для литералов  DWORD – для переменных | Литерал хранит последовательность байтов.  Переменные хранят указатель на начало строки литерала. |

## **7.3 Алгоритм работы генератора кода**

Преобразования происходят по принципу, встретив определённую лексему и зная, в каком месте программы находится сейчас лексема, программа генерирует код на языке Ассемблера.

На рисунке 7.2 представлен пример описания лексемы на языке Ассемблера.



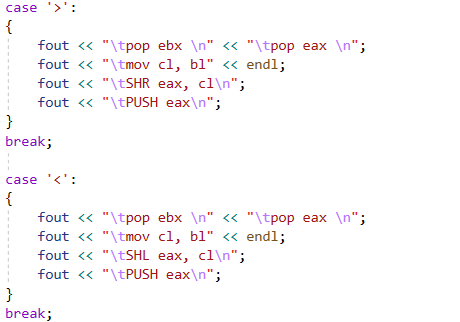


Рисунок 7.2 – Алгоритм для лексемы ‘g’ и операторов сдвига

Генерируемый код записывается в файл, который был параметром ‑out. Сгенерированный код можно посмотреть в приложении Е.

# **8. Тестирование транслятора**

## **8.1 Тестирование фазы проверки на допустимость символов**

В языке HKV-2019 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| wout(‘$’); | Ошибка 111: | IN | Недопустимый символ в исходном коде (-in): $, строка 26 |

## **Тестирование лексического анализатора**

На этапе лексического анализа могут возникнуть ошибки, описанные в пункте 3.7. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| string create 0fefe; | Ошибка 302: | LEX | Имя идентификатора не может начинаться с цифры: 0fefe, строка 32 |

## **8.3 Тестирование синтаксического анализатора**

На этапе синтаксического анализа могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| integer create z | 701: строка 32, | SYN | Ошибочный оператор |

## **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| d = strlen(654); | Ошибка 505: | SEM | strlen: аргумент не является строкой: Lit\_5, строка 19 |

## **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования HKV-2019. Таким образом, были выполнены основные задачи данной курсовой работы:

* Сформулирована спецификация языка HKV-2019;
* Разработаны конечные автоматы и алгоритмы для реализация лексического анализатора;
* Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
* Разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
* Разработан транслятор с языка программирования HKV-2019 на язык низкого уровня Assembler;
* Проведено тестирование всех вышеперечисленных компонентов.

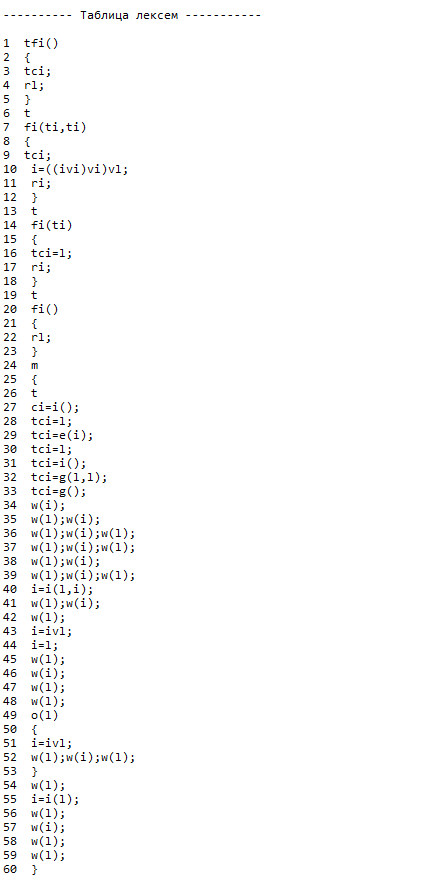
Окончательная версия языка HKV-2019 включает:

