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 MSV-2022»

Выполнил студент Маркович Святослав Витальевич

(Ф.И.О.)

Руководитель проекта ст. пр. Наркевич Аделина Сергеевна

(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 ст. пр. Наркевич Аделина Сергеевна

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

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

Нормоконтролер ст. пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Содержание

[Введение 4](#_Toc89645191)

[Глава 1. Спецификация языка программирования 5](#_Toc89645192)

[1.1 Характеристика языка программирования 5](#_Toc89645193)

[1.2 Алфавит языка 5](#_Toc89645194)

[1.3 Символы сепараторы 5](#_Toc89645195)

[1.4 Применяемые кодировки 5](#_Toc89645196)

[1.5 Типы данных 6](#_Toc89645197)

[1.6 Преобразование типов данных 7](#_Toc89645198)

[1.7 Идентификаторы 7](#_Toc89645199)

[1.8 Литералы 7](#_Toc89645200)

[1.9 Область видимости идентификаторов 7](#_Toc89645201)

[1.10 Инициализация данных 8](#_Toc89645202)

[1.11 Инструкции языка 8](#_Toc89645203)

[1.12 Операции языка 9](#_Toc89645204)

[1.13 Выражения и их вычисления 9](#_Toc89645205)

[1.14 Программные конструкции языка 9](#_Toc89645206)

[1.15 Область видимости 10](#_Toc89645207)

[1.16 Семантические проверки 10](#_Toc89645208)

[1.17 Распределение оперативной памяти на этапе выполнения 10](#_Toc89645209)

[1.18 Стандартная библиотека и её состав 10](#_Toc89645210)

[1.19 Ввод и вывод данных 10](#_Toc89645211)

[1.20 Точка входа 10](#_Toc89645212)

[1.21 Препроцессор 11](#_Toc89645213)

[1.22 Соглашения о вызовах 11](#_Toc89645214)

[1.23 Объектный код 11](#_Toc89645215)

[1.24 Классификация сообщений транслятора 11](#_Toc89645216)

[1.25 Контрольный пример 11](#_Toc89645217)

[Глава 2. Структура транслятора 12](#_Toc89645218)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 12](#_Toc89645219)

[2.2 Перечень входных параметров 13](#_Toc89645220)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 13](#_Toc89645221)

[Глава 3. Разработка лексического анализатора 14](#_Toc89645222)

[3.1 Структура лексического анализатора 14](#_Toc89645223)

[3.2 Контроль входных символов 14](#_Toc89645224)

[3.3 Удаление избыточных символов 15](#_Toc89645225)

[3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов 15](#_Toc89645226)

[3.5 Основные структуры данных 16](#_Toc89645227)

[3.6 Принцип обработки ошибок 17](#_Toc89645228)

[3.7 Структура и перечень сообщений лексического анализатора 17](#_Toc89645229)

[3.8 Параметры лексического анализатора и режимы его работы 17](#_Toc89645230)

[3.9 Алгоритм лексического анализа 17](#_Toc89645231)

[3.10 Контрольный пример 17](#_Toc89645232)

[Глава 4. Разработка синтаксического анализатора 18](#_Toc89645233)

[4.1 Структура синтаксического анализатора 18](#_Toc89645234)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 18](#_Toc89645235)

[4.3 Построение конечного магазинного автомата 20](#_Toc89645236)

[4.4 Основные структуры данных 21](#_Toc89645237)

[4.5 Описание алгоритма синтаксического разбора 21](#_Toc89645238)

[4.6 Структура и перечень сообщений синтаксического анализатора 21](#_Toc89645239)

[4.7 Параметры синтаксического анализатора и режимы его работы 22](#_Toc89645240)

[4.8 Принцип обработки ошибок 22](#_Toc89645241)

[4.9 Контрольный пример 22](#_Toc89645242)

[Глава 5. Разработка семантического анализатора 23](#_Toc89645243)

[5.1 Структура семантического анализатора 23](#_Toc89645244)

[5.2 Функции семантического анализатора 23](#_Toc89645245)

[5.3 Структура и перечень сообщений семантического анализатора 23](#_Toc89645246)

[5.4 Принцип обработки ошибок 23](#_Toc89645247)

[5.5 Контрольный пример 24](#_Toc89645248)

[Глава 6. Преобразование выражений 25](#_Toc89645249)

[6.1 Выражения, допускаемые языком 25](#_Toc89645250)

[6.2 Польская запись 25](#_Toc89645251)

[6.3 Программная реализация обработки выражений 26](#_Toc89645252)

[6.4 Контрольный пример 26](#_Toc89645253)

[Глава 7. Генерация кода 27](#_Toc89645254)

[7.1 Структура генератора кода 27](#_Toc89645255)

