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 NEA-2018»

Выполнил студент Немкевич Евгений Анатольевич

(Ф.И.О.)

Руководитель проекта преп.ст. Рауба Алексей Александрович

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

Заведующий кафедрой к.т.н., доц. Пацей Наталья Владимировна

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

Консультанты преп.ст. Рауба Алексей Александрович

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

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

Нормоконтролер преп.ст. Рауба Алексей Александрович

ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2018

**Содержание**

[Введение 4](#_Toc501144458)

[Глава 1. Спецификация языка программирования 5](#_Toc501144459)

[1.1 Характеристика языка программирования 5](#_Toc501144460)

[1.2 Алфавит языка 5](#_Toc501144461)

[1.3 Применяемые сепараторы 5](#_Toc501144462)

[1.4 Применяемые кодировки 6](#_Toc501144463)

[1.5 Типы данных 6](#_Toc501144464)

[1.6 Преобразование типов данных 6](#_Toc501144465)

[1.7 Идентификаторы 6](#_Toc501144466)

[1.8 Литералы 6](#_Toc501144467)

[1.9 Объявление данных и область видимости 7](#_Toc501144468)

[1.10 Инициализация данных 7](#_Toc501144469)

[1.11 Инструкции языка 7](#_Toc501144470)

[1.12 Операции языка 8](#_Toc501144471)

[1.13 Выражения и их вычисления 8](#_Toc501144472)

[1.14 Программные конструкции языка 8](#_Toc501144473)

[1.15 Область видимости идентификаторов 9](#_Toc501144474)

[1.16 Семантические проверки 9](#_Toc501144475)

[1.17 Распределение оперативной памяти на этапе выполнения 10](#_Toc501144476)

[1.18 Стандартная библиотека и ее состав 10](#_Toc501144477)

[1.19 Вывод данных 10](#_Toc501144478)

[1.20 Точка входа 10](#_Toc501144479)

[1.21 Препроцессор 10](#_Toc501144480)

[1.22 Соглашения о вызовах 11](#_Toc501144481)

[1.23 Объектный код 11](#_Toc501144482)

[1.24 Классификация сообщений транслятора 11](#_Toc501144483)

[1.25 Контрольный пример 11](#_Toc501144484)

[Глава 2. Структура транслятора 12](#_Toc501144485)

[2.1 Компоненты транслятора их назначение и принципы взаимодействия 12](#_Toc501144486)

[2.2 Перечень входных параметров транслятора 13](#_Toc501144487)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 13](#_Toc501144488)

[Глава 3. Разработка лексического анализатора 14](#_Toc501144489)

[3.1 Структура лексического анализатора 14](#_Toc501144490)

[3.2 Контроль входных символов 14](#_Toc501144491)

[3.3 Удаление избыточных символов 15](#_Toc501144492)

[3.4 Перечень ключевых слов, сепараторов, символов операций соответствующим им лексемам и конечных автоматов 15](#_Toc501144493)

[3.5 Основные структуры данных 15](#_Toc501144494)

[3.6 Принцип обработки ошибок 16](#_Toc501144495)

[3.7 Структура и перечень сообщений лексического анализатора 16](#_Toc501144496)

[3.8 Параметры лексического анализатора и режимы его работы 17](#_Toc501144497)

[3.9 Алгоритм лексического анализа 17](#_Toc501144498)

[3.10 Контрольный пример 17](#_Toc501144499)

[Глава 4. Разработка синтаксического анализатора 18](#_Toc501144500)

[4.1 Структура Синтаксического анализатора 18](#_Toc501144501)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 18](#_Toc501144502)

[4.3 Построение конечного магазинного автомата 19](#_Toc501144503)

[4.4 Основные структуры данных 20](#_Toc501144504)

[4.5 Описание алгоритма синтаксического разбор 20](#_Toc501144505)

[4.6 Структура и перечень сообщений синтаксического анализатора 20](#_Toc501144506)

[4.7 Параметры синтаксического анализатора и режимы его работы 20](#_Toc501144507)

[4.8 Принцип обработки ошибок 20](#_Toc501144508)

[4.9 Контрольный пример 21](#_Toc501144509)

[Глава 5. Разработка семантического анализатора 22](#_Toc501144510)

[5.1 Структура семантического анализатора 22](#_Toc501144511)

[5.2 Функции семантического анализа 22](#_Toc501144512)

[5.3 Принцип обработки ошибок 23](#_Toc501144513)

[5.4 Контрольный пример 23](#_Toc501144514)

[Глава 6. Вычисление выражений 25](#_Toc501144515)

[6.1 Выражения, допускаемые языком 25](#_Toc501144516)

[6.2 Польская запись и принцип ее построения 25](#_Toc501144517)

[6.3 Программная реализация обработки выражений 25](#_Toc501144518)

[6.4 Контрольный пример 26](#_Toc501144519)

[Глава 7. Генерация кода 27](#_Toc501144520)

[Глава 8. Тестирование транслятора 29](#_Toc501144521)

[8.1 Тестирование контрольного примера 29](#_Toc501144522)

[8.2 Тестирование программ с ошибками 29](#_Toc501144523)

[Заключение 31](#_Toc501144524)

[Приложение А 32](#_Toc501144525)

[Приложение Б 33](#_Toc501144526)

[Приложение В 36](#_Toc501144527)

[Приложение Г 40](#_Toc501144528)

[Приложение Д 42](#_Toc501144529)

[Приложение Е 49](#_Toc501144530)

[Литература 50](#_Toc501144531)

## **Введение**

Задачей данного курсового проекта является разработка компилятора для языка программирования NEA-2018. Где компилятор – это частный случай (разновидность) трансляторов. А транслятор – это программа, принимающая на вход программу на одном языке т.е. наш язык NEA-2018 и преобразующая её в программу на другом языке, в нашем случае транслируя в JavaScript в следствии чего потом будет интерпретироваться браузером.

Интерпретация – построчный анализ, обработка и выполнение исходного кода программы или запроса.

Для этого понадобилось:

* Разработать спецификацию нашего языка программирования.
* Определиться со структурой транслятора.
* Разработать лексический, синтаксический и семантический анализаторы.
* Сделать обработку выражений.
* Произвести генерацию кода, транслировать язык программирования NEA-2018 в JavaScript.
* Произвести тестирование.

Язык программирования NEA-2018 предназначен для выполнения простейших арифметических действий и операций над строками.

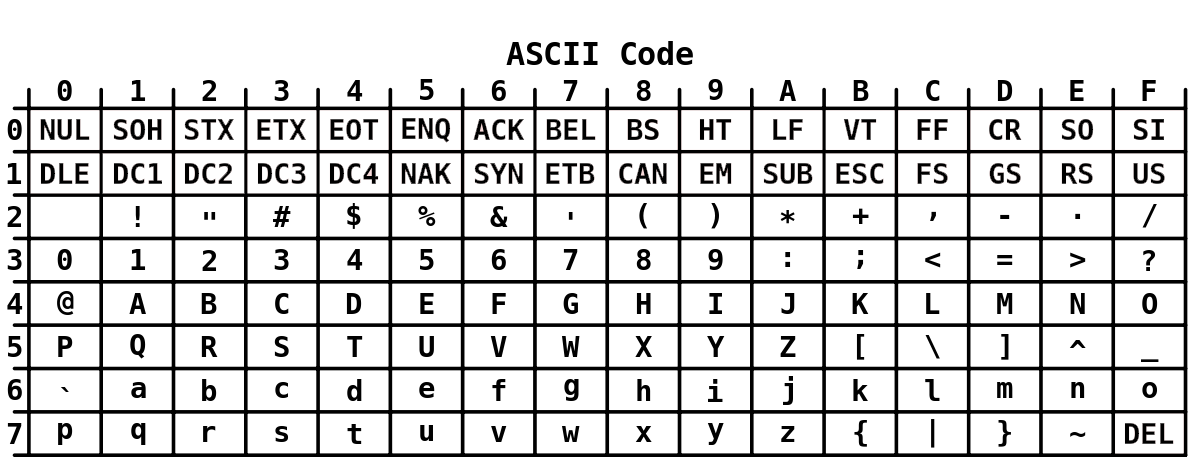
## **1 Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

Язык программирования NEA-2018 является декларативным, универсальным, процедурным, строготипизированным, не объектно-ориентированным, транслируемый.

## **Алфавит языка**

Символы, разрешенные к использованию при написании кода: латинские символы размером один байт, кодировки ASCII. Данная таблица приведена на рисунке 1.1.

Рисунок 1.1 – Основная таблица ASCII

Русский язык запрещён. В литералах, разрешено использовать десятичные цифры и алфавит латиницы.

## **Применяемые сепараторы**

Применяемые сепараторы в языке NEA-2018, приведены в таблице 1.1.

Таблица 1.1 – Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Область применения |
| « » (пробел) | Допускается везде, кроме идентификаторов и ключевых слов |
| ; (точка с запятой) | Разделение конструкций |
| { } | Заключение программного блока |
| = | Присвоение значения |
| ( ) | Приоритет операций, параметры функции |
| , (запятая) | Разделение параметров |
| +,-,\*,/ | Арифметические операции |

## **Применяемые кодировки**

Применяемая кодировка для написания программ на языке NEA-2018, - стандартная кодировка ASCII (верхняя часть), представленная на рисунке 1.1- Основная таблица ASCII.