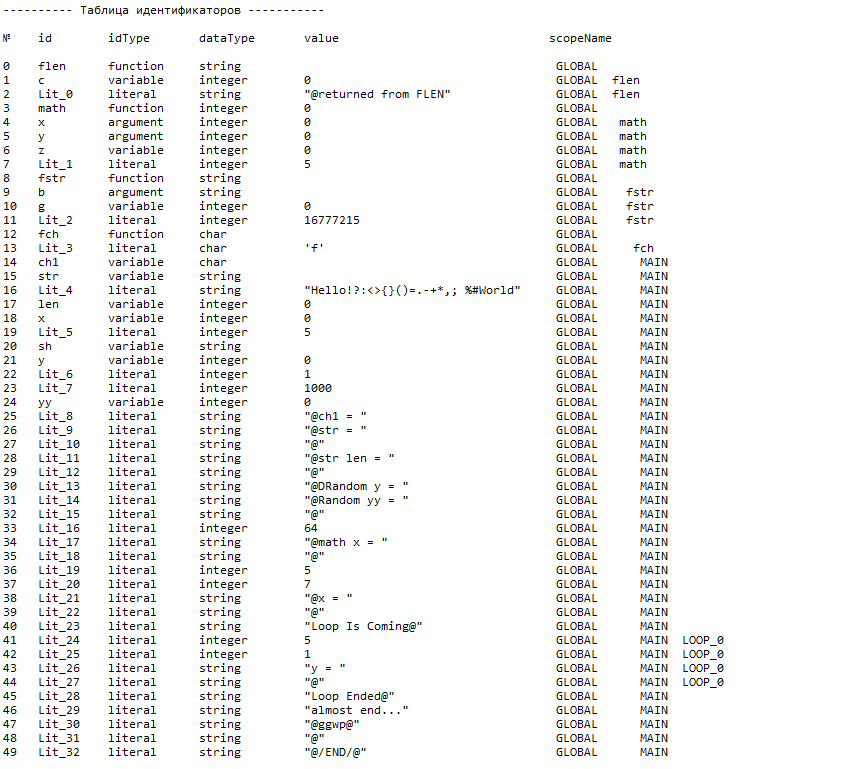
1. 3 типа данных;
2. Поддержка операции вывода;
3. Возможность вызова функций стандартной библиотеки из Ассемблера;
4. Наличие 3 арифметических и 2 сдвиговых операторов для вычисления выражений;
5. Структурированная система для обработки ошибок пользователя.
6. Оператор цикла;

