МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора EPA-2019»

Выполнил студент Ермакович Павел Анатольевич

(Ф.И.О.)

Руководитель проекта преп.-стаж. Котович Дмитрий Витальевич

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Пацей Н.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты преп.-стаж. Котович Дмитрий Витальевич

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер преп.-стаж. Котович Дмитрий Витальевич

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2019

Оглавление

[Введение 4](#_Toc27485568)

[1. Спецификация языка программирования 5](#_Toc27485569)

[1.1 Характеристика языка программирования 5](#_Toc27485570)

[1.2 Алфавит языка 5](#_Toc27485571)

[1.3 Применяемые сепараторы 6](#_Toc27485572)

[1.4 Применяемые кодировки 6](#_Toc27485573)

[1.5 Типы данных 6](#_Toc27485574)

[1.6 Преобразование типов данных 7](#_Toc27485575)

[1.7 Идентификаторы 7](#_Toc27485576)

[1.8 Литералы 7](#_Toc27485577)

[1.9 Объявление данных и область видимости 7](#_Toc27485578)

[1.10 Инициализация данных 8](#_Toc27485579)

[1.11 Инструкция языка 8](#_Toc27485580)

[1.12 Операции языка 9](#_Toc27485581)

[1.13 Выражения и их вычисления 10](#_Toc27485582)

[1.14 Программные конструкции языка 10](#_Toc27485583)

[1.15 Область видимости идентификаторов 11](#_Toc27485584)

[1.16 Семантические проверки 11](#_Toc27485585)

[1.17 Распределение оперативной памяти на этапе выполнения 11](#_Toc27485586)

[1.18 Стандартная библиотека и её состав 12](#_Toc27485587)

[1.19 Ввод и вывод данных 12](#_Toc27485588)

[1.20 Точка входа 12](#_Toc27485589)

[1.21 Препроцессор 12](#_Toc27485590)

[1.22 Соглашения о вызовах 12](#_Toc27485591)

[1.23 Объектный код 13](#_Toc27485592)

[1.24 Классификация сообщений транслятора 13](#_Toc27485593)

[1.25 Контрольный пример 13](#_Toc27485594)

[2 Структура транслятора 14](#_Toc27485595)

[2.1 Компоненты транслятора и принципы взаимодействия 14](#_Toc27485596)

[2.2 Перечень параметров транслятора входных 15](#_Toc27485597)

[2.3 Перечень протоколов формируемых транслятором 16](#_Toc27485598)

[3 Разработка лексического анализатора 17](#_Toc27485599)

[3.1 Структура лексического анализатора 17](#_Toc27485600)

[3.2 Контроль входных символов 18](#_Toc27485601)

[3.3 Удаление избыточных символов 18](#_Toc27485602)

[3.4 Перечень ключевых слов, сепараторов, символов операций, соответствующих им лексем и конечных автоматов 18](#_Toc27485603)

[3.5 Основные структуры данных 20](#_Toc27485604)

[3.6 Принцип обработки ошибок и их перечень 20](#_Toc27485605)

[3.7 Структура и перечень сообщений лексического анализатора 20](#_Toc27485606)

[3.8 Параметры лексического анализатора и режимы его работы 21](#_Toc27485607)

[3.9 Алгоритм лексического анализатора 21](#_Toc27485608)

[3.10 Контрольный пример 22](#_Toc27485609)

[4 Разработка синтаксического анализатора 23](#_Toc27485610)

[4.1 Структура синтаксического анализатора 23](#_Toc27485611)

[4.2 Грамматика языка 23](#_Toc27485612)

[4.3 Построение конечного магазинного автомата 24](#_Toc27485613)

[4.4 Основные структуры данных 25](#_Toc27485614)

[4.5 Описание алгоритма синтаксического разбора 25](#_Toc27485615)

[4.6 Структура и перечень сообщений синтаксического анализатора 25](#_Toc27485616)

[4.7 Параметры синтаксического анализатора и режимы его работы 26](#_Toc27485617)

[4.8 Принцип обработки ошибок 26](#_Toc27485618)

[4.9 Контрольный пример 26](#_Toc27485619)

[5 Разработка семантического анализатора 27](#_Toc27485620)

[5.1 Структура семантического анализатора 27](#_Toc27485621)

[5.2 Функции семантического анализатора 27](#_Toc27485622)

[5.3 Принцип обработки ошибок 27](#_Toc27485623)

[5.4 Структура и перечень сообщений семантического анализатора 27](#_Toc27485624)

[5.5 Контрольный пример 27](#_Toc27485625)

[6 Преобразование выражений 28](#_Toc27485626)

[6.1 Выражения, допускаемые языком 28](#_Toc27485627)

[6.2 Польская запись и принцип её построения 28](#_Toc27485628)

[6.3 Примеры преобразования выражений 29](#_Toc27485629)

[7 Генерация кода 30](#_Toc27485630)

[8 Тестирование транслятора 32](#_Toc27485631)

[Список используемых источников 34](#_Toc27485632)

[Приложение А 35](#_Toc27485633)

[Приложение Б: 36](#_Toc27485634)

[Приложение В: 37](#_Toc27485635)

[Приложение Г: 45](#_Toc27485636)

[Приложение Д: 47](#_Toc27485637)

[Приложение Е: 48](#_Toc27485638)

[Приложение Ж: 49](#_Toc27485639)

[Приложение З: 50](#_Toc27485640)

[Приложение И: 52](#_Toc27485641)

# Введение

В данном курсовом проекте была поставлена задача разработать собственный язык программирования, а также транслятор для него. Язык получил название EPA-2019.

Задание на курсовой проект можно разделить на следующие подзадания:

* Разработка спецификации для языка EPA-2019 (глава 1);
* Разработка структуры транслятора (глава 2);
* Разработка лексического анализатора (глава 3);
* Разработка синтаксического анализатора (глава 4);
* Разработка семантического анализатора (глава 5);
* Преобразование выражений (глава 6);
* Генерация кода (глава 7);
* Тестирование транслятора (глава 8);
* Разработка и тестирование интерпретатора (глава 9).

**Плюсы данного языка:**

* Существуют функции, позволяющие работать со строками.
* В языке реализован вызов функций, так же проверка типов параметров к ней.
* В языке существует условный оператор IF.
* В языке реализованы операторы сравнения.

**Минусы данного языка:**

* Отсутствует тип данных BOOL.
* В параметры функции могут передаваться только идентификатора.

# 1. Спецификация языка программирования

# 1.1 Характеристика языка программирования

Язык программирования EPA-2019 является процедурным, универсальным, строго типизированным, не объектно-ориентированным, компилируемый. Язык допускает использование 3 типов данных: строковый, с плавающей точкой целочисленный. Все переменные являются локальными. Для работы с численными данными разрешено использование деления, умножения, вычитания и сложения.

# 1.2 Алфавит языка

Символы, разрешенные к использованию при написании кода: латинские символы размером один байт, кодировки Windows-1251. Данная таблица приведена на Рисунке 1. Русский язык запрещён. В литералах, разрешено использовать десятичные цифры и алфавит латиницы.

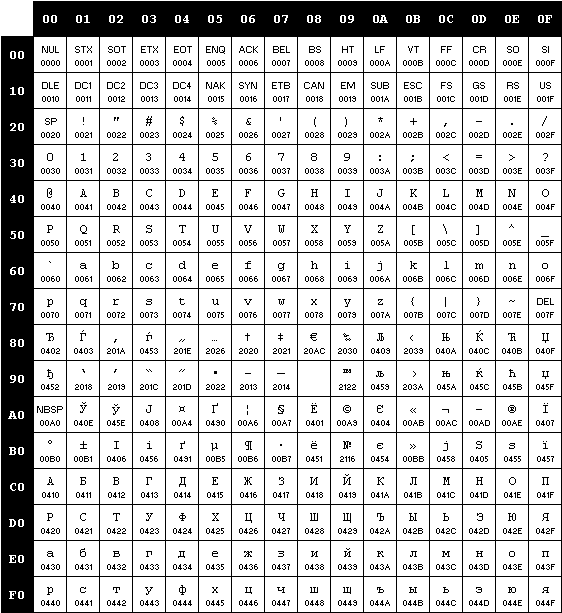


Рисунок 1 - Основная таблица Windows-1251

Символы, используемые на этапе выполнения: [a…z], [A…Z], [0…9]. А также спецсимволы: { } ( ) ; \* + - / = , ’ [ ] пробел.

# 1.3 Применяемые сепараторы

В языке EPA-2019 используются сепараторы, представленные в табл. 1.1*.* Пробел допускается везде кроме идентификаторов и ключевых слов.

Таблица 1.1 – Сепараторы, применяемые в языке EPA-2019

|  |  |
| --- | --- |
| Сепаратор | Описание |
| ; - точка с запятой | Разделитель конструкций |
| {} - фигурные скобки | Программный блок |
| () - круглые скобки. | Приоритетность операций |
| , - запятая | Разделитель параметров функции |

Таким образом, сепараторы данного языка совпадают с сепараторами схожего по синтаксису языка C++.

# 1.4 Применяемые кодировки

Кодировка, используемая для написания программ на языке EPA-2019 - стандартная кодировка Windows-1251 представленная на рисунке 1.

# 1.5 Типы данных

В данном языке можно использовать 3 типа данных, которые описаны в таблице 1.2.

Таблица 1.2 – Основные типы данных языка EPA-2019

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип данных | Краткое описание | Занимаемое место в памяти | Границы | Значение при инициализации: |
| short | Целое значение | 2 байта | От 0 до 32767 | 32767 |
| string | Строка | 256 байт | От 1 до 255  разрешенных символов | Пустая строка, длина строки 0 |

# 1.6 Преобразование типов данных

В языке EPA-2019 преобразование данных запрещено, так как язык является строго типизированным.

# 1.7 Идентификаторы

В языке EPA-2019 идентификаторы должны быть составлены только из символов нижнего регистра английского алфавита. Типы идентификаторов: имя переменной или функции, литерал, параметр, имя стандартной функции. Идентификатор составляется из букв английского алфавита от 1 до 8 символов, без пробелов. Максимальная длина идентификатора равна 8 символам, в противном случае урезается до указанной длины. Идентификатор не может совпадать с ключевыми словами.