## **Типы данных**

В язык NEA-2018 предусмотрены два типа данных: целочисленный и строковый, представленные в таблице 1.2.

Таблица 1.2 – Типы данных

| Тип | Описание |
| --- | --- |
| Integer | Целочисленный тип данных. В памяти занимает 4 байта. Автоматическая инициализация 0 (нулем). Максимальное значение 215-1, минимальное 0. |
| String | Строковый тип данных. Один символ занимает 1 байт. Максимальная длинна строки (число символов) составляет 253 символа. При объявлении автоматически инициализируется длинной 0, и концом строки “\0”. |
| Bool | Логический тип данных. В памяти занимает 1 байт. Автоматическая инициализация false. Максимальное значение True |

## **Преобразование типов данных**

Преобразование не поддерживается. Так как язык является строго типизированным.

## **Идентификаторы**

В идентификаторах языка NEA-2018 разрешается использовать только латинские буквы нижнего регистра. Типы идентификаторов: имя переменной или функции, литерал, параметр. Идентификаторы не могут совпадать с ключевыми символами. Данные правила справедливы как для переменных, так и для функций.

## **Литералы**

В языке существует 2 вида литералов: литералы целого типа и строковые, описаны в таблице 1.3. Они осуществляют инициализацию переменных.

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| Тип | Описание |
| Литералы целого типа | Интерпретируются как integer, являются rvalue[2]. Задается в десятичной форме. |
| Строковые литералы | Интерпретируются как string, заключаются в одинарные кавычки (‘Hello’), являются rvalue[2] |
| Логические литералы | Интерпретируются как bool, являются rvalue[2]. |

## **Объявление данных и область видимости**

Все переменные имеют область видимости, а именно префикс – название функции, в которой они находятся, что разрешает использование в различных функциях переменных с одинаковым именем. Параметры функции видны только внутри функции. Переменные, объявленные в одной функции, недоступны в другой.

## **Инициализация данных**

Инициализация на языке NEA-2018 будет происходить при объявлении данных начинающихся с ключевого слова enroll, указывается тип данных и имя идентификатора, где потом инициализируется идентификатор, смотрите таблицу 1.4.

Таблица 1.4 – Инициализация

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Инициализация переменной | enroll <тип данных> <идентификатор>;  <идентификатор> = значение; |

## **Инструкции языка**

Инструкция для языка NEA-2018 представлена в таблице 1.5.

Таблица 1.5 – Инструкция языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление переменной | enroll <тип данных> <идентификатор>. |
| Объявление функции | <тип данных> function <идентификатор> (<тип данных> <идентификатор>, …)  {  / программный блок /  return <идентификатор/литерал>.  } |
| Вызов функции | <идентификатор> (<идентификатор>, ...) |
| Присвоение значения | = |
| Печать данных | output <литерал/идентификатор>. |
| Функции стандартной библиотеки  (применяются при инициализации и в выражениях) | koren(integer) – определяет квадратный корень из int, применимо только для идентификаторов типа данных integer.  stepen(integer, integer) – возводит первый операнд в степень, равную значению второго операнда. |

Продолжение таблицы 1.5

|  |  |
| --- | --- |
| Возвращаемое значение | return <литерал/идентификатор>. |

## **Операции языка**

Арифметические операции, которые можно использовать представлены в таблице 1.6.

Таблица 1.6 – Операции языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Арифметические операции языка | + – бинарное сложение  - – бинарный минус  \* – бинарное произведение  / – бинарное деление  ( ) – приоритет операций |

Приоритет подчиняется правилам:

* Если в вы­ра­же­ние без ско­бок вхо­дят толь­ко сло­же­ние и вы­чи­та­ние или толь­ко умно­же­ние и де­ле­ние, то дей­ствия вы­пол­ня­ют в том по­ряд­ке, в каком они на­пи­са­ны;
* Если в вы­ра­же­ние без ско­бок вхо­дят не толь­ко дей­ствия сло­же­ния и вы­чи­та­ния, но и умно­же­ния и де­ле­ния, или оба этих дей­ствия, то сна­ча­ла вы­пол­ня­ют по по­ряд­ку (слева на­пра­во) умно­же­ние и де­ле­ние, а затем сло­же­ние и вы­чи­та­ние.
* Если в вы­ра­же­нии име­ют­ся скоб­ки, то сна­ча­ла вы­чис­ля­ют зна­че­ние вы­ра­же­ний в скоб­ках.

## **1.13 Выражения и их вычисления**

Правила для выражения и их вычисления:

* Выражение записывается только в одну строку.
* Допускается использование скобок (приоритет).
* Не допускается использование двух и более операторов.
* Выражение допускает содержать вызов функции

## **1.14** **Программные конструкции языка**

Основные программные конструкции языка NEA-2018 представлены в таблице 1.7.

Таблица 1.7 – Основные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | main ()  {  / программный блок / |

Продолжение таблицы 1.7

|  |  |
| --- | --- |
|  | } |
| Функции | <тип данных> function <идентификатор>  (<тип данных> <идентификатор>, …)  {  / программный блок /  return <идентификатор/литерал>.  }  <тип данных> function <идентификатор> ()  {  return <идентификатор/литерал>.  } |

## **1.15 Область видимости идентификаторов**

Область видимостиидентификаторов, можно обратиться только ниже описания данной сущности и введения этого идентификатора. «Ниже» понимается буквально – ниже по тексту программы. Т.е. до описания заголовка функции ее нельзя вызывать.

Глобальные переменные – это все переменные, которые объявлены в секциях объявления переменных integer, не привязанных к функциям. Их область видимости – все пространство программы ниже по тексту по отношению к месту объявления.

Локальные переменные, т.е. параметры функции или процедуры, переданные по значению, а также переменные созданные и вложенные в функцию, видны и доступны для работы с ними только внутри тела данной функции.

При совпадении имен глобальных и локальных переменных конфликта не возникает, но становятся доступны только локальные переменные, т.к. они своими идентификаторами как бы «закрывают» возможность обратиться к глобальным переменным.

## **1.16** **Семантические проверки**

Основные семантические правила проверяемые на этапах работы транслятора, представлены в таблице 1.8.

Таблица 1.8 – Семантические правила

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Проверка на наличие точки входа в программу main() |
| 2 | Усечение идентификаторов до 15 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 |

Продолжение таблицы 1.8

|  |  |
| --- | --- |
| 6 | Переменные, используемые в функции, должны быть предварительно объявлены, после чего инициализированы |
| 7 | Проверка на максимальное и минимальное значение целочисленного литерала |
| 8 | Проверка на максимальную длину строкового литерала |

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

На этапе выполнения все переменные помещаются в стек.

## **1.18** **Стандартная библиотека и ее состав**

Подключение стандартной библиотеки: CMatLib; Обязательно внутри главной функции main { }. Таблица с описанием функций стандартной библиотеки языка NEA-2018 представлена в таблице 1.9.

Таблица 1.9 – Библиотека

|  |  |
| --- | --- |
| Функции стандартной библиотеки CTD (применяются при инициализации и в выражениях) | length(integer) – определяет длину для типа integer.  concat(string, string) – объединение двух строк. |

## **1.19** **Вывод данных**

Для вывода данных используется функция output (<имя идентификатора>). Например: output(a);

**1.20**  **Точка входа**

Функция точки входа должна быть объявлена в соответствии с соглашением о вызовах стандартного запроса, смотрите таблицу 1.10.

Таблица 1.10 – Точки входа

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | main ()  {  / программный блок /  } |

## **1.21** **Препроцессор**

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в NEA-2018 отсутствует.

## **1.22** **Соглашения о вызовах**

Соглашение о вызовах – это протокол для передачи аргументов функциям. Другими словами, это договоренность между вызывающим и вызываемым кодом. И в данном языке NEA-2018 по умолчанию применяется соглашение \_stdcall, где параметры помещаются в стек, порядок параметров передается справа налево, где стек освобождает вызываемый код, возврат через регистр EAX.

## **1.23** **Объектный код**

Исходный код, написанный на языке NEA-2018, транслируется в язык JavaScript, после чего интерпретируется браузером.

## **1.24** **Классификация сообщений транслятора**

В ходе своей работы, транслятор генерирует сообщения, которые информируют пользователя о допущенных ошибках. Все сообщения транслятора разделены на интервалы, в зависимости от того на каком этапе была обнаружена ошибка. Все интервалы представлены в таблице 1.12.

Таблица 1.12 – Сообщения транслятора

|  |  |
| --- | --- |
| Интервал | Описание |
| 0-99 | Ошибки системы |
| 100-119 | Ошибки входного кода |
| 120-200 | Ошибки на этапе лексического анализа |
| 400-500 | Ошибки на этапе семантического анализа |
| 600-700 | Ошибки на этапе синтаксического анализа |

## **1.25** **Контрольный пример**

Контрольный пример языка программирования NEA-2018 представлен в приложении А.