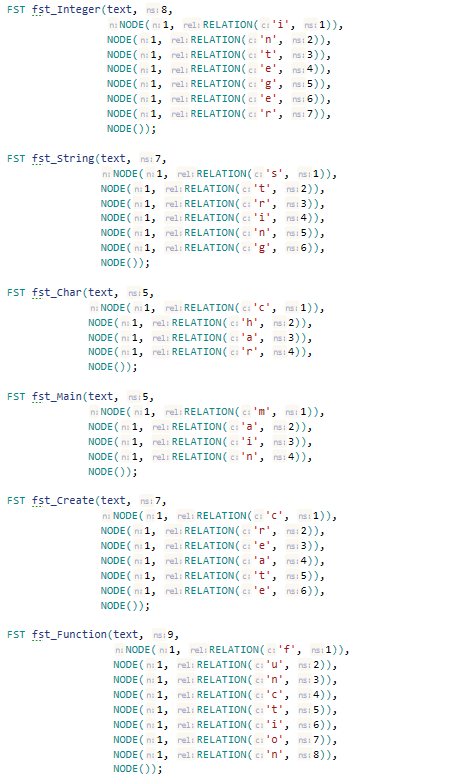
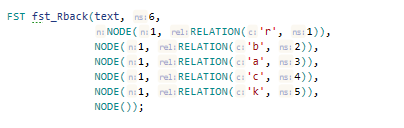
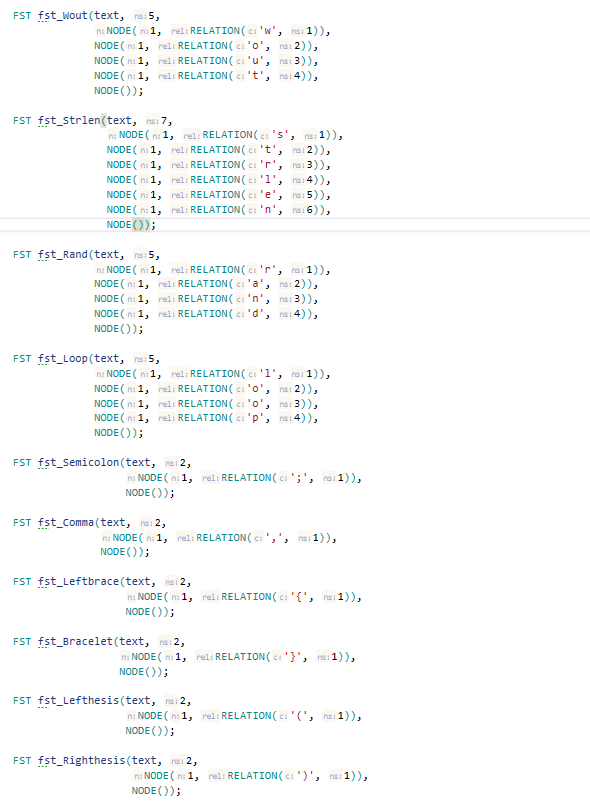
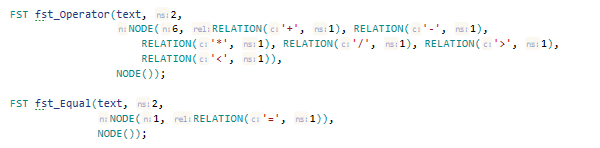
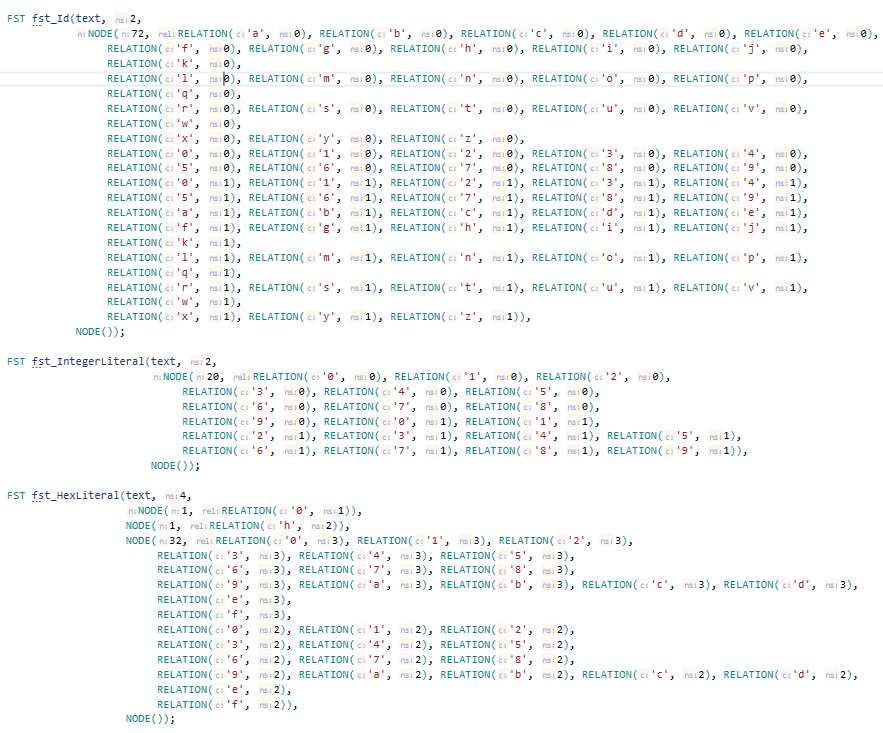
## **Приложения**