# 1.8 Литералы

Таблица 1.3 – Литералы в языке EPA-2019

|  |  |  |
| --- | --- | --- |
| Тип литерала | Допустимые символы | Допустимые значения: |
| short | Цифры от 0 до 9 | От 0 до 32767 |
| str | Разрешенные символы | От 1 до 255 символов |

Строковый тип – комбинация символов, которая может содержать символы латинского алфавита, а также символы пробела, цифр, спец символов (исключая символ двойных кавычек), ограниченных с помощью двойных кавычек с обеих сторон.

Целочисленный тип – комбинация из цифр от 0 до 9.

# 1.9 Объявление данных и область видимости

В языке EPA-2019 объявление данных начинается с ключевого слова def, указывается тип данных и имя идентификатора.

Пример: def short ch, def str stroka;

Область видимости: сверху вниз, параметры внутри функции, объявления внутри функции видны только внутри функции, объявления переменных за пределами функций и главной функции не предусмотрены.

# 1.10 Инициализация данных

Инициализация переменной происходит после её объявления. Инициализация переменной в момент объявления запрещена.

Например: def str stroka; stroka = “слово”; def short ch; ch = 5;

# 1.11 Инструкция языка

В языке EPA-2019 предусмотрены следующие инструкции. Представлены в таблице 1.4.

Таблица 1.4 - Инструкции языка EPA-2019

|  |  |
| --- | --- |
| Инициализация  переменной | Имя переменной = значение; |
| Создание внешней функции | function идентификатор (тип данных идентификатор, …)  Область видимости сверху вниз (по принципу С++). Все переменные являются локальными. |
| Главная функция | glavn  {  …  } |
| Возврат из подпрограммы | back идентификатор или литерал; |
| Вывод данных | write(идентификатор); |
| Условие | if( условие )  {  …  } |

# 1.12 Операции языка

В языке EPA-2019 предусмотрены следующие операции с данными. Приоритетность операций определяется с помощью (). Операции представлены в таблице 1.5.

Таблица 1.5 - Операции языка EPA-2019

|  |  |
| --- | --- |
| Операция | Описание |
| + | бинарный, арифметическое суммирование, (short + short); |
| - | бинарный, арифметическое вычитание, (short - short); |
| \* | бинарный, арифметическое умножение, (short \* short); |
| / | бинарный, арифметическое целочисленное деление, (short / short); |
| < | Арифметическое сравнение (Знак меньше),  (short < short) |
| > | Арифметическое сравнение (Знак больше),  (short > short) |
| ^ | Арифметическое сравнение (Знак больше  либо равно) , (short ^ short) |
| % | Арифметическое сравнение (Знак меньше либо равно),  (short % short) |
| ~ | Арифметическое сравнение (Знак сравнения),  (short ~ short) |
| | | Арифметическое сравнение (Знак не равно),  (short | short) |

# 1.13 Выражения и их вычисления

Вычисление выражений осуществляется по некоторым правилам, которые приведены ниже:

1. Выражение записываются в одну строку;

2. Допускается использовать скобки (приоритет);

3. Использование двух операторов подряд не допускается;

4. Выражение может содержать вызов функции.

# 1.14 Программные конструкции языка

В языке программирования EPA-2019 предусмотрена одна главная функция и внешние функции. Программные конструкции представлены в Таблице 1.6.

Таблица 1.6 - Программные конструкции EPA-2019

|  |  |
| --- | --- |
| Внешняя функция | Тип данных function идентификатор (тип данных идентификатор, … )  {  …  back идентификатор / литерал;  }  Область видимости сверху вниз (по принципу С++). |
| Главная функция | glavn  {  /программный блок/  }  Область видимости сверху вниз. Все переменные являются локальными. |

# 1.15 Область видимости идентификаторов

Сверху вниз, параметры внутри функции, объявления внутри функции видны только внутри функции, объявления за пределами функций и главной функции запрещены.

# 1.16 Семантические проверки

Основные семантические правила проверяемые на этапах работы транслятора, представлены в Таблице 1.7. Набор правил, приведенный в данной таблице, облегает дальнейшую обработку программы, написанной на языке EPA-2019.

Таблица 1.7 - Семантические правила

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Проверка на наличие точки входа в программу glavn |
| 2 | Усечение идентификаторов до 8 символов (если требуется) |
| 3 | Изначально проверка на ключевые слова, а затем на идентификатор. Не допускаются идентификаторы, совпадающие с ключевыми словами |
| 4 | Названия функций не должны повторяться |
| 5 | Нет повторяющихся объявлений идентификаторов в одной области видимости |
| 6 | Переменные, используемые в функции, должны быть предварительно объявлены, после чего инициализированы |
| 7 | Тип возвращаемого значения и тип функции должны соответствовать |
| 8 | Проверка на максимальное и минимальное значение целочисленного литерала |
| 9 | Проверка на максимальную длину строкового литерала |

# 1.17 Распределение оперативной памяти на этапе выполнения

Транслированный код использует 2 области памяти.

В область констант заносятся литералы. В область глобальных переменных заносятся переменные и параметры, локальная область видимости в исходном коде определяется за счет использования правила именования данных идентификаторов.

# 1.18 Стандартная библиотека и её состав

В EPA-2019 присутствует стандартная библиотека Sys:LB. Подключение стандартной библиотеки обязательно перед использованием функций. Пример: Sys:LB; Библиотека представлена в Таблице 1.8.

Таблица 1.8 - Стандартная библиотека

|  |  |
| --- | --- |
| Функция | Описание |
| short strlen (str идентификатор); | Находит длину строки. Применима только для идентификаторов типа переменная, тип данных str. |
| short compare (str идентификатор, str идентификатор); | Сравнение строк. Применима только для идентификаторов тип данных str |

# 1.19 Ввод и вывод данных

Ввод данных в языке EPA-2019 не предусмотрен.

Вывод происходит с помощью оператора write.

Например: write(идентификатор), write(“текст”).

# 1.20 Точка входа

В языке EPA-2019 точкой входа является функция glavn. Наличие данной функции обязательно. В программе должна быть только одна точка входа. Имена идентификаторов не должны совпадать с именем точки входа.

# 1.21 Препроцессор

При трансляции языка EPA-2019 использование препроцессора в данной программе не предусмотрено.

# 1.22 Соглашения о вызовах

Отсутствуют.

# 1.23 Объектный код

Язык программирования EPA-2019 транслируется на JavaScript (ECMAScript 2018).

# 1.24 Классификация сообщений транслятора

Транслятор, в ходе своей работы, генерирует сообщения, которые информируют пользователя о допущенных ошибка. Все сообщения транслятора разделены на интервалы, в зависимости от того на каком этапе была обнаружена ошибка. Все интервалы представлены в таблице 1.9.

Таблица 1.9 - Сообщения транслятора

|  |  |
| --- | --- |
| Интервал | Описание |
| 0,1,100,104,  110,112 | Ошибки системы |
| 102 | Ошибки входного кода |
| 5,105,111,  117,124,125 | Ошибки на этапе лексического анализа |
| 3,4,6,7,101,  106-109  114,116,118-  123,126,127,  605-614 | Ошибки на этапе семантического анализа |
| 8,600-605 | Ошибки на этапе синтаксического анализа |

# 1.25 Контрольный пример

Контрольный пример, демонстрирующий все возможности языка EPA-2019 представлен в Приложении А.

# 2 Структура транслятора

# 2.1 Компоненты транслятора и принципы взаимодействия

Транслятор языка EPA-2019 разделён на отдельные части, которые взаимодействуют между собой и выполняют отведённые им функции. Алгоритм выполнения и описание каждой части приложения представлено в Таблице 2.1.

Таблица 2.1 - Структура транслятора

|  |  |
| --- | --- |
| Наименование подпрограммы | Предназначение |
| Лексический анализатор | Обрабатывает входной файл исходного кода, проверяя его на разрешённые, запрещённые и игнорируемые символы. Преобразуя исходный код в более простой, с помощью замены длинных слов на лексемы, состоящие из одного символа, что упрощает последующую работу с кодом. Каждая лексема несёт в себе многочисленную информацию: имена идентификаторов, тип данных, тип переменной и так далее. После выполнения данной подпрограммы на выходе мы получаем таблицу лексем и таблицу идентификаторов. |
| Синтаксический анализатор | С помощью синтаксического анализатора проверяется правильность написанных конструкций по заданной грамматике. |
| Семантический анализатор | Реализован как отдельная часть. Нужен для того, чтобы проверить все правила, которые невозможно проверить на этапах лексического и синтаксического анализаторов |
| Транслятор кода | Связан с работой лексического анализатора. Посредством полученных данных: таблицы лексем и идентификаторов, код на языке EPA-2019 транслируется в код на языке JavaScript (ECMAScript 2018) |

Каждый этап программы имеет взаимосвязь, которую мы можем наблюдать на рис. 2.