[7.2 Представление типов данных в оперативной памяти 27](#_Toc89645256)

[7.3 Статическая библиотека 27](#_Toc89645257)

[Глава 8. Тестирование транслятора 29](#_Toc89645258)

[8.1 Тестирование фазы проверки на допустимость символов 29](#_Toc89645259)

[8.2 Тестирование лексического анализатора 29](#_Toc89645260)

[8.3 Тестирование синтаксического анализатора 29](#_Toc89645261)

[8.4 Тестирование семантического анализатора 30](#_Toc89645262)

[Приложение А 31](#_Toc89645263)

[Приложение Б 33](#_Toc89645264)

[Приложение В 36](#_Toc89645265)

[Приложение Г 40](#_Toc89645266)

[Приложение Д 42](#_Toc89645267)

[Приложение Е 44](#_Toc89645268)

[Приложение Ж 46](#_Toc89645269)

[Приложение З 47](#_Toc89645270)

[Литература 50](#_Toc89645271)

# **Введение**

Задачей данного курсового проекта была поставлена разработка транслятора своего языка программирования MSV-2022. Он предназначен для выполнения простейших операций и арифметических действий над строками и числами.

Главной задачей транслятора является проверка исходного кода на языке MSV-2022 и генерация на его основе JavaScript кода для последующего выполнения программы .

Исходя из цели курсового проекта, были определены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка выражений с помощью польской инверсии;

– генерация кода в javaScript;

– тестирование транслятора;

Способы решения каждой задачи будут описаны в соответствующих главах курсового проекта.

В первой главе работы определена спецификация языка программирования.

Во второй главе представлена структура транслятора. В ней перечислены компоненты транслятора, их назначения и принципы взаимодействия.

В третьей главе описана разработка лексического анализатора, который создаёт таблицы лексем и идентификаторов.

В четвертой главе описана разработка синтаксического анализатора, который выполняет разбор исходного кода в соответствии с правилами языка программирования.

В пятой главе описан семантический анализатор, которые проверяет исходный код программы на наличие семантических ошибок.

В шестой главе описан способ преобразования выражений в польский формат.

В седьмой главе представлена генерация кода в javaScript с помощью таблиц лексем и идентификаторов

В восьмой главе описывается тестирование транслятора

# **Глава 1. Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

Язык программирования MSV-2022 – это строго типизируемый, процедурный язык высокого уровня, который транслируется в javaScript.

## **1.2 Алфавит языка**

Алфавит языка программирования – набор символов, которые могут использоваться при написании исходного кода.

Язык программирования MSV-2022 включает в себя кириллицу и символы латинского алфавита верхнего и нижнего регистров, арабские цифры, знаки препинания, знаки арифметических и логических операций.

## **1.3 Символы сепараторы**

Сепараторы необходимы для разделения операций языка. Сепараторы, используемые в языке программирования MSV-2022, приведены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ;  « » (пробел)  ' | Разделение конструкций |
| =  +  -  \*  % | Арифметические операции |
| >  < | Логические операции |
| {} | Программный блок инструкций |
| () | Параметры функций, изменение приоритетов в выражениях, условие цикла |

## **1.4 Применяемые кодировки**

Для написания исходного кода на языке программирования MSV-2022 используется кодировка Windows-1251 – набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии – представленная на рисунке 1.1.

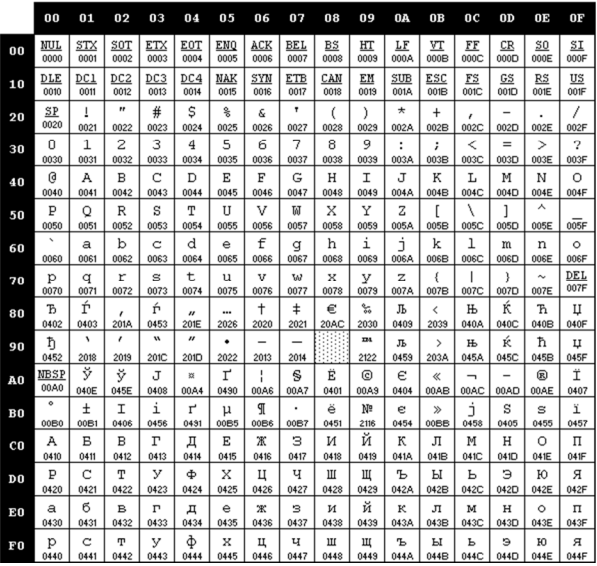


Рисунок 1.1 – Кодировка Windows-1251

## **1.5 Типы данных**

В языке программирования MSV-2022 используются три основных типа данных, которые описываются в таблице 1.2. Пользовательские типы данных не поддерживаются.