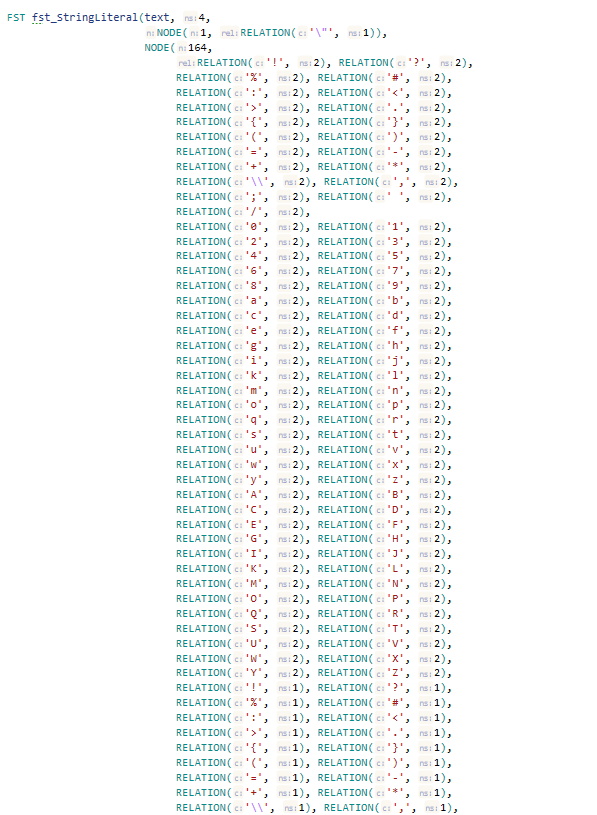
## **Контрольный пример**

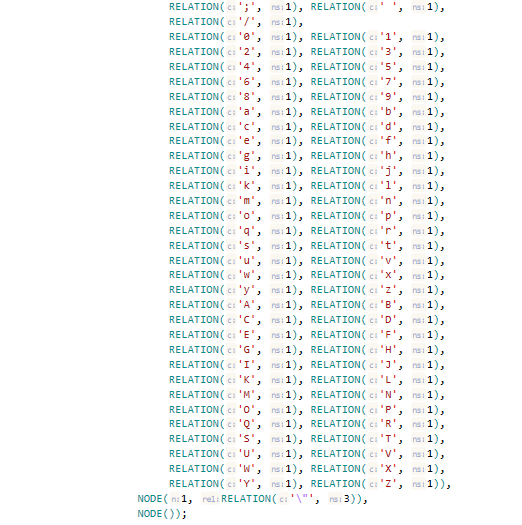
## **Приложение А**