Продолжение таблицы 2.1

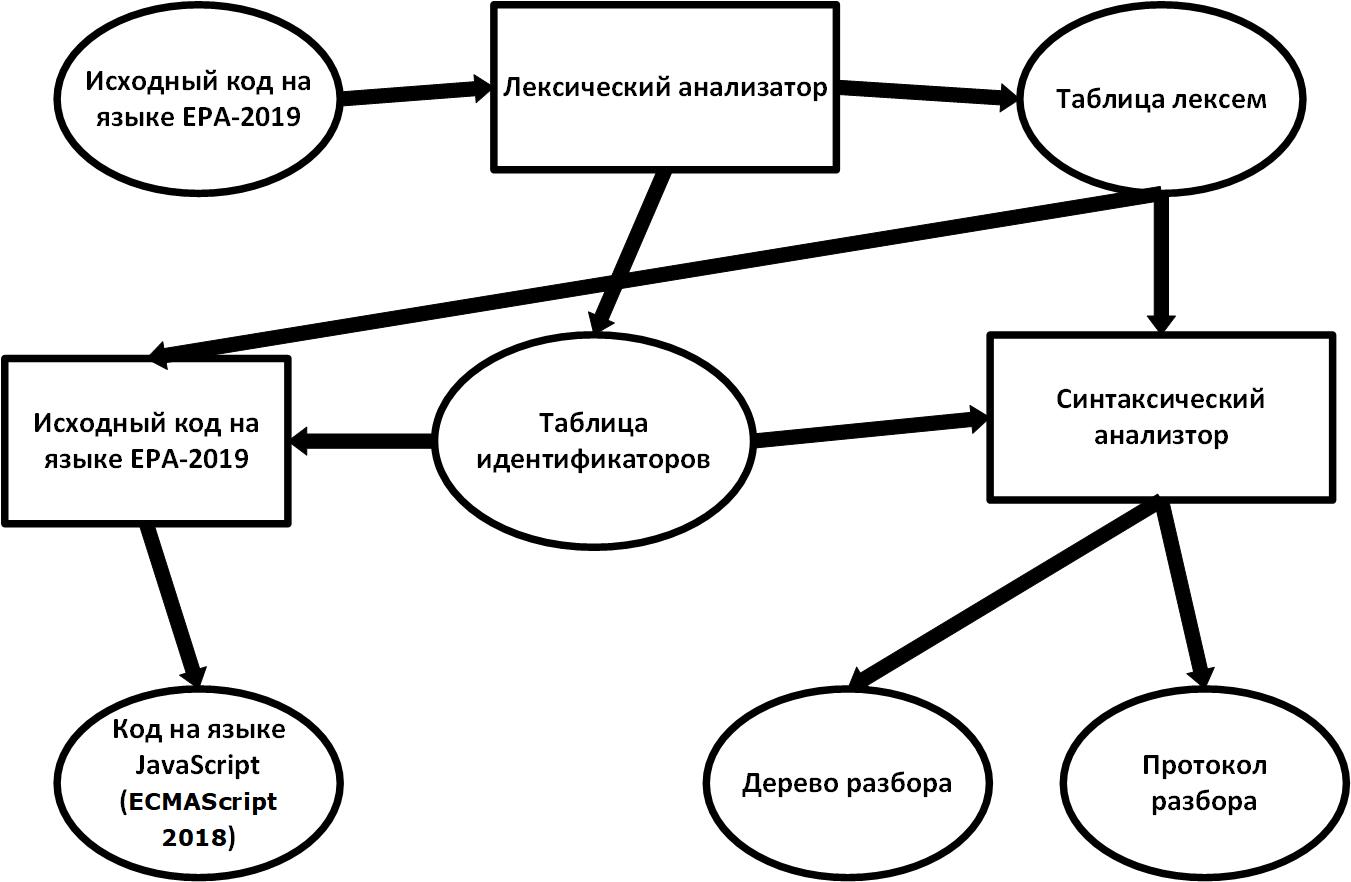


Рисунок 2

# 2.2 Перечень параметров транслятора входных

Для указания запуска параметров запуска транслятора следует указывать параметры командной строки, которые представлены и описаны в Таблице 2.2.

Таблица 2.2 - Параметры транслятора

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Правило указания** | **Описание параметра** |
| -in: | -in:[полный путь к файлу] | Предназначен для определения местонахождения файла с исходным кодом, для проверки и разбора его на лексическом анализаторе.  Указание данного параметра обязательно. |
| -log: | -log:[полный путь к файлу] | Устанавливается местонахождение и имя файла, содержащего информацию о ходе работы транслятора.  Указание данного параметра необязательно. |

# 2.3 Перечень протоколов формируемых транслятором

Содержание логируещего файла может быть различно в зависимости от параметров командной строки, которые представлены в Таблице 2.2*.* Обязательность, а также описание информации, выводимой в лог-файл можно посмотреть в Таблице2.3.

Таблица 2.3 - Протоколы транслятора

|  |  |  |
| --- | --- | --- |
| **Тип информации** | **Описание информации** | **Обязательность вывода** |
| Дата и время трансляции | Выводит дату и время в формате: дд.мм. гг., чч: мм: сс. | Обязательно |
| Параметры командой строки | Выводит информацию об указанных параметрах командной строки. | Обязательно |
| Полная таблица лексем | Выводит таблицу лексем с информацией к каждой лексеме в log.txt. | Необязательно |
| Упрощённая таблица лексем | Выводит только лексемы без дополнительной информации. | Необязательно |
| Таблица идентификаторов | Выводит таблицу идентификаторов с дополнительной информацией. | Необязательно |
| Трассировочная информация синтаксического анализа | Выводит полную информацию о разборе таблицы лексем на синтаксическом анализаторе. | Необязательно |
| Правила разбора | Выводит правила по которым осуществился разбор исходного кода. | Необязательно |
| Польская нотация | Выводит польскую нотацию к конечной версии таблицы лексем. | Необязательно |

Продолжение таблицы 2.3

# 3 Разработка лексического анализатора

# 3.1 Структура лексического анализатора

Входными данными для лексического анализатора является массив цепочек, предварительно сформированный на первичной обработке исходного кода программы. Выходными данными являются таблица лексем, таблица идентификаторов, а также протокол работы. Структура лексического анализатора EPA-2019 представлена на Рисунке 3.

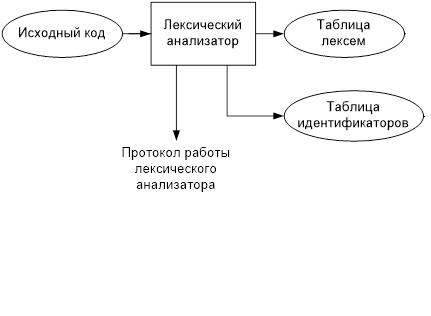


Рисунок 2 - Структура лексического анализатора EPA-2019

# **3.2 Контроль входных символов**

Таблица входных символов представлена на рис. 1. Реализация таблицы на языке C++ представлена в приложении Б*.* Обозначения символов в таблице представлено в Таблице 3.1.

Таблица 3.1 - Контроль входных символов

|  |  |
| --- | --- |
| **Символы** | **Значение в таблице символов** |
| () \* + - = , / \ % | SEP |
| Двойные кавычки | QUOTE |
| Запрещённый | F |
| Разрешённый | T |
| Игнорируемый | I |
| Пробел | PR |

# 3.3 Удаление избыточных символов

При считывании из файла с исходным кодом в случае, если считанный символ оказался пробелом, тогда мы проверяем предыдущий символ, если предыдущий символ являлся пробелом, значит текущий пробел мы просто игнорируем и не заносим в массив.

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

Ключевые слова EPA-2019: def, short, str, function, back, strlen, compare, write, glavn, while, if. Сепараторы EPA-2019 представлены в Таблице 1.1. Графы переходов представлены в Приложении В.

Для каждой фразы также соответствует автомат, по которому происходит разбор выражения. Автомат является – детерминированным, то есть имеет конечное состояние. Проверка происходит следующим образом: на каждый автомат в массиве подаётся фраза и с помощью графа переходов происходит разбор, если разбор выполнен, то происходит заполнение таблицы лексем и при необходимости таблицы идентификаторов.

Каждой фразе соответствует отдельный символ, называемый лексемой. Соответствие фраз с такими лексемами представлено в Таблице 3.2.

Таблица 3.2 - Лексемы

|  |  |  |
| --- | --- | --- |
| **Фрагмент** | **Лексема** | **Примечание** |
| short  str | t | Тип данных |
| идентификатор | i | (переменная, функция или параметр) |
| литералы | l | Данные |
| function | f | функция |
| back | b | Возврат значения из функции |
| write | o | Вывод |
| compare | C | Сравнение строк. |
| strlen | n | Нахождение длины строки |
| glavn | g | Главная функция |
| ; | ; | Сепаратор |
| { | { | Начало тела внешней функции |
| } | } | Конец тела внешней функции |
| ( | ( | Начало перечисления параметров функции |
| ) | ) | Конец перечисления параметров функции |
| = | = | Оператор присваивания  (или сравнивания в if) |
| +  -  \*  / | +  -  \*  / | Арифметические операторы |
| , | , | Сепаратор параметров |
| Sys:LB | B | Встроенная библиотека |
| def | d | Создание переменной |
| while | w | Оператор цикла |
| if | u | Оператор условия |
| <  >  ^  %  ~  ne | <  >  ^  %  ~  | | Операторы сравнения |

# 3.5 Основные структуры данных

Продолжение таблицы 3.2

Код С++ с описанием структур таблиц лексем и идентификаторов, приведён в приложении Г.

# 3.6 Принцип обработки ошибок и их перечень

При обнаружении ошибки во время работы транслятора, вызывается функция получения ошибки, в которую передается, в зависимости от места возникновения ошибки, следующая информация: код ошибки, номер строки в коде, номер позиции в строке или только код ошибки.

# 3.7 Структура и перечень сообщений лексического анализатора

Для обработки ошибок лексический анализатор использует таблицу с сообщениями, структура которой представлена в Приложении Г. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. При возникновении сообщения, лексический анализатор продолжает работу с исходным кодом, проверяет его до конца и выводит оставшиеся сообщения об ошибках, если они существуют.

Перечень сообщений представлены в Таблице 3.3.

Таблица 3.3 - Сообщения лексического анализатора

|  |  |  |
| --- | --- | --- |
| Код | Сообщение | Пояснение |
| 11 | Отсутствие главной функции | Не удалось найти точки входа в программу – **glavn** функции. |
| 12 | Попытка переопределения | Попытка изменить уже существующий идентификатор. |
| 13 | Отсутствует предварительное определение | Использование идентификатора без его предварительного объявления |
| 14 | Превышен размер таблицы лексем | Переполнение таблицы лексем |

# 3.8 Параметры лексического анализатора и режимы его работы

Входные параметры используются для вывода результата работы лексического анализатора. Представлены в Таблице 3.4.