## **2 Структура транслятора**

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

Транслятор языка NEA-2018 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. Графически схема трнаслятора представлена на рисунке 2.1.

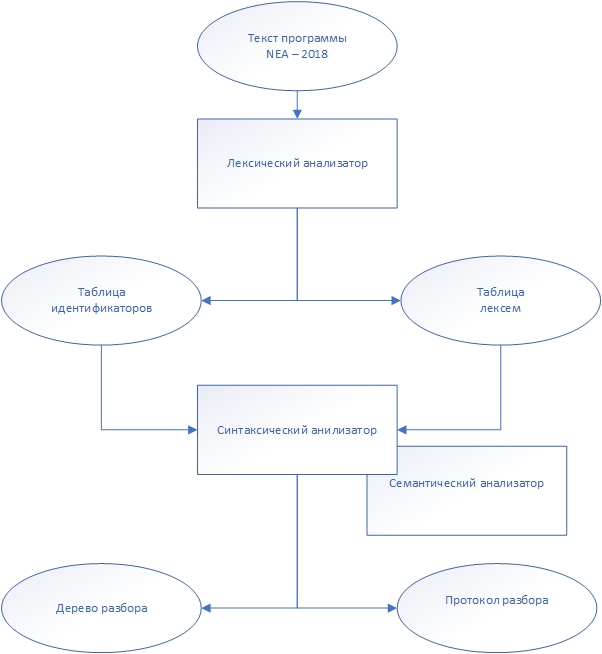


Рисунок 2.1 – Схема структуры транслятора

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. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Синтаксический анализатор | С помощью синтаксического анализатора проверяется правильность написанных конструкций по заданной грамматике. |
| Семантический анализатор | Как отдельная программа в приложении не реализована, вся проверка по правилам осуществляется на этапах лексического и синтаксического анализа |
| Транслятор кода | Связан с работой лексического анализатора. Посредством полученных данных: таблицы лексем и идентификаторов, код на языке NEA-2018 транслируется в код на языке JavaScript. |

## **2.2** **Перечень входных параметров транслятора**

Благодаря входным параметрам транслятора осуществляется контроль за выводом протоколов работы разных компонент, а также указывается файл с исходным кодом программы. Перечень входных параметров представлена в таблице 2.2.

Таблица 2.2 – Входные параметры транслятора

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Тип параметра |
| -in:<имя.txt> | Входной файл | Обязательный |
| -log:<имя.txt> | Вывод основной информации и ошибок | Не обязательный |

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

Представлено в таблице 2.3.

Таблица 2.3 – Протокол транслятора.

|  |  |
| --- | --- |
| Тип информации | Описание  информации |
| Дата и время трансляции | Выводит дату и время. |
| Параметры командой строки | Выводит информацию об указанных параметрах командной строки. |
| Полная таблица лексем | Выводит таблицу лексем с информацией к каждой лексеме. |
| Таблица идентификаторов | Выводит таблицу идентификаторов с дополнительной информацией. |
| Трассировочная информация синтаксического анализа | Выводит полную информацию о разборе таблицы лексем на синтаксическом анализаторе. |
| Правила разбора | Выводит правила, по которым осуществился разбор исходного кода. |

## **3 Разработка лексического анализатора**

## **3.1** **Структура лексического анализатора**

Для лексического анализатора входными данными является массив цепочек, предварительно сформированный на первичной обработке исходного кода программы. Результатом работы лексического анализатора является протокол работы, перечень всех найденных в исходном коде программы лексем. Кроме того, информация о некоторых лексемах, найденных в исходной программе, должна помещаться в таблицу идентификаторов. Структура лексического анализатора NEA-2018 представлена на рисунке 3.1.

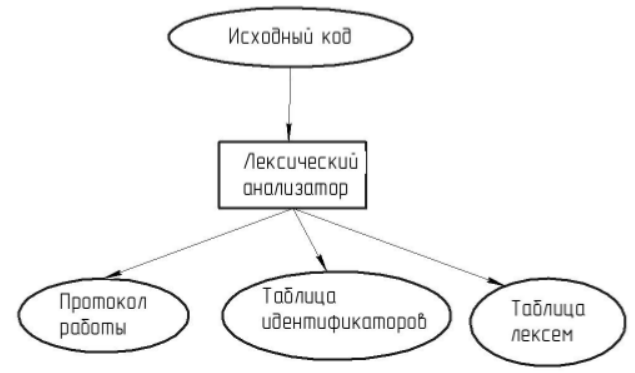


Рисунок 2.1 – Структура лексического анализатора NEA-2018

**3.2**  **Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2.

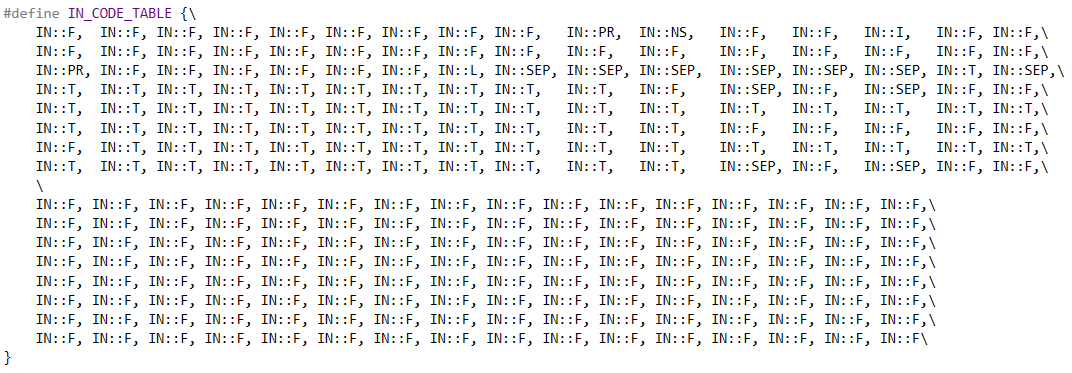


Рисунок 3.2 – Структура лексического анализатора NEA-2018

Принцип работы таблицы: каждому элементу соответствует значение в шестнадцатеричной системе счисления - такое же, как и в ASCII (см. рисунок 1.1).

В представленной таблице: F - запрещённый символ, T - разрешённый символ, I - игнорируемый символ, SEP – символы-сепараторы, PR - пробел, NS - новая строка, L – кавычка.

## **3.3** **Удаление избыточных символов**

Описание алгоритма удаления избыточных символов, а также разбиения исходного кода на цепочки:

1. Посимвольно считываем файл с исходным кодом программы.
2. Если встречаем кавычку (‘’), то срабатывает флаг, запрещающий удаление каких-либо символов внутри кавычек (т.к. внутри кавычек содержатся данные пользователя – литералы).
3. Встреча пробела включает проверку предыдущего (буферного) символа.
   1. Если предыдущий символ является пробелом, то текущий символ игнорируется.
   2. Если предыдущий символ является допустимым символом, то текущий пробел – разделитель единиц языка, следовательно, сохраняем считанную единицу в формируемую структуру первично обработанного исходного кода.
4. В конце каждой итерации посимвольного считывания текущий символ сохраняется в буфер.

## **3.****4 Перечень ключевых слов, сепараторов, символов операций соответствующим им лексемам и конечных автоматов**

Благодаря замене цепочек, написанных на языке NEA-2018, лексемами -упрощается дальнейшая обработка исходного кода программы. Перечень цепочек, соответствующих им автоматов и лексем, представлена в приложении Б.

**3.5 Основные структуры данных**

Описание основных структур данных, используемых для хранения таблиц лексем, представлено на рис. 3.3.

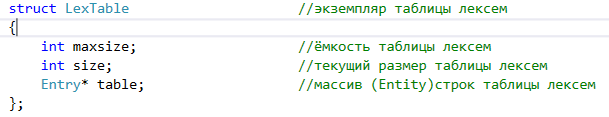
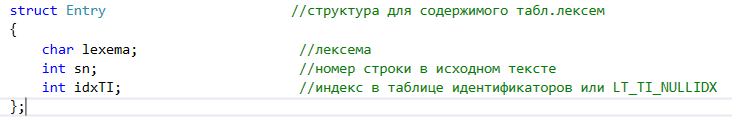
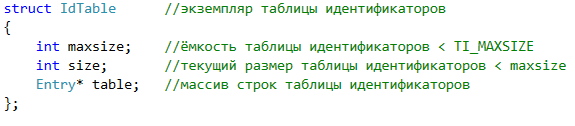


Рисунок 3.3 – Структуры таблиц лексем NEA-2018

Описание основных структур данных, используемых для хранения таблиц идентификаторов, представлено на рис. 3.4.