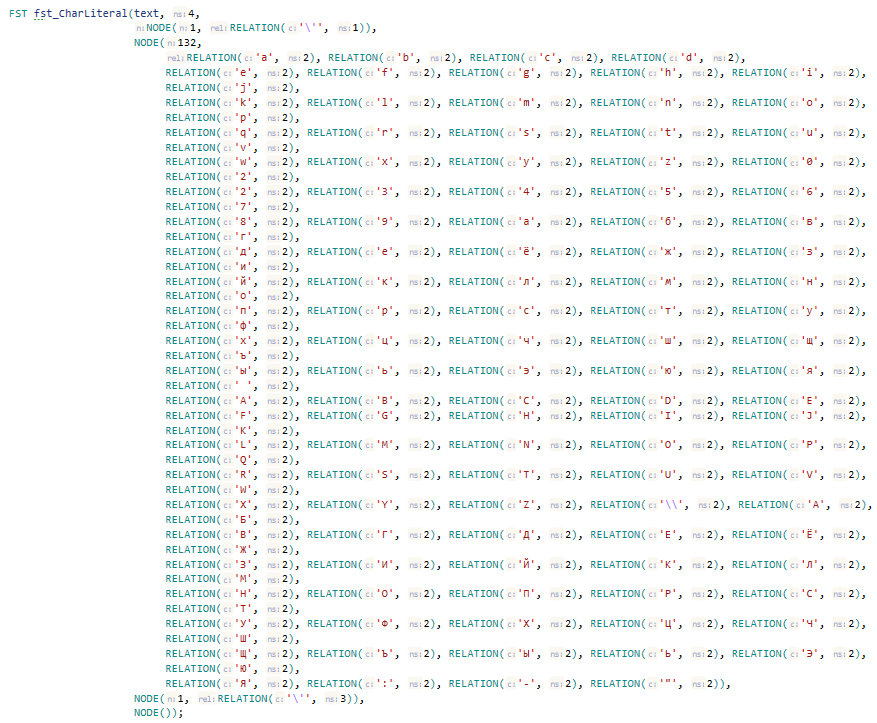


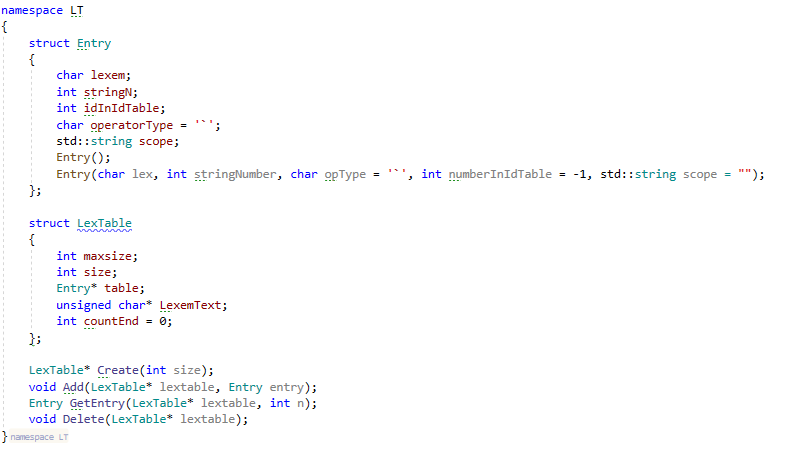




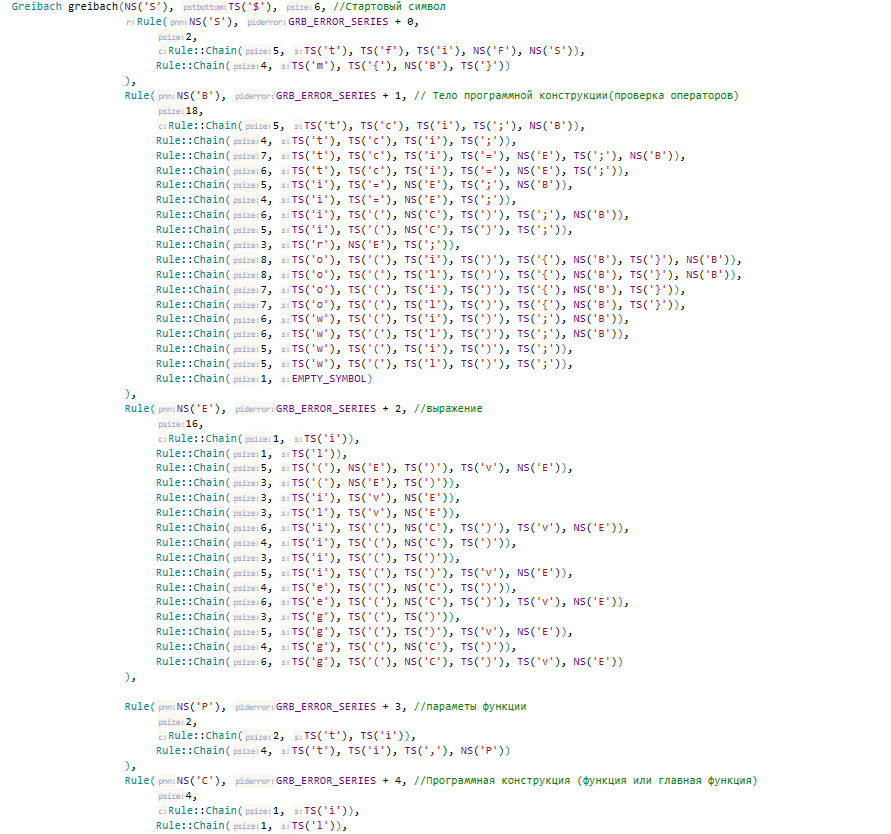