Таблица 3.4 - Входные параметры лексического анализатора

|  |  |
| --- | --- |
| Параметр | Описание |
| -LT | Параметр для вывода таблицы лексем в файл “log.txt” |
| -IT | Параметр для вывода таблицы идентификаторов в файл “log.txt” |

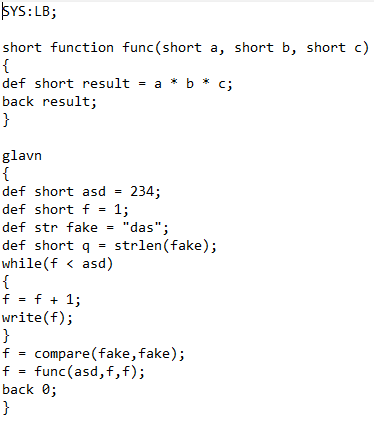
# 3.9 Алгоритм лексического анализатора

После разбиения текста из файла с исходным кодом на слова, для каждого слова подбирается автомат способный его разобрать, в случае если такой автомат существует, тогда цепочка будет разобрана, иначе ошибка. Далее лексический анализатор анализирует лексему, соответствующую данному слову, и выполняет действия, описанные для данной лексемы. Лексический анализатор продолжает работать пока не будет разобрано последнее слово.

# 3.10 Контрольный пример

В результате работы лексического анализатора мы получим таблицу лексем, таблицу идентификаторов и информацию об ошибках. Текст таблицы лексем и таблицы идентификаторов представлен в Приложении Г.

Текст входного Файла:



# 4 Разработка синтаксического анализатора

# 4.1 Структура синтаксического анализатора

Структура работы синтаксического анализатора и входные-выходные данные представлены на Рисунке 4.

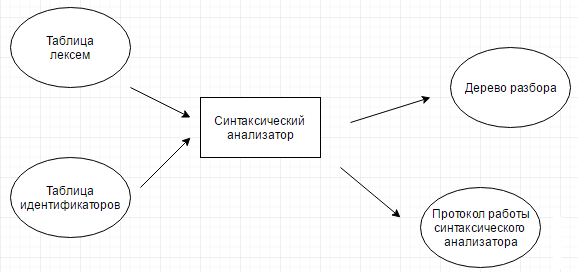


Рисунок 4 - Синтаксический анализатор

# 4.2 Грамматика языка

Грамматика для синтаксического разбора языка с представлена в Таблице 4.1. Для описания языка, разбираемого синтаксического анализатором, применяют грамматики типа 2 – контекстно-свободные грамматики.

Грамматика типа 2:  - контекстно-свободная грамматика, где T – множество терминалов, N – множество нетерминалов, P – правила перехода, S – стартовый символ. Описание терминалов представлено в Таблице 3.2.

Правила имеют вид: , где , ,  - словарь грамматики , где Т-правило нашей грамматики.

Таблица 4.1 - Правила перехода нетерминальных символов

|  |  |
| --- | --- |
| Нетерминал | Правила |
| S – структура программы | B;S | tfi(F){N};S | t,f,i(F){N}S; |  t,f,i(F){N}S | t,f,i(){N}S | g{N}| |
| N – конструкции внутри функции | dti;N | dti=E;N | i=E;N | o=E;N | i=E;| bE; | dtp(F);N | dtfs(F);N | dte(F);N|i=C(E,E);N|  dti=C(E,E);N| w(E){}N|u(E){N} | w(E){N}N | w(E){} | u(e){N} |
| F – параметры функции | ti | ti,F |
| E – выражения | i | l | | iE | iM | lM | (W) | (E) | (E)M | s(W) | e(W) | p(W) | e(W)M | s(W)M| p(W)M| (W)M| i(W) |
| W – параметры функции при вызове | l | i | i,W | l,W |
| M – знаки | \*E | /E |+E | -E | +EM | -EM |\*EM | /EM|<E|>E|~E|^E|%E |

# 4.3 Построение конечного магазинного автомата

Продолжение таблицы 4.1

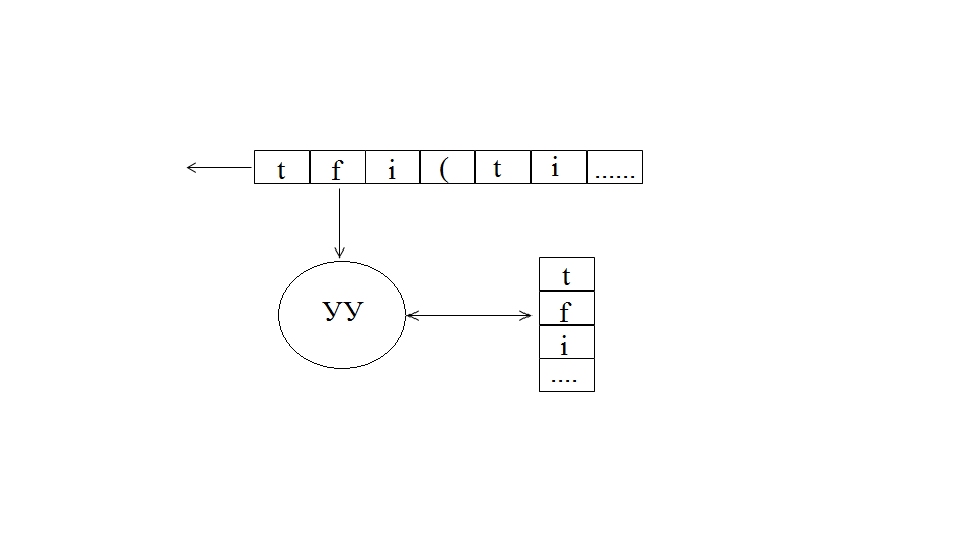
Конечный автомат с магазинной памятью можно представить в виде семёрки , где М – автомат, Q – множество состояний, V – алфавит входных символов, Z – алфавит магазина,  - функция переходов,  – начальное состояние автомата,  – начальное состояние магазина, F – множество конечных состояний. Автомат с магазинной памятью представлен на Рисунке 5. 

Рисунок 5 - Автомат с магазинной памятью.

Принцип работы автомата следующий:

* В магазин записывается стартовый символ;
* На основе полученных раннее таблиц формируется входная лента;
* Запускается автомат;
* Выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
* Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется с ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала;
* Если в магазине встретился нетерминал, переходим к пункту 4;
* Если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе вызывается ошибка.

Структура магазинного автомата представлена в виде структуры С++ (описание автомата находится в зоне комментариев) в приложении Ж: Магазинный автомат.

# 4.4 Основные структуры данных

Код C++ с описанием структур данных, описывающих грамматику представлен в Приложении Е*.*

# 4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора основан на работе автомата с магазинной памятью. Работа данного автомата описана в пункте 4.3.

# 4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен в Таблице 4.2.

Таблица 4.2 - Перечень сообщений синтаксического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 600 | Неверная структура программы. |
| 601 | Ошибка в параметрах функции. |
| 602 | Ошибочный оператор. |
| 603 | Ошибка в выражении. |
| 604 | Ошибка в параметрах вызываемой функции. |
| 8 | Ошибка в синтаксическом анализе. Неизвестное правило разбора! Сверьтесь с разрешёнными комбинациями цепочек! |

Продолжение таблицы 4.2

# 4.7 Параметры синтаксического анализатора и режимы его работы

Для управления результата работы синтаксического анализатора используются входные параметры, описанные в пункте 2.2Перечень входных параметров транслятора в Таблице 2.2.

# 4.8 Принцип обработки ошибок

При возникновении ошибок синтаксический анализатор пытается вернутся на предыдущее сохраненное состояние назад, если это возможно и повторить шаг по следующей цепочке правил, в случае невозможности вызывается специализированная ошибка для правила, до которого максимально добрался анализатор. Ошибки выдаваемы синтаксическим анализатором описаны в Приложении З: Таблица ошибок.

# 4.9 Контрольный пример

Протокол разбора и дерево разбора представлены в приложении И: Результат работы синтаксического анализатора.

# 5 Разработка семантического анализатора

# 5.1 Структура семантического анализатора

Проверка на ошибки в исходном коде производится как одновременно с работой синтаксического и лексического анализаторов, так и в отдельной части транслятора. Семантические правилаязыка EPA-2019 представлены в гл. 1. п. 1.16.

# 5.2 Функции семантического анализатора

Семантический анализатор выполняет проверку на основные правила языка (семантики языка), которые представлены в гл. 1. п. 1.16*.* Таблица ошибок представлена в виде фрагмента кода C++ в приложении З: Таблица ошибок.

# 5.3 Принцип обработки ошибок

В случае возникновения ошибок, вызываем функцию получения ошибки, которая принимает обязательным параметром код ошибки в таблице сообщений. Затем производится вывод ошибки в поток лог-файла, а также ошибка выводится в консоль. Структура данных C++ в которой хранится информация об ошибках представлена в приложении З: Таблица ошибок*.*

# 5.4 Структура и перечень сообщений семантического анализатора

Со структурой и перечнем сообщений, можете ознакомится в приложении К: Структура и перечень сообщений семантического анализа.

# 5.5 Контрольный пример

Контрольные примеры, предназначенные для проверки обработки всех возможных ошибок, представлены в виде текстовых файлов с исходным кодом на языке EPA-2019, которые прилагаются вместе с файлами решения под среду разработки MS Visual Studio 2017. Также несколько примеров которые демонстрируют работу обработчика ошибок представлены в приложении Д.

# 6 Преобразование выражений

# 6.1 Выражения, допускаемые языком

В языке EPA-2019 допускаются вычисления выражений исключительно с целочисленными типами данных. Вычисление выражений в языке EPA-2019 происходит c преобразованием в польскую запись. Приоритет операций представлен в Таблице 6.1.

Таблица 6.1 - Приоритеты операций

|  |  |
| --- | --- |
| Приоритет | Операция |
| 1 | ( |
| 1 | ) |
| 2 | \* |
| 2 | / |
| 3 | + |
| 3 | - |

# 6.2 Польская запись и принцип её построения

Польская запись - это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: прямая и обратная, также известные как префиксная и постфиксная. Отличие их от классического, инфиксного способа заключается в том, что знаки операций пишутся не между, а, соответственно, до или после аргументов.