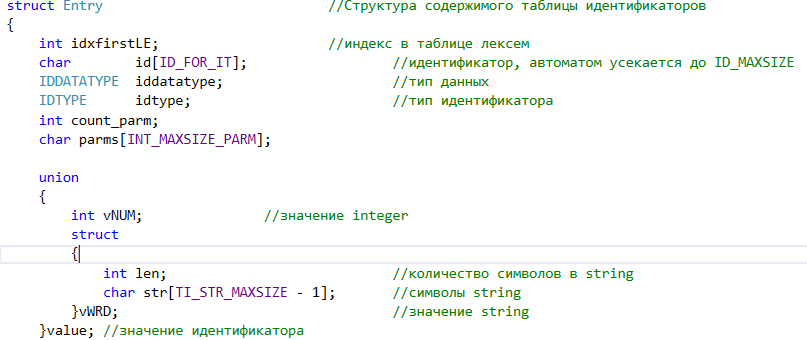


Рисунок 3.4. – Структуры таблиц идентификаторов NEA-2018

## **3.6** **Принцип обработки ошибок**

В трансляторе NEA-2018 предусмотрена генерация ошибок разной степени: критических и предупреждений. При обнаружении предупреждения, работа транслятора не останавливается, предупреждения записываются в структуру, которая наполняется по ходу работы транслятора, но при обнаружении критической ошибки, работа транслятора останавливается.

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

Перечень сообщений, формируемых лексическим анализатором в ходе своей работы, представлен в таблице 3.2.

Таблица 3.2 – Перечень сообщений лексического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 103 | Превышено максимальное количество параметров при вызове функции |
| 105 | Превышен максимальный размер литерала |
| 107 | Попытка переопределения |
| 118 | Отсутствует предварительное определение |
| 115 | Запрещено присваивать значение функции |
| 117 | Превышен размер таблицы лексем |
| 119 | Отсутствие сепаратора для разделения конструкций |

## **3.8** **Параметры лексического анализатора и режимы его работы**

Транслятор допускает использование параметров для управления работой лексического анализатора, а именно: выводом таблицы лексем и таблицы идентификаторов. Описание параметров представлено в таблице 2.2.

## **3.9** **Алгоритм лексического анализа**

Лексический анализ – является первой и наиболее простой фазой трансляции. Алгоритм лексического анализатора заключается в распознавании и разборе цепочек исходного кода. Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно представить в виде графа. Пример графа представлен на рисунке 3.5. В виде кода представлен на рисунке 3.6. На рисунке осуществляется разбор цепочки “main”, где S0 – начальное состояние, а S4 – конечное.



Рисунок 3.5 – Граф переходов для цепочки “main”

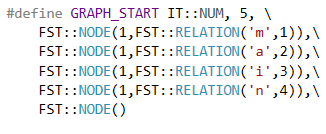


Рисунок 3.6 – Граф переходов для цепочки “main”

## **3.10 Контрольный пример**

Результатом работы лексического анализатора являются таблица лексем и таблица идентификаторов. Содержимое таблиц на основе исходного кода из приложения А представлено в приложении В.

## **4 Разработка синтаксического анализатора**

## **4.1** **Структура Синтаксического анализатора**

Синтаксический анализатор – является следующей ступенью транслятора. Синтаксический анализатор на свой вход принимает таблицу лексем, которую в ходе свой работы он перебирает с целью обнаружений в ней неверных синтаксических конструкций. Свою работу начинает только при условии отсутствии ошибок в их общей структуре. Если в ходе его работы не было обнаружено ошибок, то формируется дерево разбора. Структура синтаксического анализатора представлена на рисунке 4.1.

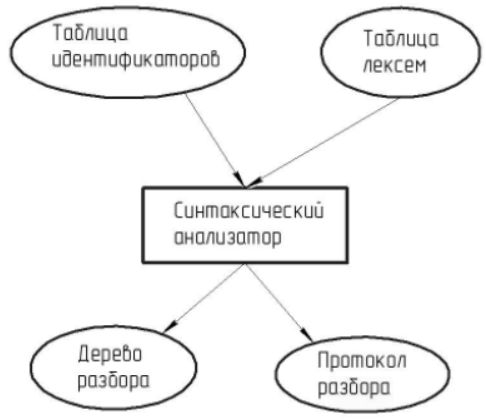


Рисунок 4.1 – Структура синтаксического анализатора NEA-2018

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

Каждый язык программирования описывается с помощью набора правил, определяющих структуру правильной программы. Наиболее удобным формализмом для описания синтаксических конструкций языка программирования являются контекстно-свободные грамматики.

Грамматика для синтаксического разбора языка NEA-2018 представляется четверкой G = <T, N, P, S>, где Т – множество терминальных символов, N – множество нетерминальных символов, P – множество правил языка, S – начальный символ грамматики, являющийся нетерминалом.

В грамматике языка NEA-2018 множество нетерминальных символов представлено следующим образом:

1. S – Порождает правила, описывающие общую структуру программы;
2. N – Порождает правила, описывающие основные конструкции языка;
3. Е – Порождает правила, описывающие выражения;
4. F – Порождает правила, описывающие формальные параметры функции;
5. M – Порождает правила, описывающие арифметические действия;
6. W – Порождает правила, описывающие фактические параметры функции;

Перечень правил, описывавших контекстно-свободную грамматику языка NEA-2018, представлен в таблице 4.1.

Таблица 4.1 – Правила грамматики NEA-2018

|  |  |
| --- | --- |
| Нетерминальный символ | Цепочки правил |
| S→ | qfi(F){N}S | qfi(){N};S | nfi(F){N}S | nfi(){N};S | s{N} |
| N→ | cni;N | cqi;N | i=E;N | z;N | oE;N | rE; | cni; | cqi; | i=E; | z; | oE; |
| E→ | i | l | i,E |(E)| lM | iM | (W) | (W)M | (E)M | i() | i()M | k(W) | k(W)M | b(W)M | b(W) |
| F→ | qi,F | ni,F | qi | ni |
| W→ | i | l | i,W | l,W |
| M→ | +E | -E | \*E | /E |

## **4.3** **Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку,где Q – множество состояний автомата, V – алфавит входных символов, Z – алфавит специальных магазинных символов,  - функция переходов автомата,  - начальное состояние автомата,  - начальное состояние магазинного автомата, F – множество конечных состояний.

Пример работы магазинного автомата для цепочки представлен на рисунке 4.2.

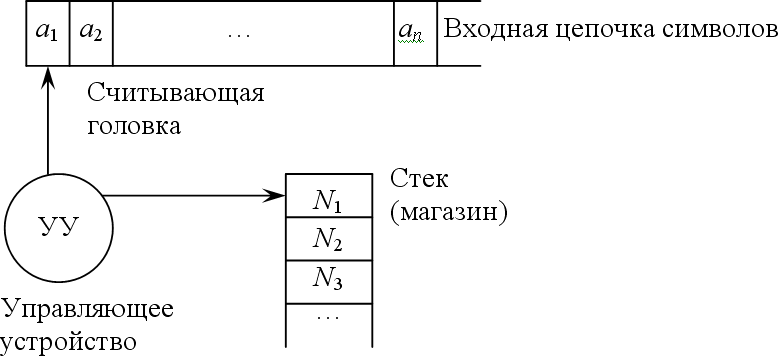


Рисунок 4.2 – Автомат с магазинной памятью

## **4.4** **Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающая правила языка NEA-2018. Структура синтаксического анализатора представлена в приложении Г.

## **4.5** **Описание алгоритма синтаксического разбор**

Описание алгоритм синтаксического разбора:

* Происходит поиск и выделение синтаксических конструкций в исходном тексте (разбор).
* Распознавание (проверка правильности) синтаксических конструкций.
* Выявление ошибок и продолжение процесса распознавания после обработки ошибок.
* В случае отсутствия ошибок, формируется дерево разбора.

## **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 |
| 605 | Ошибка в построении выражения |

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

Транслятор допускает использование параметров для управления работой синтаксического анализатора, а именно: вывода дерева разбора и трассировки. Описание параметров представлено в таблице 2.2.

## **4.8** **Принцип обработки ошибок**

Принцип заключается в том, что синтаксический анализатор перебирает всевозможные правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем. В случае если не была найдена ни одна подходящая цепочка, то формируется соответствующая ошибка из таблицы 4.2. Все ошибки записываются в общую структуру ошибок, а также отображаются на консоли.

## **4.9** **Контрольный пример**

Из контрольного примера представленного в приложении А результатом работы синтаксического анализатора, является трассировка и дерево разбора, представленные в приложении Д. Также представлено графическое представление дерева в графической работе №1.

**5 Разработка семантического анализатора**

## **5.1** **Структура семантического анализатора**

Семантический анализатор принимает на свой вход таблицы лексем и идентификаторов. Проверка на ошибки в исходном коде производится одновременно с работой синтаксического и лексического анализаторов, из-за чего исключается отдельная часть семантического анализа. Семантические правилаязыка NEA-2018 представлены в таблице 5.1.