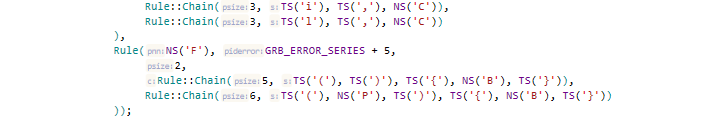




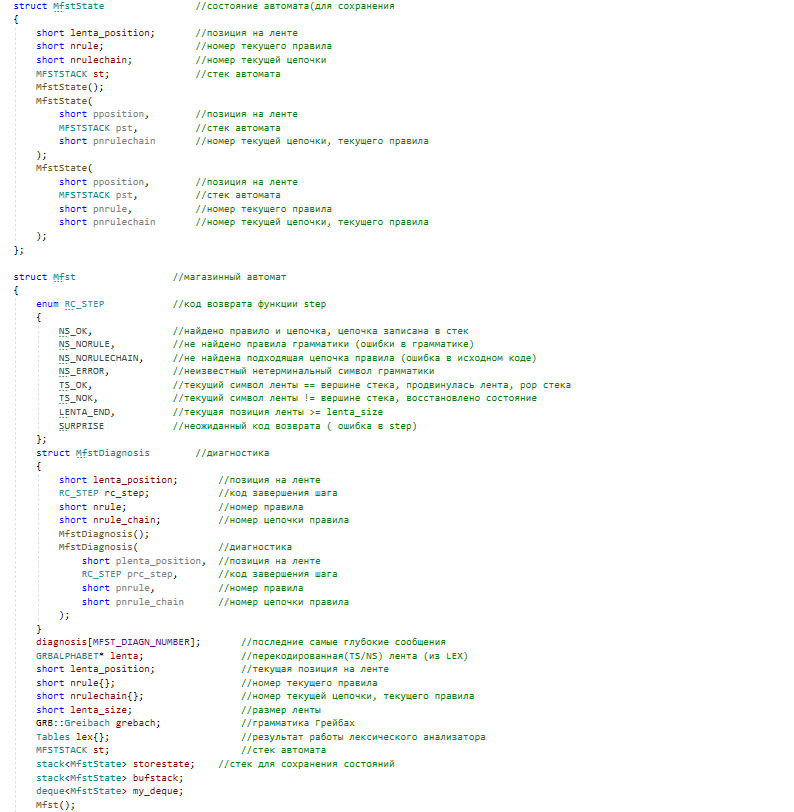
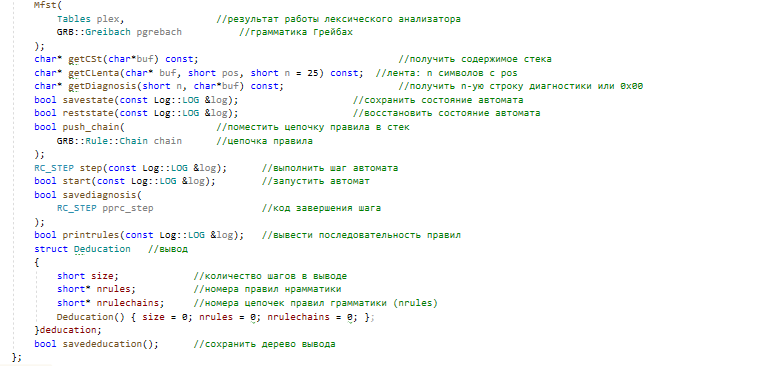
****