Таблица 1.2 – Типы данных

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| int | Фундаментальный тип данных, используемый для объявления целочисленных данных. Без явно указанной инициализации переменной, присваивается нулевое значение. |
| string | Фундаментальный тип данных, используемый для объявления строк. Без явно указанной инициализации переменной, присваивается нулевое значение (пустая строка). |

## **1.6 Преобразование типов данных**

В языке программирования MSV-2022 преобразование типов данных не поддерживается.

## **1.7 Идентификаторы**

Идентификаторы применяются для наименования переменных, функция и параметров. Идентификаторы, объявленные внутри функционального блока, получают префикс, который отображается в таблице идентификаторов. Предусмотрены несколько правил составления идентификатора:

– состоит из символов латинского алфавита любого регистра и цифр;

– могут начинаться с символа нижнего подчеркивания «\_»;

– максимальная длина идентификатора равна 5;

– идентификатор не может совпадать с ключевыми словами языка программирования.

– регулярное выражение для разбора идентификатора выглядит следующим образом: [\_]\*[a-zA-Z0-9\_]+

## **1.8 Литералы**

В языке программирования MSV-2022 существует только 2 типа литералов: целые и символьные. Их краткое описание представлено в таблице 1.3.

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные литералы. Могут быть представлены в двоичной системе счисления. |
| Строковые литералы | Символы, заключенные в одинарные кавычки (''). Максимальное число символов в строковом литерале – 255. |

## **1.9 Область видимости идентификаторов**

Область видимости в языке программирования MSV-2022 работает по принципу C++ («сверху вниз»). Перед использованием переменной необходимо её объявление. Допускается использование переменной только внутри её области видимости. Допускается объявление переменных с одинаковыми именами в разных программных блоках.

## **1.10 Инициализация данных**

В языке программирования MSV-2022 отсутствует инициализация и значение переменным можно присвоить только после объявления

## **1.11 Инструкции языка**

Все возможные инструкции языка программирования MSV-2022 представлены в общем виде в таблице 1.4.

Таблица 1.4 – Инструкции языка программирования MSV-2022

|  |  |
| --- | --- |
| Инструкция | Запись |
| Объявление переменной | declare <тип данных> <идентификатор>; |
| Присваивание | <идентификатор> = <выражение >; |
| Объявление функции | <тип данных> function <идентификатор> ([<тип данных> <идентификатор>][, <тип данных> <идентификатор>]\*) {…} |
| Блок инструкций | {  …  } |
| Возврат из функции | return <выражение>; |
| Вывод данных | print <выражение>; |
| Оператор цикла | echo (<условие>)  {…} |

## **1.12 Операции языка**

Язык программирования MSV-2022 может выполнять арифметические и логические операции, представленные в таблице 1.5.

Таблица 1.5 – Операции и их приоритеты

|  |  |
| --- | --- |
| Операция | Приоритет операции |
| (  >  < | 0 |
| , | 1 |
| +  - | 2 |
| \*  / (деление с остатком)  % (деление по модулю) | 3 |
| ) | 4 |

## **1.13 Выражения и их вычисления**

В выражении должны участвовать операторы и операнды одного типа, а также функции, возвращающие значения того же типа. Круглые скобки в выражении используются для изменения приоритетов операций. Не допускается запись двух подряд арифметических операций. Также круглые скобки могут использоваться для передачи параметров функций . Фигурные скобки содержат блоки кода функций и циклов.

## **1.14 Программные конструкции языка**

Ключевые программные конструкции языка программирования MSV-2022 представлены в таблице 1.6

Таблица 1.6 – программные конструкции

|  |  |
| --- | --- |
| Главная функция (точка входа в приложение) | main  {…} |
| Функция | <тип данных> function <идентификатор> ([<тип данных> <идентификатор>][, <тип данных> <идентификатор>]\*)  {  …  return <выражение>;  } |

## **1.15 Область видимости**

В языке программирования MSV-2022 переменные обязаны находиться внутри программного блока функций. Внутри разных областей видимости разрешено объявление переменных с одинаковыми именами. Все переменные, параметры или функции внутри области видимости получают префикс, который отображается в таблице идентификаторов. Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

## **1.16 Семантические проверки**

Перечень семантических проверок, предусмотренных языком, приведён в таблице 1.7.

Таблица 1.7 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы функций не должны повторяться |
| 2 | Идентификаторы в одной области видимости не должны повторяться |
| 3 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении |
| 4 | Тип данных функции должен совпадать с её возвращаемым значением |
| 5 | Идентификатор должен быть объявлен до его использования |
| 6 | Операнды в арифметическом выражении не могут быть разных типов |

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

Все переменные размещаются в структуре Entry. Таблицы лексем и идентификаторов размещены в структуры с выделенной под них оперативной памятью, которая очищается по окончанию работы транслятора.