Таблица 5.1 – Семантические правилаязыка NEA-2018

|  |  |
| --- | --- |
| № | Правило |
| 1 | Наличие функции main |
| 2 | Усечение слишком длинных идентификаторов до 7 символов ([a-z]) |
| 3 | Сначала осуществляется проверка на ключевые слова, а затем на идентификатор. Не допускаются идентификаторы, совпадающие с ключевыми словами |
| 4 | Нет повторяющихся наименований функций |
| 5 | Нет повторяющихся объявлений идентификаторов |
| 6 | Предварительное объявление и инициализация, применяемых функций |
| 7 | Предварительное объявление и инициализация, применяемых идентификаторов. |
| 8 | Соответствие типов формальных и фактических параметров при вызове функций |
| 9 | Если ошибка возникает на этапе лексического анализа, синтаксический анализ не выполняется |
| 10 | При возникновении ошибки в процессе лексического анализа, ошибочная фраза игнорируется (предполагается, что ее нет) и осуществляется попытка разбора следующей фразы. Граница фразы, любой сепаратор (пробел, скобка, запятая, точка с запятой и пр.) |
| 11 | При возникновении ошибки в процессе синтаксического анализа, ошибочная фраза игнорируется (предполагается, что ее нет) и осуществляется попытка разбора следующей фразы. Граница фразы – точка с запятой (;). |

## **5.2** **Функции семантического анализа**

Семантический анализатор предназначен для проверки соответствия исходного кода спецификации, например, соответствие исходного кода заданной грамматике (синтаксический анализ), длина идентификаторов.

В данной программе в отдельный блок семантического анализа вынесены те проверки, которые невозможно выполнить на других этапах, например, связанные с типами данных, а также те проверки, которые проще обработать в отдельном блоке, чем в связке с другими этапами.

Принцип обработки ошибок семантическим анализатором заключается в следующем: все обнаруженные ошибки записываются в общую структуру и продолжается дальнейшая работа транслятора. Сообщения, формируемые семантическим анализатором представлены в таблице 5.2.

Таблица 5.2 – Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 400 | Точка входа main () не найдена |
| 401 | Попытка преобразования типов |
| 402 | Использование стандартных функций без подключения стандартной библиотеки |
| 403 | Подключено более одной стандартной библиотеки |
| 404 | Попытка проведения операций со строковым типом данных |
| 405 | Тип возвращаемого значения и тип функции не соответствуют |
| 406 | Неправильное количество параметров при вызове функции |
| 407 | Неправильный тип параметров при вызове функции |
| 408 | Несоответствие типов |
| 409 | Неправильный вызов функции |

## **5.3** **Принцип обработки ошибок**

В случае возникновения ошибок, вызываем функцию получения ошибки, которая принимает обязательным параметром код ошибки в таблице сообщений, см. таблицу 5.1. Затем производится вывод ошибки в поток Log-файла, в случае отсутствия данного потока, ошибка выводится в консоль. При возникновении ошибки работа транслятора не прекращается, что даёт возможность анализировать одновременно несколько возникших проблем.

## **5.4** **Контрольный пример**

Контрольный пример заключается в тестировании функций семантического анализатора при наличии соответствующих ошибок в исходном коде. Тестирование представлено в таблице 5.3.

Таблица 5.3 – Тестирование функций

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| integer function inc(integer a)  {  enroll string b;  return b;  }  main  {  enroll integer a;  a = 1;  a = inc(a);  } |  |
| integer function inc(integer a, string b)  {  return a;  }  main  {  enroll integer a;  enroll integer b;  a = 1;  a = inc(a,b);  } |  |
| integer function inc(integer a, string b)  {  return a;  }  main  {  enroll integer a;  integer b;  a = 1;  a = inc(a);  } |  |
| main  {  integer a;  string b;  a = 1;  b = a;  } |  |

**6 Вычисление выражений**

## **6.1** **Выражения, допускаемые языком**

Выражения, допускаемые языком NEA-2018, выполняются над целочисленными типами данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, / и (), и вызовы функций из них.

Приоритетность арифметических операций представлена в таблице 6.1.

Таблица 6.1 – Приоритетность операций

|  |  |  |
| --- | --- | --- |
| Операция | Приоритет | Порядок выполнения |
| ( | 1 | Наивысший приоритет (выполняется в первую очередь) |
| ) | 1 | Наивысший приоритет (выполняется в первую очередь) |
| \* | 2 | Вторая степень приоритетности |
| / | 2 | Вторая степень приоритетности |
| + | 3 | Наименьший приоритет (выполняется в последнюю очередь) |
| - | 3 | Наименьший приоритет (выполняется в последнюю очередь) |

## **a.** **Польская запись и принцип ее построения**

Польская запись — это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Ее отличие заключается в том, что знаки операций пишутся не между аргументами, а до или после них. Алгоритм построения польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.

## **b.** **Программная реализация обработки выражений**

Программная реализация обработки выражений происходит по алгоритму, описанному в пункте 6.2.

## **c.** **Контрольный пример**

Контрольный пример на основе исходного кода из приложения А польской записи выражений представлен в приложении Е. В качестве примера хода разбора выражения в польский формат записи приведена таблица 6.2 и рисунок 6.1.

Таблица 6.2 – Пример конвертации выражения в польскую запись

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| 3\*(5-2)+3/6 |  |  |
| \*(5-2)+3/6 |  | 3 |
| (5-2)+3/6 | \* | 3 |
| 5-2)+3/6 | \*( | 3 |
| -2)+3/6 | \*( | 35 |
| 2)+3/6 | \*(- | 35 |
| )+3/6 | \*(- | 352 |
| +3/6 | \* | 352- |
| 3/6 | **+** | 352-\* |
| /6 | **+** | 352-\*3 |
| 6 | **+/** | 352-\*3 |
|  | **+/** | 352-\*36 |
|  |  | 352-\*36/+ |

Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления.

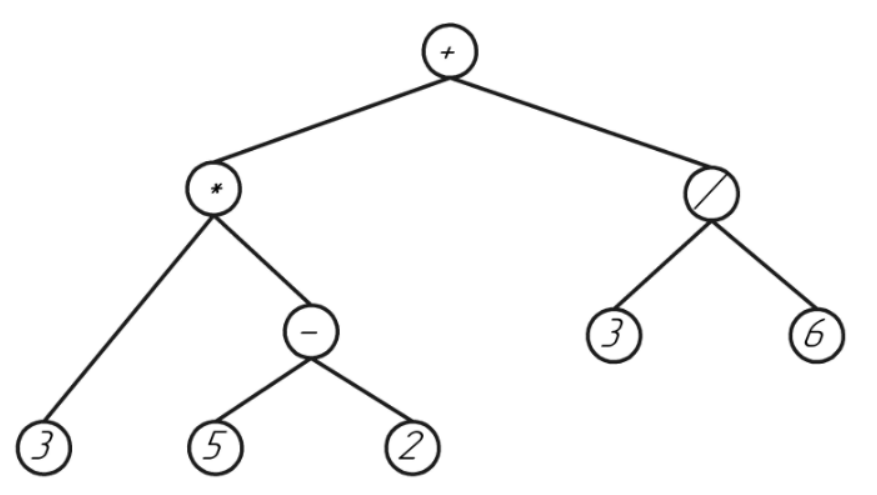


Рисунок 6.1 – Пример конвертации выражения в польскую запись

## **7 Генерация кода**

Интерпретация кода – это часть процесса трансляции. Транслятор интерпретирует код на языке NEA-2018 в код на языке JavaScript на основе таблицы лексем и информации из файла с исходным кодом для последующей его интерпретации в браузере. Схематично интерпретация кода показана на рисунке 7.1.Интерпретатор начинает свою работу только в том случае если код на языке NEA-2018 прошёл предыдущие компоненты транслятора без ошибок.



Рисунок 7.1 – Структура интерпретатора

Трансляция будет осуществляться функцией Generation, принимающей таблицу лексем и таблицу идентификаторов. Выходным будет являться файл, генерируемый во время трансляции, - “Gen.html”, который будет содержать в себе транслированный исходный код.

Алгоритм генерации исходного кода на языке NEA-2018:

* В файл “Out.html” записываются начальные теги html-документа, представленные в таблице 7.1, для корректного вывода информации.

Таблица 7.1 – Начальные теги html-документа

|  |
| --- |
| fout << "<html>" << endl;  fout << "<head>" << endl;  fout << "</head>" << endl;  fout << "<title>" << endl;  fout << "NEA-2018" << endl;  fout << "</title" << endl;  fout << "<body>" << endl;  fout << "<header>" << endl;  fout << "</header>" << endl;  fout << "<main>" << endl;  fout << "<script>" << endl << endl; |

* Далее каждый элемент таблицы лексем проверяется с помощью switch на соответствие с зарезервированными лексемами.
* Если соответствие найдено, то в выходной файл “Gen.html” записывается соответствующее выражение. Пример данной операции представлен в таблице 7.2.

Таблица 7.2 – Часть кода генерации

|  |
| --- |
| switch (Lextable.table[i].lexema)  {  case LEX\_OUT:{fout << "document.output" << endl; break;}  case tochka:{fout << endl; break; }  case LEX\_FUNCTION:{fout << "function ";break;}  …  } |

* Когда каждая лексема из таблицы будет разобрана, в файл “Gen.html” записываются закрывающие теги html-документа, для корректного вывода информации, а также вызов главной функции, для начала работы программы.

|  |
| --- |
| fout << "</script>" << endl;  fout << "</head>" << endl;  fout << "</html>" << endl; |

* Закрытие файла для записи.

## **Тестирование транслятора**

## **8.1** **Тестирование контрольного примера**

В результате обработки исходного кода программы, транслятор генерирует общий протокол работы, куда выносится структура ошибок и предупреждений.

В результате обработки транслятором исходного кода программы, представленного в приложении А, формируется информирование об ошибках в общем протоколе работы.