## **Приложение Б**



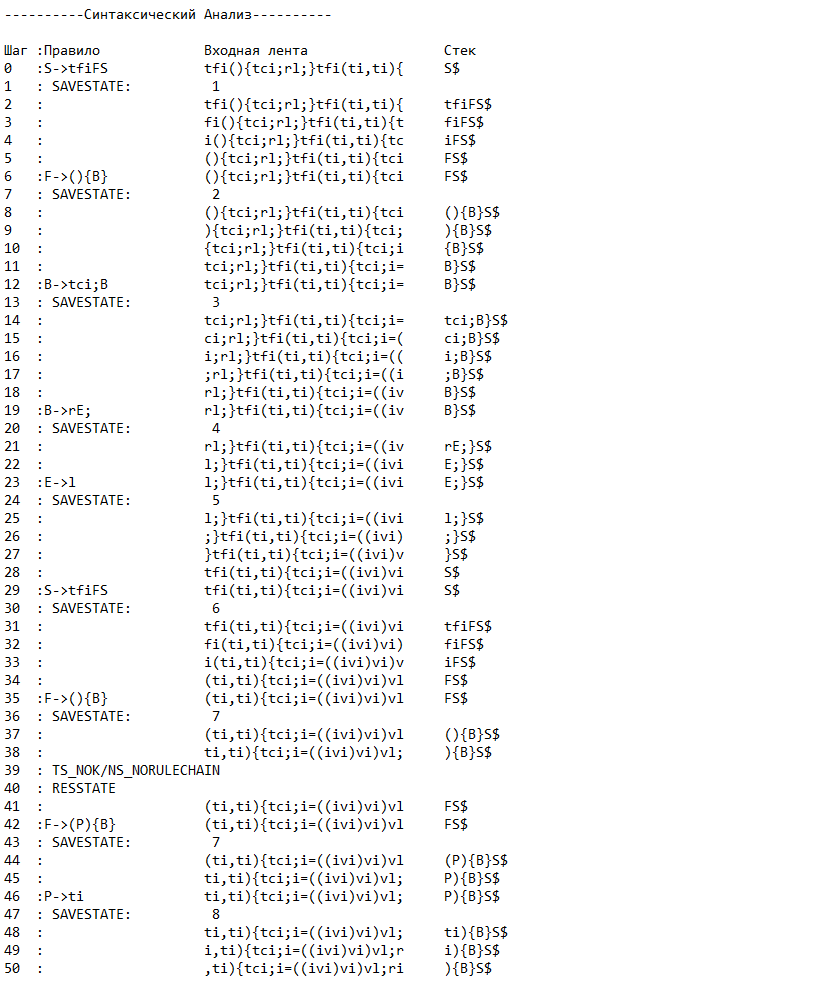


## **Приложение В**

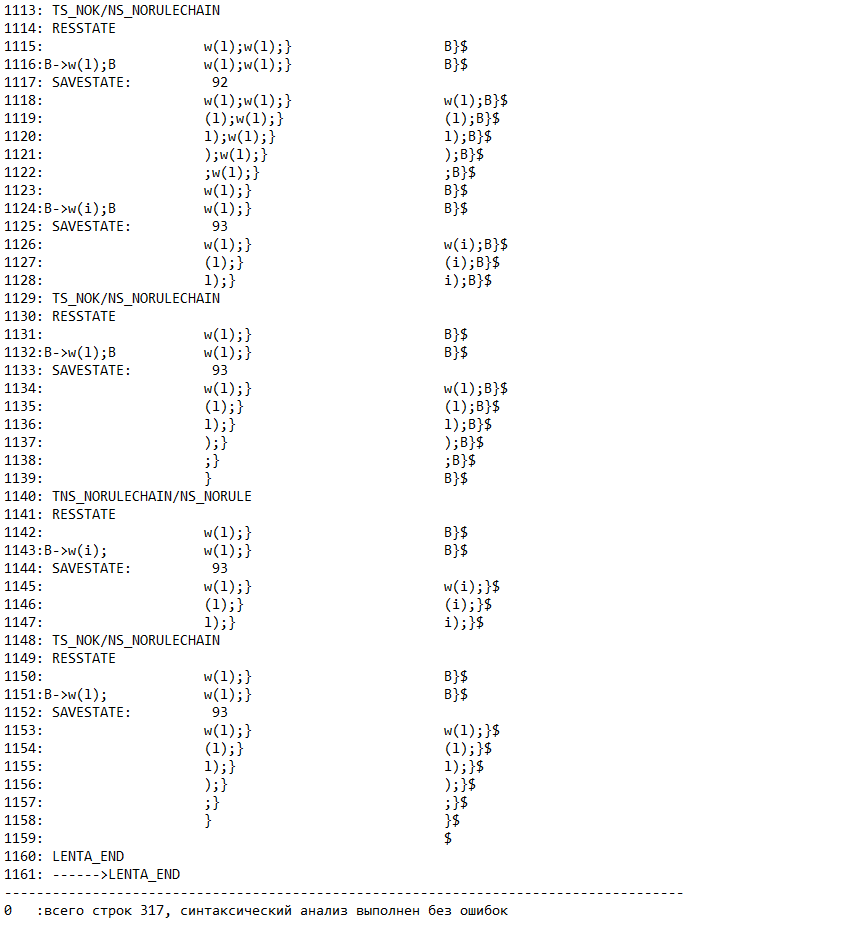


## **Приложение Г**

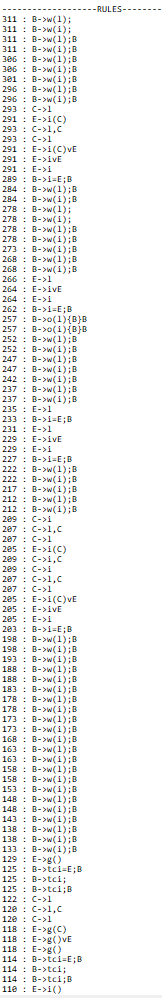
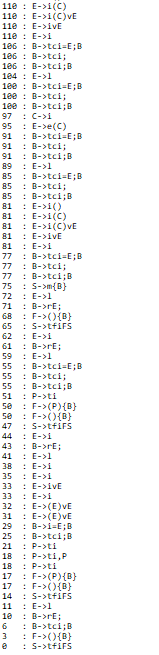
Начало разбора