Суть алгоритма заключается в следующем. Просматривается исходная строка символов S слева направо, операнды переписываются в выходную строку В, а знаки операций заносятся в стек, который первоначально пуст, на основе следующих правил:

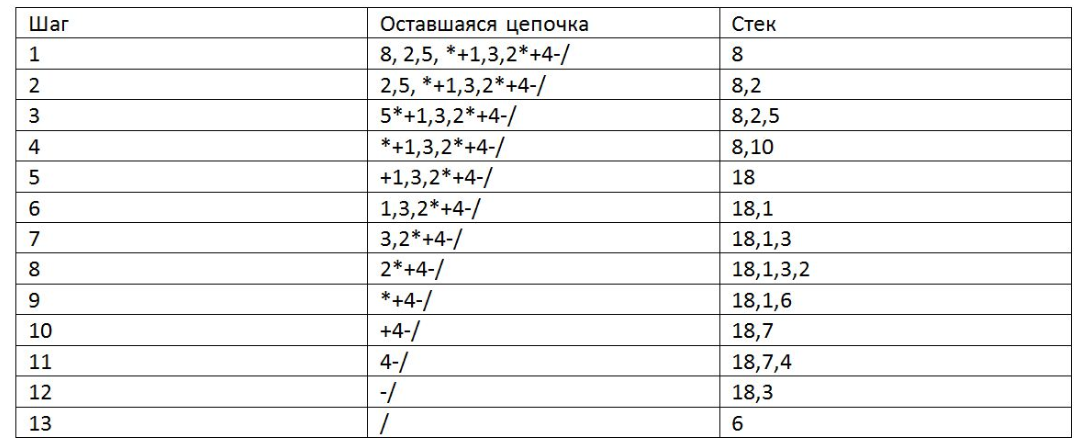
* Если в строке S встретился операнд, то помещаем его в строку В;
* Если в S встретилась открывающая скобка, то помещаем ее в стек;
* Если в S встретилась закрывающая скобка, то выталкиваем из стека в строку В все операции до открывающей скобки, саму отрывающую скобку также извлекаем из стека; обе скобки (открывающая и закрывающая) игнорируются;
* Если в S встретилась операция Х, то выталкиваем из стека все операции, приоритет которых не ниже Х, после чего операцию Х записываем в стек;
* При достижении конца строки S, если стек не пуст, переписываем его элементы в выходную строку В.

# 6.3 Примеры преобразования выражений

Примеры преобразования выражений к польской записи представлены в Таблице 6.2.

Таблица 6.2 - Примеры выражений в польской записи

|  |  |
| --- | --- |
| Выражение | Выражение в польской записи |
| 2\*(3+7) | 237+\* |
| 2+3 | 23+ |
| x+x\*(3+x) | xx3x++\* |
| ((ab+5)/6-4)\*2 | ab 5 + 6 / 4 - 2 \* |

Пример разбора выражения ((8+2\*5)/(1+3\*2-4)) в стеке 

# 7 Генерация кода

Генерация кода – заключительный этап работы транслятора. Результатом данного этапа будет код, сгенерированный для выполнения Java 8 на основе таблицы лексем и таблицы идентификаторов, что является требуемым результатом работы программы.Транслятор кода начинает свою работу только в том случае если код на языке EPA-2019 прошёл предыдущие компоненты транслятора без ошибок. Схема данного этапа изображена на Рисунке 6.

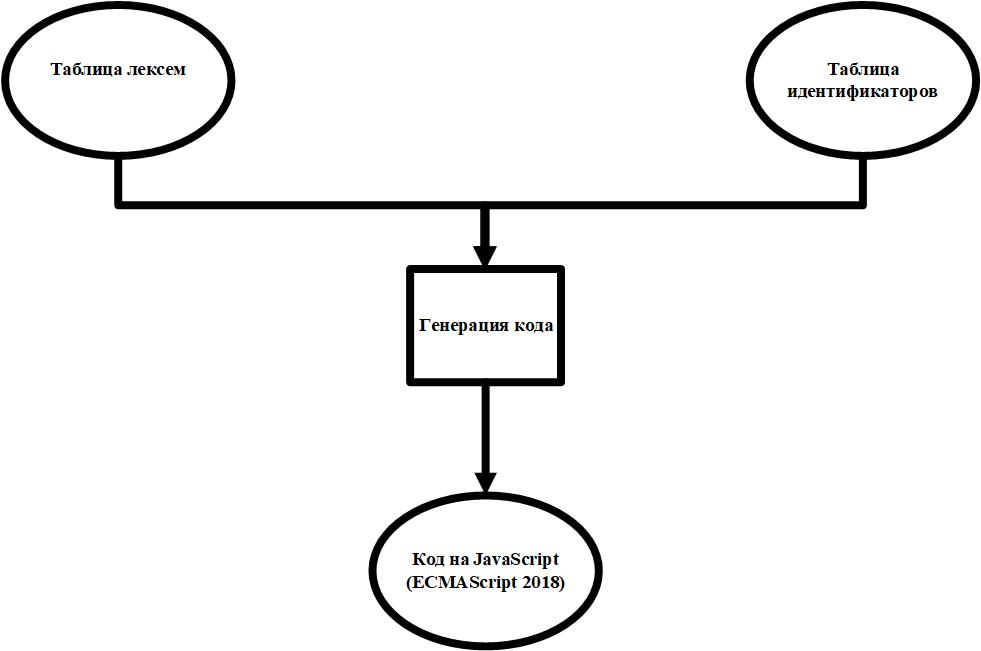


Рисунок 6 – Схема работы генератора кода

На вход подаются по очереди лексемы, и в зависимости от поданной лексемы языка EPA-2019, в файл Gen.html, в который первоначально записывается шаблон на html, а после записывается соответствующее ей слово. После разбора последней лексемы будет получен файл с кодом на JavaScript (ECMAScript 2018). После открытия этого файла, откроется браузер по умолчанию, в котором мы увидим результат выполнения этой программы. В Таблице 7.1 представлены лексемы и соответствующие им слова для записи в файл.

Таблица 7.1 – Трансляция на JavaScript (ECMAScript 2018)

|  |  |
| --- | --- |
| Лексема | Слово |
| { | Переход на новую строку с табуляцией |
| d | var |
| ; | Переход на новую строку |
| B | Запись не производится |
| o | document.writeln |
| g | Запись не производится |
| ( | ( |
| ) | ) |
| } | Переход на новую строку с возвращением в начало |
| + | + |
| - | - |
| \* | \* |
| / | / |
| f | function |
| , | , |
| b | return |
| e | .length |
| u | if |
| | | != |
| = | = |
| ~ | == |
| < | < |
| > | > |
| ^ | >= |
| % | <= |
| t | Запись не проивзодится |
| C | .localeCompare() |

# 8 Тестирование транслятора

Тестирование транслятора EPA-2019 будет выполняться посредством компиляции исходного кода в проекте Visual Studio 2017 и запуска, полученного файла с исходным кодом на языке JavaScript (ECMAScript 2018). Во время тестирования транслятора файл с исходным кодом EPA-2019 будет изменяться для проверки эффективности и работоспособности транслятора в различных ситуациях. В случае успешной компиляции, мы увидим результат компиляции в веб-браузере по умолчанию.

Результат выполнения на Рисунке 7.

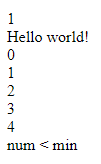


Рисунок 7 – Результат выполнения на JavaScript (ECMAScript 2018)

В случае ошибок в исходном коде, веб-браузер не откроется, а в консоли вы увидите ошибку в консоли.

**Заключение**

В данном курсовом проекте были выполнены поставленные минимальные требования. В ходе работы были изучены этапы компиляции, а также закреплены знания, которые были получены ранее. Также стоит отметить, что данный курсовой проект позволил совместить закрепление знаний сразу по двум языкам программирования, таких как C++ и JavaScript (ECMAScript 2018). При написании приложения были усвоены такие понятия как синтаксический, лексический и семантический анализатор и многие другие.

В итоге был получен примитивный язык программирования EPA-2019, который не имеет сложных конструкций, которые реализованы на сегодняшний день во многих других языках программирования. Финальная версия транслятора состоит из 3232 строк кода.

Окончательная версия языка EPA-2019 включает:

1. 2 типа данных;
2. Поддержка оператора вывода;
3. Возможность подключения и вызова функций стандартной библиотеки;
4. Наличие 4 арифметических операторов для вычисления выражений;
5. Наличие 6 условных операторов;
6. Возможность вызова функции в выражении;
7. Поддерживает двоичную систему счисления;
8. Поддерживает восьмеричную систему счисления;
9. Возможность использовать цикл;
10. Возможность использовать условный блок if;
11. Обработку ошибок;

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении трансляторов, а также основные различия и преимущества тех или иных средств трансляции.

# Список используемых источников

1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.

2. Молчанов, А. Ю. Системное программное обеспечение / А. Ю. Молчанов. – СПб.: Питер, 2010. – 400 с.

3. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.

4. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.

5. Орлов, С.А. Теория и практика языков программирования / С.А. Орлов – 2014. – 689 с.

6. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.

7. Смелов, В. В. Введение в объектно-ориентированное программирование на С++ / В. В. Смелов.– Минск : БГТУ, 2009. - 94с.

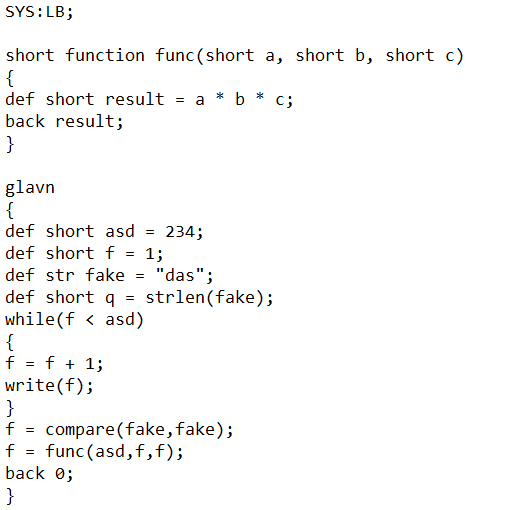
8. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с.

9.Сайт Microsoft [Электронный ресурс] - \_\_stdcall: Режим доступа: <https://msdn.microsoft.com/en-us/library/zxk0tw93.aspx> - Дата доступа: 04.12.2017.