## **8.2** **Тестирование программ с ошибками**

Транслятор языка NEA-2018 представляет диагностику и выявление ошибок на разных этапах трансляции. Ниже будут приведены результаты обработки транслятором исходного кода с заранее допущенными ошибками.

Тестирование ошибок на этапе лексического анализатора представлено в таблице 8.1. и на рисунке 8.1.

Таблица 8.1 – Пример исходного кода программы с лексической ошибкой

|  |
| --- |
| main  {  integer lex;  integer lex;  lex = 19;  output(lex);  } |



Рисунок 8.1 – Формируемое сообщение об ошибке

Тестирование ошибок на этапе семантического анализатора представлено в таблице 8.2. и на рисунке 8.2.

Таблица 8.2 – Пример исходного кода программы с семантической ошибкой

|  |
| --- |
| integer function semantik(integer a)  {  a = 19;  output(a);  } |

Рисунок 8.5 – Формируемое сообщение об ошибке

Тестирование ошибок на этапе синтаксического анализатора представлено в таблице 8.3. и на рисунке 8.3.

Таблица 8.3 – Пример исходного кода программы с синтаксической ошибкой

|  |
| --- |
| main  {  enroll string syntax;  syntax = syntax ++ 1;  output (syntax);  } |

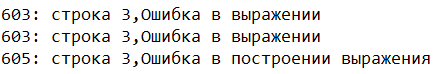


Рисунок 8.3. – Формируемое сообщение об ошибке

## **Заключение**

В данном курсовом проекте были выполнены поставленные минимальные требования. Основной целью курсовой работы было понять принцип работы языков программирования и усвоить эти знания на собственном примере. В ходе работы было изучено много нового, а также закреплены знания, которые были получены ранее. Так же стоит отметить, что полученные знания дают более глубокое понимание надстроек языков и некоторое представление о создании фреймворков. При написании приложения были усвоены такие понятия как синтаксический, лексический и семантический анализатор и многие другие.

В итоге был получен примитивный язык программирования NEA-2018, который не имеет сложных конструкций, которые реализованы на сегодняшний день во многих других языках программирования и в целом нуждается в доработке и внедрении в его структуру циклов.

Окончательная версия языка NEA-2018 включает:

* 3 типа данных;
* Поддержка операции вывода;
* Возможность вызова функций стандартной библиотеки;
* Наличие 4 арифметических операторов для вычисления выражений;
* Структурированная система для обработки ошибок пользователя.
* Поддержка операторов ветвления

Основные характеристики транслятора NEA-2018:

* Возможность обработки 4 входных параметров;
* Возможность обработки 32 ошибок;
* Реализация 15 конечных автоматов;
* Реализация 43 цепочек правил грамматики;
* Наличие порядка 3000 строк кода.

Полученные знания при выполнении курсового проекта будут способствовать последующему изучению новых технологий, так как изучение новых языков программирования уже будет проходить на уровне понимания работы языка, а не просто изучение синтаксиса.

## **Список используемых источников**

1. Польская запись [Электронный ресурс] - Режим доступа: <https://studfiles.net/preview/2792990/> . Дата посещения: 1.11.2018;
2. ASCII Table and Description [Электронный ресурс] – Режим доступа: <http://www.asciitable.com/> . Дата посещения: 7.11.2018;
3. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с. Дата посещения: 11.11.2018;
4. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с. Дата посещения: 15.11.2018;
5. Системное программное обеспечение: Учебник для вузов – СПб.: Питер, 2006.–396с.:ил. <https://books.google.by/books?id=PZc6qVbOacwC&printsec=frontcover&hl=ru#v=onepage&q&f=false> . Дата посещения: 1.12.2018;
6. Разработка компиляторов: А.А. Терехов, А.Е. Москаль, Д.Ю. Булычев, Н.Н. Вояковская, 2016. – 375с. <http://www.intuit.ru/goods_store/ebooks/8168> . Дата посещения: 12.12.2018;

## **Приложение А**

|  |  |
| --- | --- |
| 1. integer function int(integer a, integer b) 2. { 3. enroll integer c; 4. c = 5\*(a+b)\*12; 5. return c; 6. } 7. string function str(string e) 8. { 9. enroll string d; 10. d = e; 11. output(d); 12. return d; 13. } 14. integer function qwerty() 15. { 16. enroll integer c; 17. c = 1; 18. return c; 19. } 20. bool function boolean(bool a) 21. { 22. return a$true; 23. } 24. main 25. { 26. STD; 27. enroll integer g; 28. enroll integer j; 29. enroll integer s; 30. enroll integer r; 31. enroll string l; 32. enroll string p; 33. enroll bool bf; 34. enroll string probel; 35. enroll bool bl; 36. enroll bool bg; 37. probel = ' '; 38. g = 3; 39. bf = false; 40. output(bf); 41. bg = true; | 1. l =' Ctrl' ; 2. bf = bg; 3. output(strcon(p,l,probel)); 4. bl = isempty(p); 5. output(' '); 6. output(bl); 7. output(' '); 8. output(boolean(bg)); 9. output(probel); 10. output(length('aaa')); 11. g = 5; 12. r = 2; 13. output(r); 14. iftrue(!bg){ 15. output('it is a true'); 16. } 17. iffalse{ 18. output('it is a false'); 19. } 20. j = int(g, r); 21. g = 5+int(j, r); 22. p = str(probel); 23. p = str(probel); 24. output(j); 25. p = str(l) ; 26. g = qwerty() + g + 8; 27. p = str(probel); 28. output(g); 29. } |

## **Приложение Б**

|  |  |  |  |
| --- | --- | --- | --- |
| Цепочка | Тип | Лексема | Конечный автомат |
| main | Ключевое слово | s | #define GRAPH\_START IT::NUM, 5, \  FST::NODE(1,FST::RELATION('m',1)),\  FST::NODE(1,FST::RELATION('a',2)),\  FST::NODE(1,FST::RELATION('i',3)),\  FST::NODE(1,FST::RELATION('n',4)),\  FST::NODE() |
| length | Ключевое слово | b | #define GRAPH\_KOREN 6, \  FST::NODE(1,FST::RELATION('l',1)),\  FST::NODE(1,FST::RELATION('e',2)),\  FST::NODE(1,FST::RELATION('n',3)),\  FST::NODE(1,FST::RELATION('g',4)),\  FST::NODE(1,FST::RELATION('t',5)),\  FST::NODE(1,FST::RELATION('h',5)),\  FST::NODE() |
| concat | Ключевое слово | k | #define GRAPH\_STEPEN 7,\  FST::NODE(1,FST::RELATION('c',1)),\  FST::NODE(1,FST::RELATION('o',2)),\  FST::NODE(1,FST::RELATION('n',3)),\  FST::NODE(1,FST::RELATION('c',4)),\  FST::NODE(1,FST::RELATION('a',5)),\  FST::NODE(1,FST::RELATION('t',6)),\ FST::NODE() |
| CTD | Ключевое слово | z | #define GRAPH\_CTD 8,\  FST::NODE(1,FST::RELATION('C',1)),\  FST::NODE(1,FST::RELATION('T',2)),\  FST::NODE(1,FST::RELATION('D',3)),\  FST::NODE() |
| function | Ключевое слово | f | #define GRAPH\_FUNCTION 9, \  FST::NODE(1, FST::RELATION('f', 1)),\  FST::NODE(1, FST::RELATION('u', 2)),\  FST::NODE(1, FST::RELATION('n', 3)),\  FST::NODE(1, FST::RELATION('c', 4)),\  FST::NODE(1, FST::RELATION('t', 5)),\  FST::NODE(1, FST::RELATION('i', 6)),\  FST::NODE(1, FST::RELATION('o', 7)),\  FST::NODE(1, FST::RELATION('n', 8)),\  FST::NODE() |
| Идентификатор |  | i | #define GRAPH\_ID 2, \  FST::NODE(26, FST::RELATION('a',1), FST::RELATION('b',1), \  FST::RELATION('c', 1), FST::RELATION('d', 1), \  FST::RELATION('e', 1), FST::RELATION('f', 1), \  FST::RELATION('g', 1), FST::RELATION('h', 1), \  FST::RELATION('i', 1), FST::RELATION('j', 1), \  FST::RELATION('k', 1), FST::RELATION('l', 1), \  FST::RELATION('m', 1), FST::RELATION('n', 1), \  FST::RELATION('o', 1), FST::RELATION('p', 1), \  FST::RELATION('q', 1), FST::RELATION('r', 1), \  FST::RELATION('s', 1), FST::RELATION('t', 1), \  FST::RELATION('u', 1), FST::RELATION('v', 1), \  FST::RELATION('w', 1), FST::RELATION('x', 1), \  FST::RELATION('y', 1), FST::RELATION('z', 1)), \ |