Конец разбора

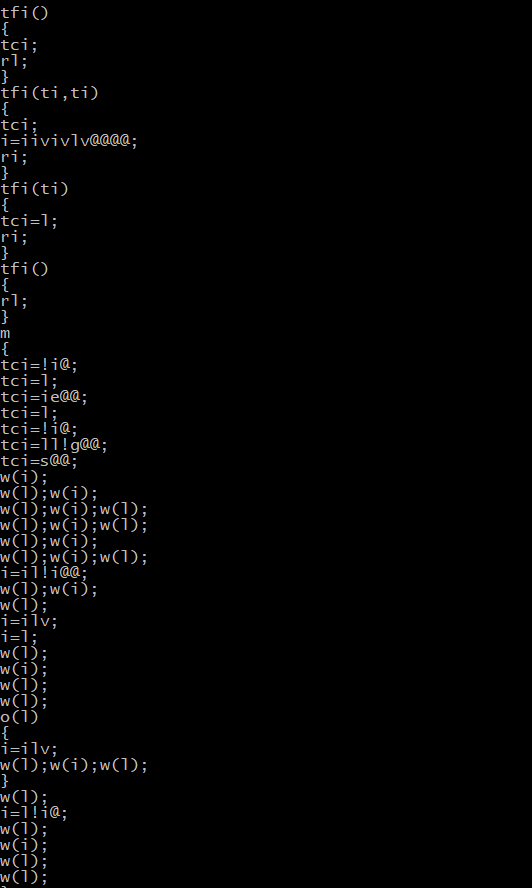


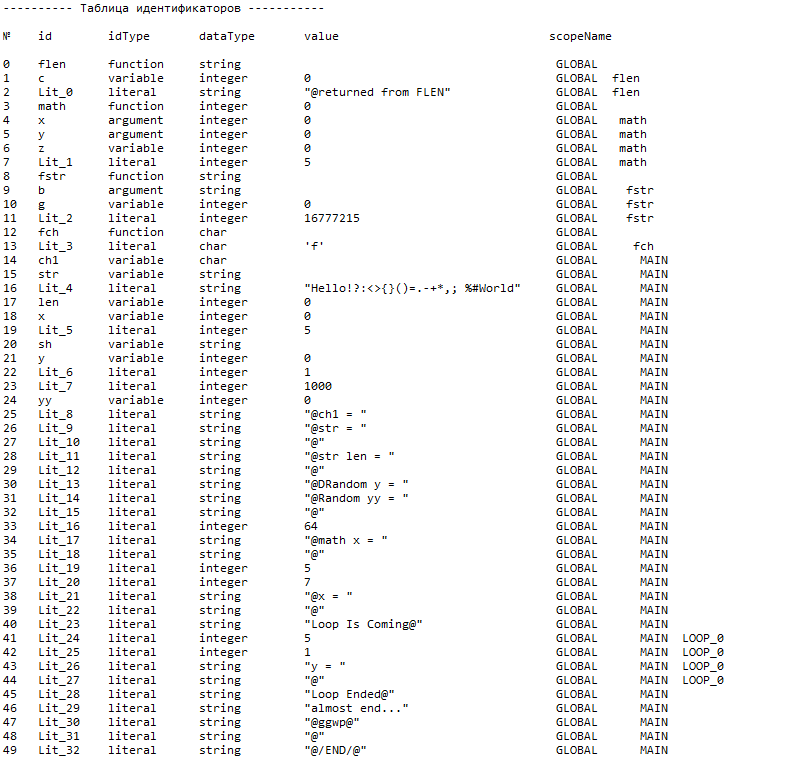
Правила разбора



## **Приложение Д**







## **Приложение Е**