10. ASCII Table and Description [Электронный ресурс] – Режим доступа: <http://www.asciitable.com/> - Дата доступа: 02.12.2017

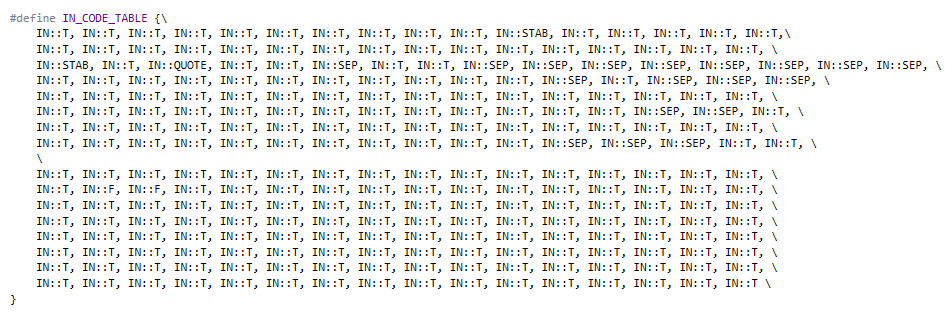
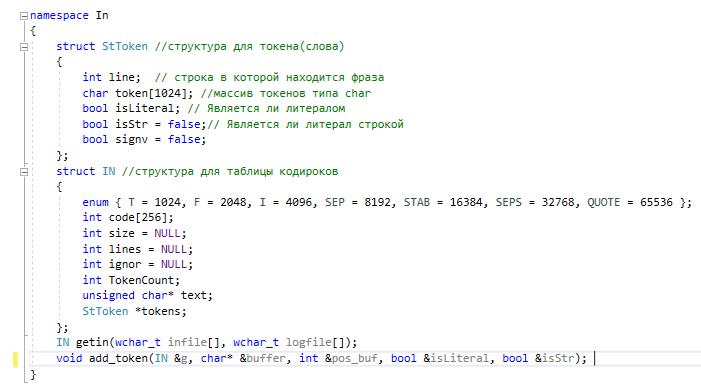
# Приложение А

Контрольный пример



# Приложение Б:

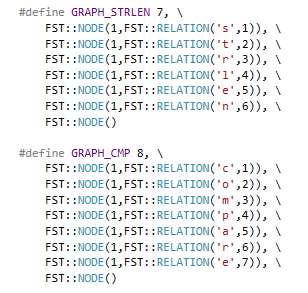
Таблица входных символов.

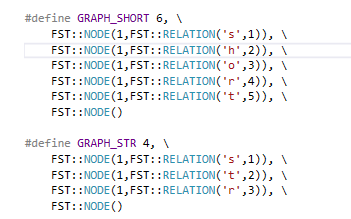
# Приложение В:

Графы переходов.

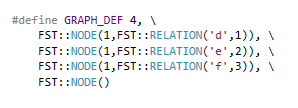
Граф функций стандартной библиотеки



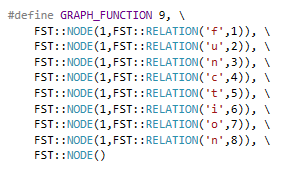
Графы типов данных



Граф объявления



Граф функции



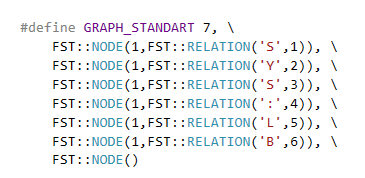
Графы сепараторов



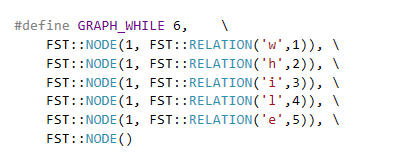
Графы операторов сравнения и присваивания



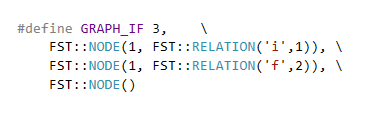
Граф подключения стандартной библиотеки



Граф оператора цикла



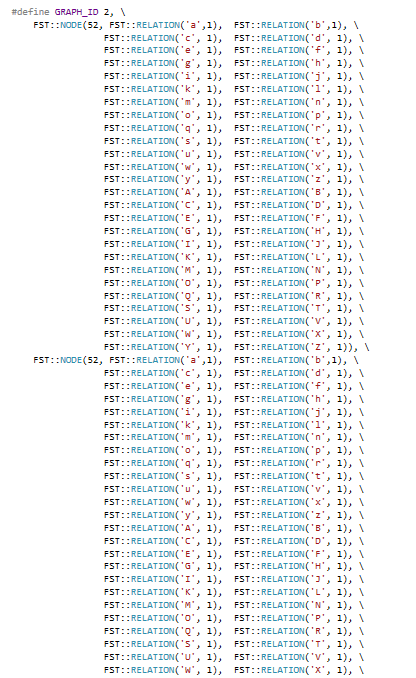
Граф условного оператора



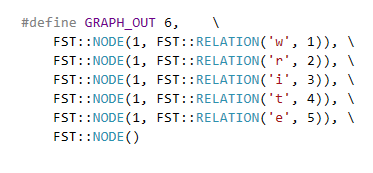
Графы стандартных функций:



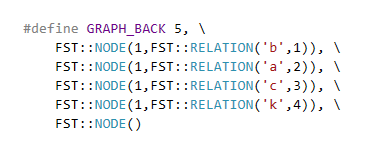
Граф имени идентификатора



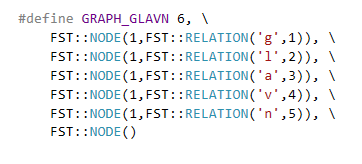
Граф оператора вывода



Граф оператора возврата

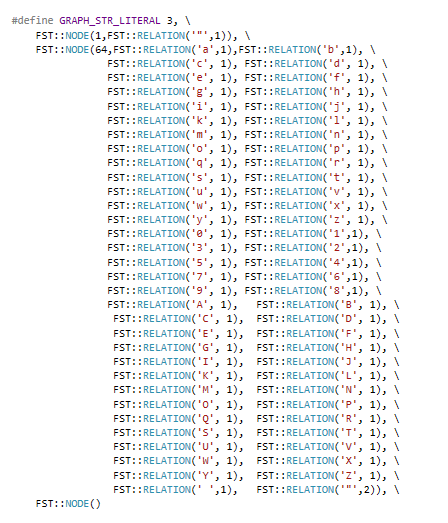


Граф определения главной функции (точки входа)



Графы литералов





# Приложение Г:

Структуры таблицы лексем и таблицы идентификаторов.

Таблица лексем

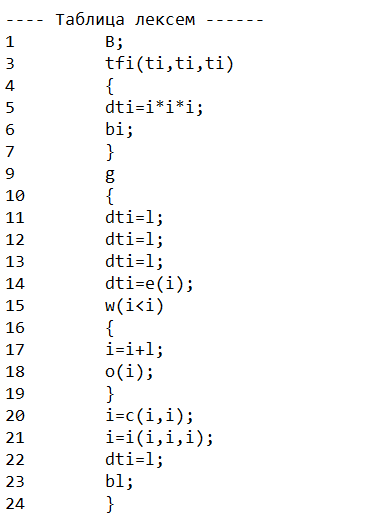
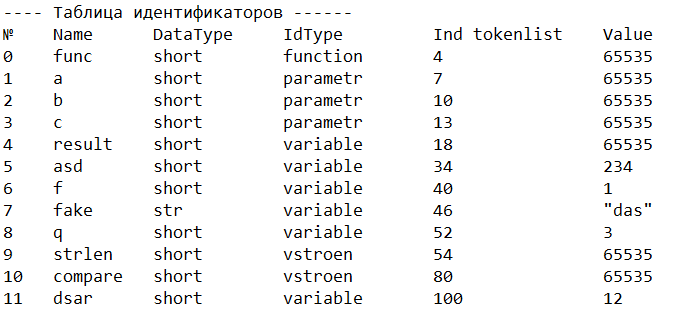
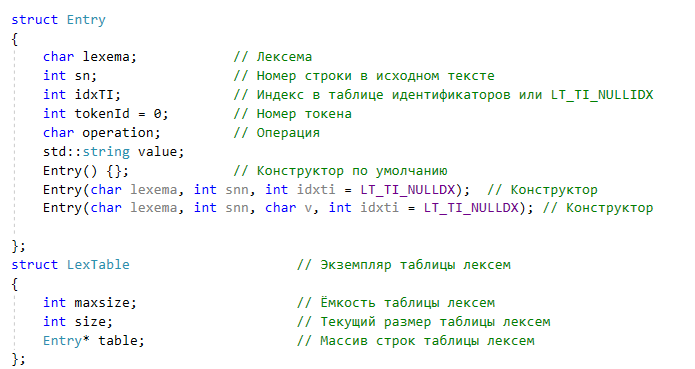
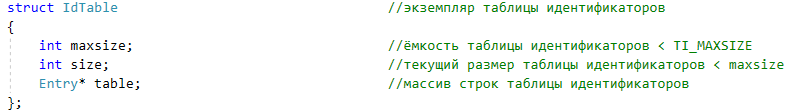


Таблица идентификаторов







# Приложение Д:

Семантические проверки.

1.)



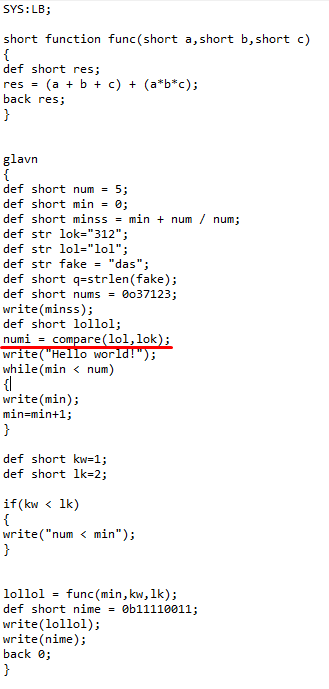


2.)