Продолжение приложения Б

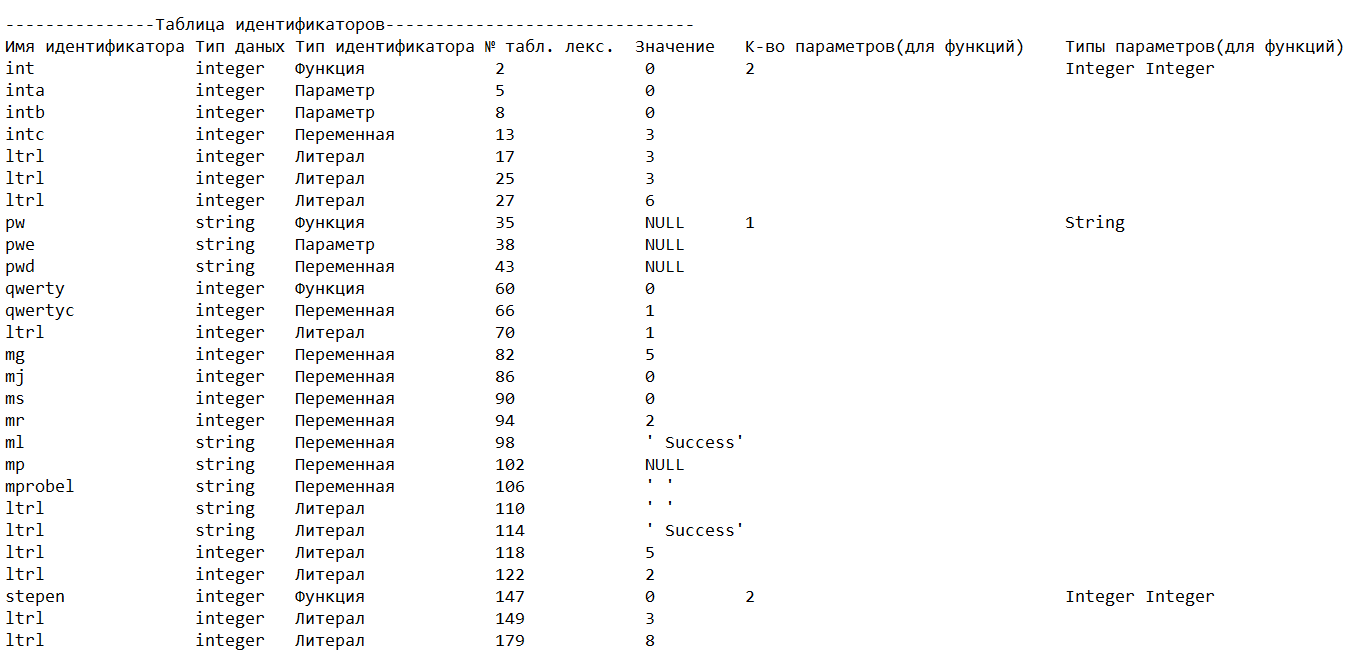
|  |  |  |  |
| --- | --- | --- | --- |
| Целочисленный литерал |  | literal | #define GRAPH\_NUMBER\_LITERAL IT::NUM, 2, \  FST::NODE(10,FST::RELATION('0',1),FST::RELATION('1',1),\  FST::RELATION('3',1),FST::RELATION('2',1),\ FST::RELATION('5',1),FST::RELATION('4',1),\ FST::RELATION('7',1),FST::RELATION('6',1),\ FST::RELATION('8',1),FST::RELATION('9',1)),\  FST::NODE(10,FST::RELATION('0',1),FST::RELATION('1',1),\ FST::RELATION('3',1),FST::RELATION('2',1),\ FST::RELATION('5',1),FST::RELATION('4',1),\ FST::RELATION('7',1),FST::RELATION('6',1),\ FST::RELATION('8',1),FST::RELATION('9',1)) |
| Строковый литерал |  | integer | #define GRAPH\_STRING\_LITERAL IT::STR, 3, \  FST::NODE(1, FST::RELATION('\'',1)),\  FST::NODE(64,FST::RELATION('a',1),FST::RELATION('b',1), \  FST::RELATION('c', 1), FST::RELATION('d', 1), \  FST::RELATION('e', 1), FST::RELATION('f', 1), \  FST::RELATION('g', 1), FST::RELATION('h', 1), \  FST::RELATION('i', 1), FST::RELATION('j', 1), \  FST::RELATION('k', 1), FST::RELATION('l', 1), \  FST::RELATION('m', 1), FST::RELATION('n', 1), \  FST::RELATION('o', 1), FST::RELATION('p', 1), \  FST::RELATION('q', 1), FST::RELATION('r', 1), \  FST::RELATION('s', 1), FST::RELATION('t', 1), \  FST::RELATION('u', 1), FST::RELATION('v', 1), \  FST::RELATION('w', 1), FST::RELATION('x', 1), \  FST::RELATION('y', 1), FST::RELATION('z', 1), \  FST::RELATION('0', 1), FST::RELATION('1',1), \  FST::RELATION('3', 1), FST::RELATION('2',1), \  FST::RELATION('5', 1), FST::RELATION('4',1), \  FST::RELATION('7', 1), FST::RELATION('6',1), \  FST::RELATION('9', 1), FST::RELATION('8',1), \  FST::RELATION('A', 1), FST::RELATION('B', 1), \  FST::RELATION('C', 1), FST::RELATION('D', 1), \  FST::RELATION('E', 1), FST::RELATION('F', 1), \  FST::RELATION('G', 1), FST::RELATION('H', 1), \  FST::RELATION('I', 1), FST::RELATION('J', 1), \  FST::RELATION('K', 1), FST::RELATION('L', 1), \  FST::RELATION('M', 1), FST::RELATION('N', 1), \  FST::RELATION('O', 1), FST::RELATION('P', 1), \  FST::RELATION('Q', 1), FST::RELATION('R', 1), \  FST::RELATION('S', 1), FST::RELATION('T', 1), \  FST::RELATION('U', 1), FST::RELATION('V', 1), \  FST::RELATION('W', 1), FST::RELATION('X', 1), \  FST::RELATION('Y', 1), FST::RELATION('Z', 1), \  FST::RELATION(' ', 1), FST::RELATION('\'',2)),\  FST::NODE() |
| string | Ключевое  слово | string | #define GRAPH\_STRING IT::STR, 7, \  FST::NODE(1,FST::RELATION('s',1)),\  FST::NODE(1,FST::RELATION('t',2)),\  FST::NODE(1,FST::RELATION('r',3)),\  FST::NODE(1,FST::RELATION('i',4)),\  FST::NODE(1,FST::RELATION('n',5)),\  FST::NODE(1,FST::RELATION('g',6)),\  FST::NODE() |
| bool | Ключевое слово | bool | #define GRAPH\_CREATE 5, \  FST::NODE(1, FST::RELATION('b', 1)),\  FST::NODE(1, FST::RELATION('o', 2)),\  FST::NODE(1, FST::RELATION('o', 3)),\  FST::NODE(1, FST::RELATION('l', 4)),\  FST::NODE() |

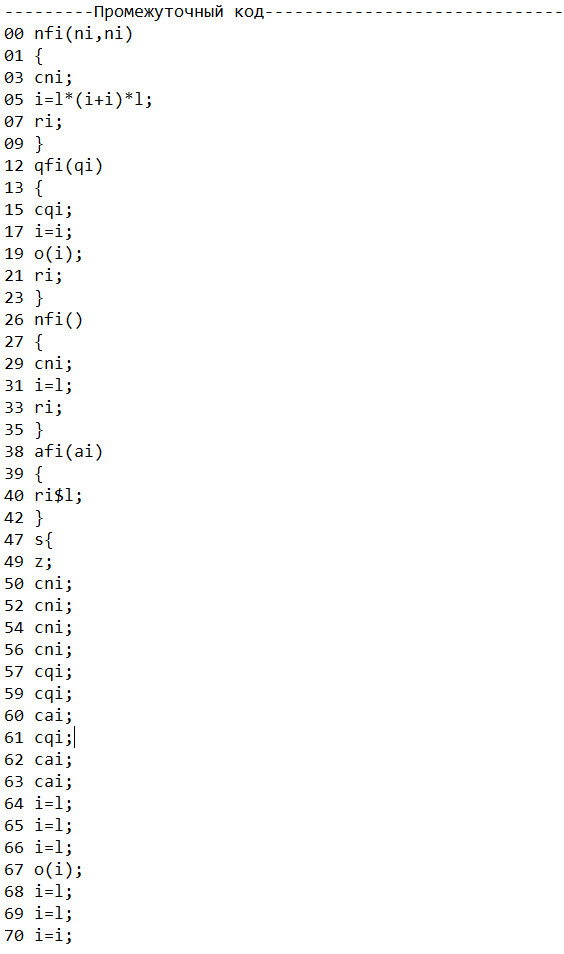
Продолжение приложения Б

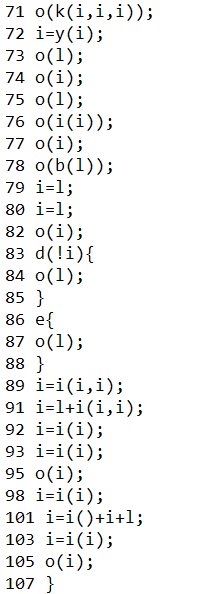
|  |  |  |  |
| --- | --- | --- | --- |
| integer | Ключевое слово | n | #define GRAPH\_INTEGER IT::NUM, 8, \  FST::NODE(1,FST::RELATION('i',1)),\  FST::NODE(1,FST::RELATION('n',2)),\  FST::NODE(1,FST::RELATION('t',3)),\  FST::NODE(1,FST::RELATION('e',4)),\  FST::NODE(1,FST::RELATION('g',5)),\  FST::NODE(1,FST::RELATION('e',6)),\  FST::NODE(1,FST::RELATION('r',7)),\  FST::NODE() |
| return | Ключевое слово | r | #define GRAPH\_RETURN 7, \  FST::NODE(1,FST::RELATION('r',1)),\  FST::NODE(1,FST::RELATION('e',2)),\  FST::NODE(1,FST::RELATION('t',3)),\  FST::NODE(1,FST::RELATION('u',4)),\  FST::NODE(1,FST::RELATION('r',5)),\  FST::NODE(1,FST::RELATION('n',6)),\  FST::NODE() |
| output | Ключевое слово | o | #define GRAPH\_OUT 6, \  FST::NODE(1, FST::RELATION('o', 1)),\  FST::NODE(1, FST::RELATION('u', 2)),\  FST::NODE(1, FST::RELATION('t', 3)),\  FST::NODE(1, FST::RELATION('p', 4)),\  FST::NODE(1, FST::RELATION('u', 5)),\  FST::NODE(1, FST::RELATION('t', 6)),\  FST::NODE() |
| +,-,\*,/ | Операторы | S | #define GRAPH\_SEPARATORS 2,\  FST::NODE(13, FST::RELATION(';',1), FST::RELATION('=',1),\ FST::RELATION(',',1), FST::RELATION('{',1),\  FST::RELATION('}',1), FST::RELATION('(',1),\ FST::RELATION(')',1), FST::RELATION('\*',1),\  FST::RELATION(']',1), FST::RELATION('[',1),\  FST::RELATION('+',1), FST::RELATION('-',1),\  FST::RELATION('.',1), FST::RELATION('!',1),\  FST::RELATION('/',1)),\  FST::NODE() |
| = | Оператор | = |
| ., {}() | Сепараторы | ., {}() |
| Новая строка | Сепаратор | | |

## **Приложение В**

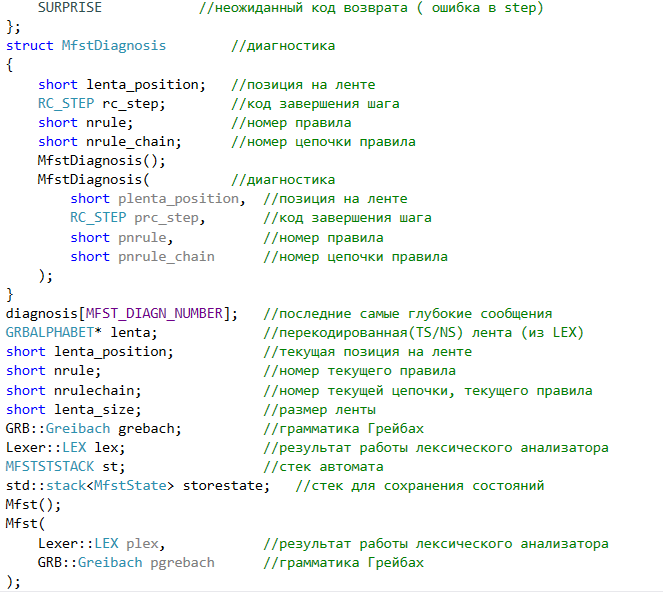
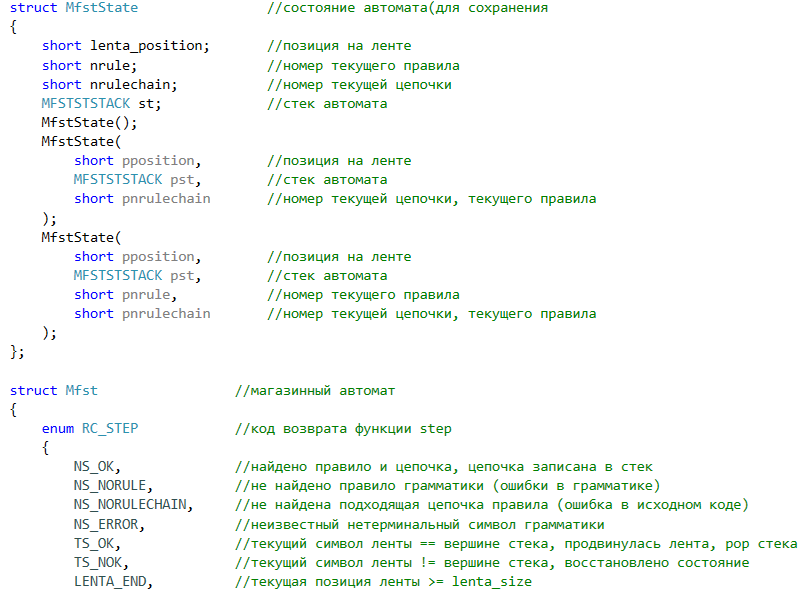
|  |  |  |  |
| --- | --- | --- | --- |
|  | |  | |
|  | |  | |
|  | |  | | |
|  | |  | | |

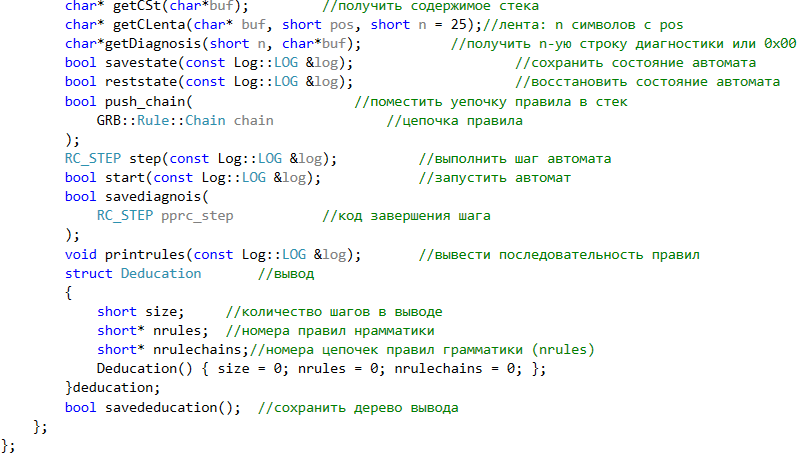




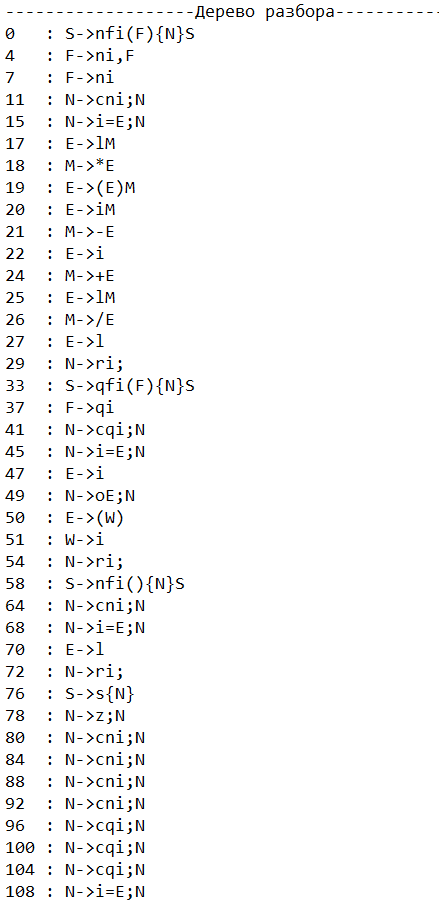


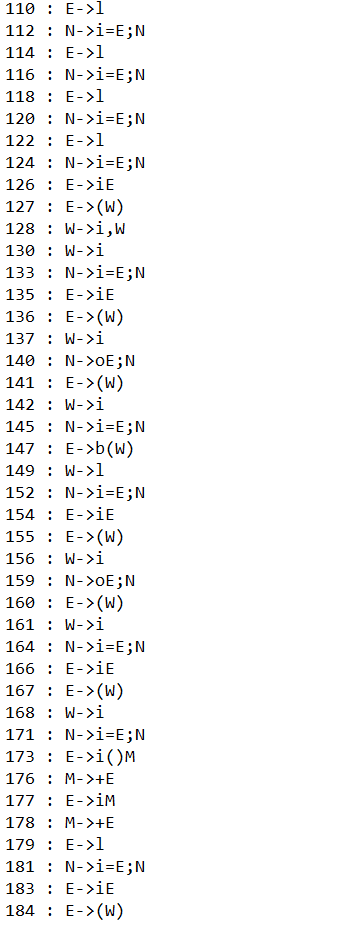
**Приложение Г**

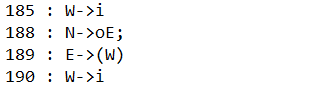




## **Приложение Д**



****

****

|  |  |
| --- | --- |
|  |  |

## **Приложение Е**