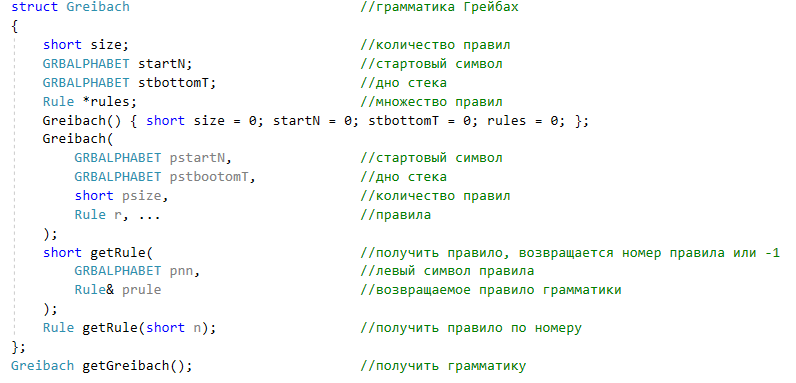
3.)

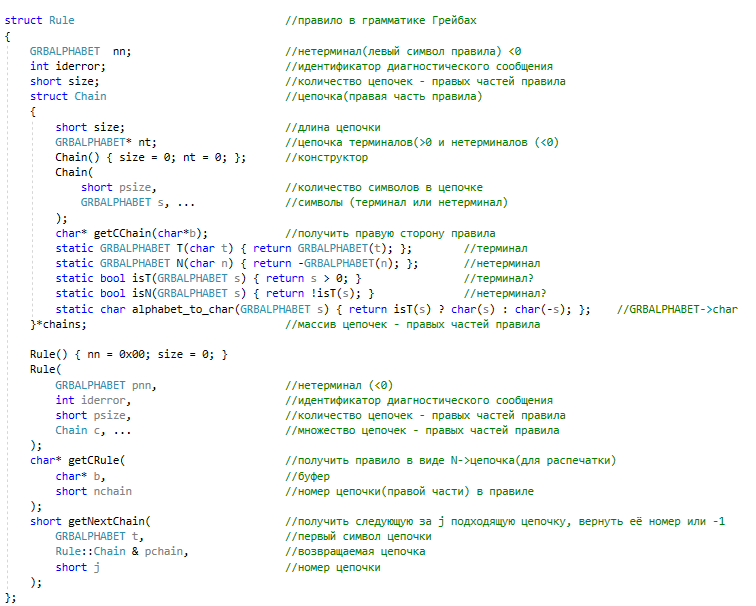




# Приложение Е:

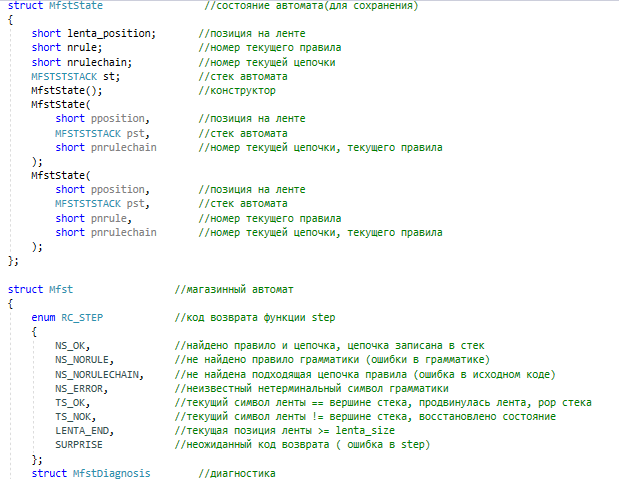
Грамматика.

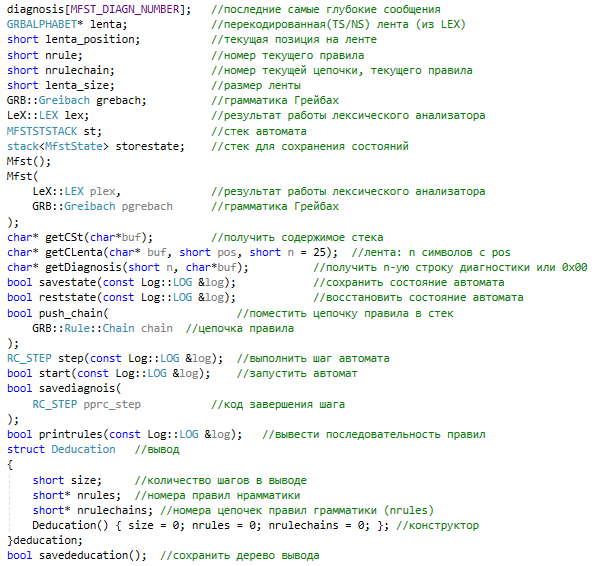




# Приложение Ж:

Магазинный автомат.





# Приложение З:

Перечень ошибок:

ERROR\_ENTRY(0, "[Системная] Недопустимый код ошибки"),

ERROR\_ENTRY(1, "[Системная] Системный сбой"),ERROR\_ENTRY\_NODEF(2),

ERROR\_ENTRY(3, "[Семантическая] Несоответствие типов"),

ERROR\_ENTRY(4, "[Семантическая] Несоответствие присваиваемого типа типу переменной"),

ERROR\_ENTRY(5, "[Лексическая] Неопознанный символ"),

ERROR\_ENTRY(6, "[Семантическая] Выход за предел допустимого значения переменной"),

ERROR\_ENTRY(7, "[Семантическая] Не хватает закрывающей кавычки"),

ERROR\_ENTRY(8, "[Синтаксическая] Ошибка в синтаксическом анализе. Неизвестное правило разбора! Сверьтесь с разрешёнными комбинациями цепочек!"),

ERROR\_ENTRY\_NODEF(9),

ERROR\_ENTRY\_NODEF10(10), ERROR\_ENTRY\_NODEF10(20), ERROR\_ENTRY\_NODEF10(30), ERROR\_ENTRY\_NODEF10(40), ERROR\_ENTRY\_NODEF10(50),

ERROR\_ENTRY\_NODEF10(60), ERROR\_ENTRY\_NODEF10(70), ERROR\_ENTRY\_NODEF10(80), ERROR\_ENTRY\_NODEF10(90),

ERROR\_ENTRY(100, "[Системная] Параметр -in должен быть задан"),

ERROR\_ENTRY(101, "[Семантическая] Отсутствие главной функции"),

ERROR\_ENTRY(102, "[Входящая] Превышена длина строкового литерала "),

ERROR\_ENTRY(103, "[Лексическая] Попытка переопределения"),

ERROR\_ENTRY(104, "[Системная] Превышена длина входного параметра"),

ERROR\_ENTRY(105, "[Лексическая] Отсутствует предварительное определение"),

ERROR\_ENTRY(106, "[Семантическая] Попытка использования операции присваивания между двумя идентификаторами"),

ERROR\_ENTRY(107, "[Семантическая] Обнаружены str идентификаторы в целочисленном выражении"),

ERROR\_ENTRY(108, "[Семантическая] Обнаружена неопознанная операция между строками"),

ERROR\_ENTRY(109, "[Семантическая] Обнаруженно невозможное деление на ноль числа или попытка деления строки на ноль"),

ERROR\_ENTRY(110, "[Системная] Ошибка при открытии файла с исходным кодом(-in)"),

ERROR\_ENTRY(111, "[Лексическая] Недопустимый символ в исходном файле (-in)"),

ERROR\_ENTRY(112, "[Системная] Ошибка при создании файла протокола(-log)"), ERROR\_ENTRY\_NODEF(113),

ERROR\_ENTRY(114, "[Семантическая] Несоответствие типов"),ERROR\_ENTRY\_NODEF(115),

ERROR\_ENTRY(116, "[Семантическая] Использование встроенной функции без подключения библиотеки"),

ERROR\_ENTRY(117, "[Лексическая] Превышен размер таблицы лексем"),

ERROR\_ENTRY(118, "[Семантическая] Обнаружена вторая главная функция"),

ERROR\_ENTRY(119, "[Семантическая] Запрещено присваивать значение функции"),

ERROR\_ENTRY(120 ,"[Семантическая] У функции Compare не подходящие тип данных у параметров"),

ERROR\_ENTRY(121, "[Семантическая] Ошибка с параметрами"),

ERROR\_ENTRY(122, "[Семантическая] Неверное количество параметров при вызове функции"),

ERROR\_ENTRY(123, "[Семантическая] Отсутствие главной функции"),

ERROR\_ENTRY(124, "[Лексическая] Не удалось разобрать слово"),

ERROR\_ENTRY\_NODEF(125),

ERROR\_ENTRY(126, "[Семантическая] Отсутствует возвращение из функции"),

ERROR\_ENTRY(127, "[Семантическая] Не верный тип параметра (STRLEN)"),

ERROR\_ENTRY\_NODEF10(130), ERROR\_ENTRY\_NODEF10(140), ERROR\_ENTRY\_NODEF10(150),

ERROR\_ENTRY\_NODEF10(160), ERROR\_ENTRY\_NODEF10(170), ERROR\_ENTRY\_NODEF10(180), ERROR\_ENTRY\_NODEF10(190),

ERROR\_ENTRY\_NODEF100(200), ERROR\_ENTRY\_NODEF100(300), ERROR\_ENTRY\_NODEF100(400), ERROR\_ENTRY\_NODEF100(500),

ERROR\_ENTRY(600, "[Синтаксическая] Неверная структура программы"),

ERROR\_ENTRY(601, "[Синтаксическая] Ошибка в операторах"),

ERROR\_ENTRY(602, "[Синтаксическая] Ошибка в выражении"),

ERROR\_ENTRY(603, "[Синтаксическая] Ошибка в параметрах функции"),

ERROR\_ENTRY(604, "[Синтаксическая] Ошибка в конструкции функции"),

ERROR\_ENTRY(605, "[Семантическая] В if не может использоваться переменная, которой не присвоено значение"),

ERROR\_ENTRY(606, "[Семантическая] Параметр не определен (использование в if невозможно!)"),

ERROR\_ENTRY(607, "[Семантическая] Параметр не определен (использование в while невозможно!)"),

ERROR\_ENTRY(608, "[Семантическая] Параметр не определен (использование в Compare невозможно!)"),

ERROR\_ENTRY(609, "[Семантическая] Ошибка в конструкции функции IF "),

ERROR\_ENTRY(610, "[Семантическая] Ошибка в типе данных IF "),

ERROR\_ENTRY(611, "[Семантическая] Параметр не определен "),

ERROR\_ENTRY(612, "[Семантическая] Переменная не может использовать имя ключевых слов! "),

ERROR\_ENTRY(613, "[Семантическая] Параметр должен быть условием! "),

ERROR\_ENTRY(614, "[Семантическая] Ошибка в типе параметров While! "),

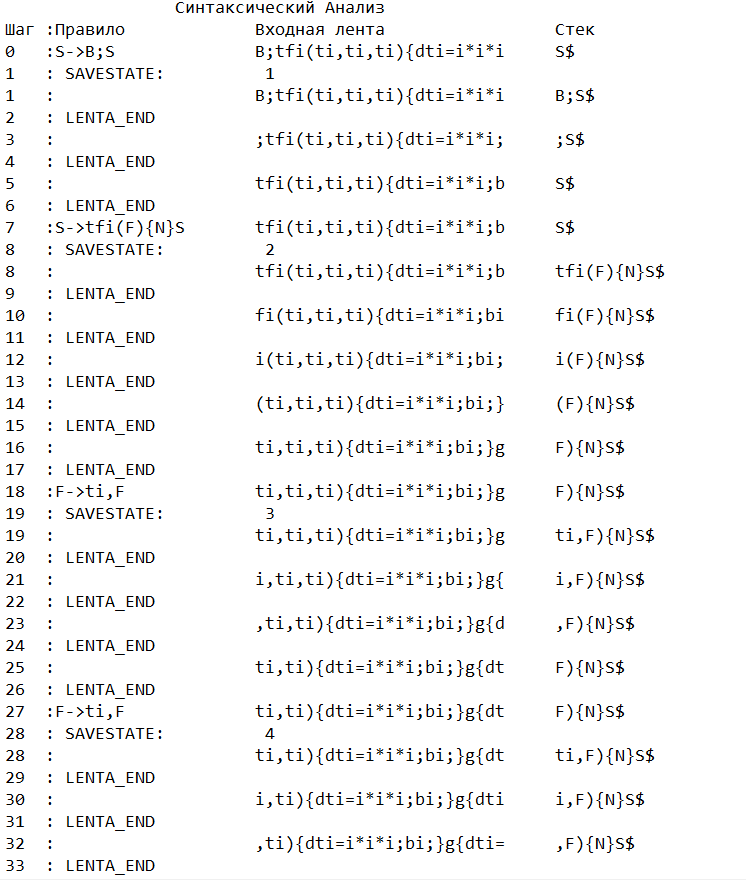
ERROR\_ENTRY(615, "[Семантическая] Параметр не определен (использование в Strlen невозможно!) "),

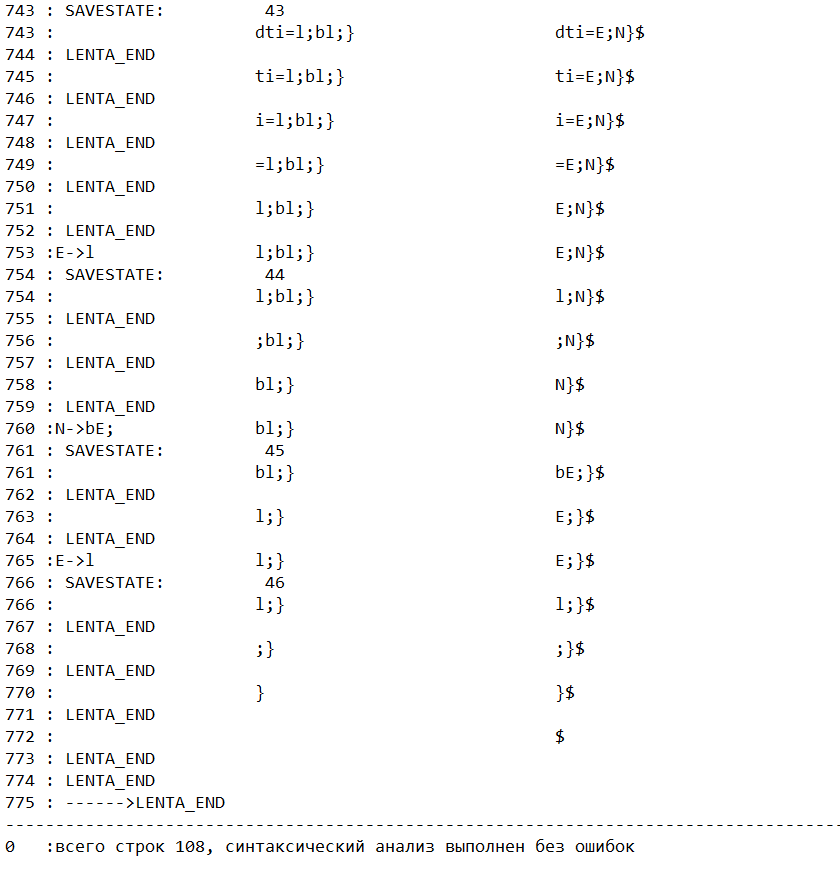
ERROR\_ENTRY(620),ERROR\_ENTRY\_NODEF10(630),ERROR\_ENTRY\_NODEF10(640),

ERROR\_ENTRY\_NODEF10(650),ERROR\_ENTRY\_NODEF10(660),ERROR\_ENTRY\_NODEF10(670),ERROR\_ENTRY\_NODEF10(680),ERROR\_ENTRY\_NODEF10(690),

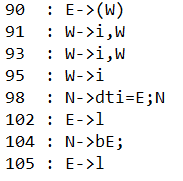
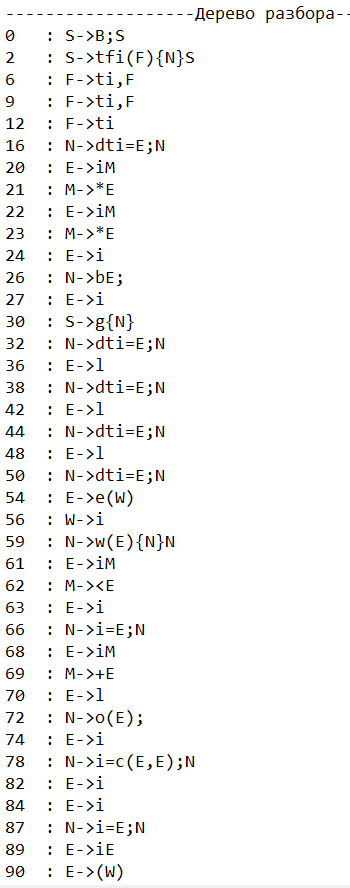
ERROR\_ENTRY\_NODEF100(700), ERROR\_ENTRY\_NODEF100(800),ERROR\_ENTRY\_NODEF100(900)

# Приложение И:

Результаты работы синтаксического анализатора. 



Дерево разбора



Приложение K: Структура и перечень сообщений семантического анализатора.

ERROR\_ENTRY(3, "[Семантическая] Несоответствие типов"),

ERROR\_ENTRY(4, "[Семантическая] Несоответствие присваиваемого типа типу переменной"),

ERROR\_ENTRY(6, "[Семантическая] Выход за предел допустимого значения переменной"),

ERROR\_ENTRY(7, "[Семантическая] Не хватает закрывающей кавычки"),

ERROR\_ENTRY(101, "[Семантическая] Отсутствие главной функции"),

ERROR\_ENTRY(106, "[Семантическая] Попытка использования операции присваивания между двумя идентификаторами"),

ERROR\_ENTRY(107, "[Семантическая] Обнаружены str идентификаторы в целочисленном выражении"),

ERROR\_ENTRY(108, "[Семантическая] Обнаружена неопознанная операция между строками"),

ERROR\_ENTRY(109, "[Семантическая] Обнаруженно невозможное деление на ноль числа или попытка деления строки на ноль"),

ERROR\_ENTRY(114, "[Семантическая] Несоответствие типов"),ERROR\_ENTRY\_NODEF(115),

ERROR\_ENTRY(116, "[Семантическая] Использование встроенной функции без подключения библиотеки"),

ERROR\_ENTRY(118, "[Семантическая] Обнаружена вторая главная функция"),

ERROR\_ENTRY(119, "[Семантическая] Запрещено присваивать значение функции"),

ERROR\_ENTRY(120 ,"[Семантическая] У функции Compare не подходящие тип данных у параметров"),

ERROR\_ENTRY(121, "[Семантическая] Ошибка с параметрами"),

ERROR\_ENTRY(122, "[Семантическая] Неверное количество параметров при вызове функции"),

ERROR\_ENTRY(123, "[Семантическая] Отсутствие главной функции"),

ERROR\_ENTRY(126, "[Семантическая] Отсутствует возвращение из функции"),

ERROR\_ENTRY(127, "[Семантическая] Не верный тип параметра (STRLEN)"),

ERROR\_ENTRY(605, "[Семантическая] В if не может использоваться переменная, которой не присвоено значение"),

ERROR\_ENTRY(606, "[Семантическая] Параметр не определен (использование в if невозможно!)"),

ERROR\_ENTRY(607, "[Семантическая] Параметр не определен (использование в while невозможно!)"),

ERROR\_ENTRY(608, "[Семантическая] Параметр не определен (использование в Compare невозможно!)"),

ERROR\_ENTRY(609, "[Семантическая] Ошибка в конструкции функции IF "),

ERROR\_ENTRY(610, "[Семантическая] Ошибка в типе данных IF "),

ERROR\_ENTRY(611, "[Семантическая] Параметр не определен "),

ERROR\_ENTRY(612, "[Семантическая] Переменная не может использовать имя ключевых слов! "),

ERROR\_ENTRY(613, "[Семантическая] Параметр должен быть условием! "),

ERROR\_ENTRY(614, "[Семантическая] Ошибка в типе параметров While! "),

ERROR\_ENTRY(615, "[Семантическая] Параметр не определен (использование в Strlen невозможно!) "),