## **1.18 Стандартная библиотека и её состав**

Стандартными функциями языка программирования MSV-2022 являются функции для работы со строками(concatstr(str result, str firstconc, str secondconc) – конкатенация строк и copystr(str somestring, str stringtocopy) – копирование строк)

## **1.19 Ввод и вывод данных**

В языке программирования MSV-2022 ввод данных не поддерживается. Вывод данных происходит с помощью функции print(<выражение>);

## **1.20 Точка входа**

Точкой входа в языке программирования MSV-2022 является функция main.

## **1.21 Препроцессор**

В языке программирования MSV-2022 имеется препроцессор. Директива $include[<имя библиотеки>] подключает выбранную библиотеку языка MSV-2022.

## **1.22 Соглашения о вызовах**

Во всех подпрограммах используется стандартное соглашение о вызовах

## **с**

Язык программирования MSV-2022 транслируется в JavaScript, после чего Движок компилирует JavaScript-код в машинные инструкции в ходе исполнения программы

## **1.24 Классификация сообщений транслятора**

В случае возникновения ошибки в исходном коде программы на языке программирования MSV-2022 и выявлении её транслятором в файл протокола выводится сообщение. Классификация обрабатываемых ошибок приведена в таблице 1.9.

Таблица 1.9 – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-109 | Системные ошибки |
| 100-199 | Ошибки лексического анализа |
| 600-699 | Ошибки семантического анализа |
| 300-399 | Ошибки синтаксического анализа |

## **1.25 Контрольный пример**

Контрольные примеры представлены в приложении А.

# **Глава 2. Структура транслятора**

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

Транслятор преобразует программу, написанную на языке программирования MSV-2022 в программу на языке JavaScript. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода. Принцип взаимодействия представлен на рисунке 2.1.

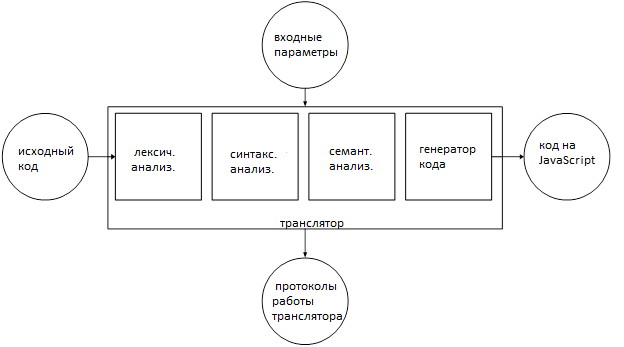


Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. На вход лексический анализатор получает исходный код на языке программирования MSV-2022.

Задачей лексического анализатора является нахождение лексических ошибок и формирование таблиц лексем и идентификаторов.

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

Семантический анализ, в свою очередь, является проверкой исходного кода программы на семантическую согласованность с определением конструкций языка, то есть проверяет правильность текста исходной программы с точки зрения семантики.

Генератор кода – этап транслятора, выполняющий генерацию кода на языке JavaScript на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует кода на языке программирования MSV-2022 в код JavaScript

## 

## **2.2 Перечень входных параметров**

Таблица 2.1 Входные параметры транслятора языка MSV-2022

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя файла> | Входной файл с расширением .txt, в котором содержится исходный код на языке MSV-2022 | Не предусмотрено |
| -log:<имя файла> | Файл для записи протокола работы транслятора. | <имя файла>.log |
| -out:<имя файла> | Файл для записи программы на языке MSV-2022 с заменёнными символами табуляции. | <имя файла>.out |

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

Перечень протоколов, формируемых транслятором языка программирования MSV-2022 и их назначением представлен в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка MSV-2022

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала с параметром <log> | Содержит информацию о времени выполнения приложения; входных параметрах в приложение; промежуточное представление кода; трассировку синтаксического анализа; дерево разбора и время выполнения разбора; промежуточное представление кода после приведения его к польской нотации. |
| Выходной файл с параметром <out> | код на языке программирования MSV-2022 с сепараторами и без избыточных пробелов, табуляций и переходов на новую строку. |
| Файл, генерирующийся во время работы лексического анализатора <TableOfLexems.txt> | таблица лексем . |
| Файл, генерирующийся во время работы лексического анализатора <TableOfIdentificators.txt> | таблица идентификаторов. |
| Файл, генерирующийся во время работы преобразователя выражений к польской нотации анализатора <TableOfLexemsPN.txt> | таблица лексем с выражениями, приведёнными к виду польской нотации. |

# **Глава 3. Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает на вход исходный код языка программирования MSV-2022. Если лексический анализатор не может разобрать отдельные последовательности символов, он генерирует исключение. При успешном анализе на выходе формируются таблицы лексем и идентификаторов. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 Структура лексического анализатора MSV-2022

## **3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2.

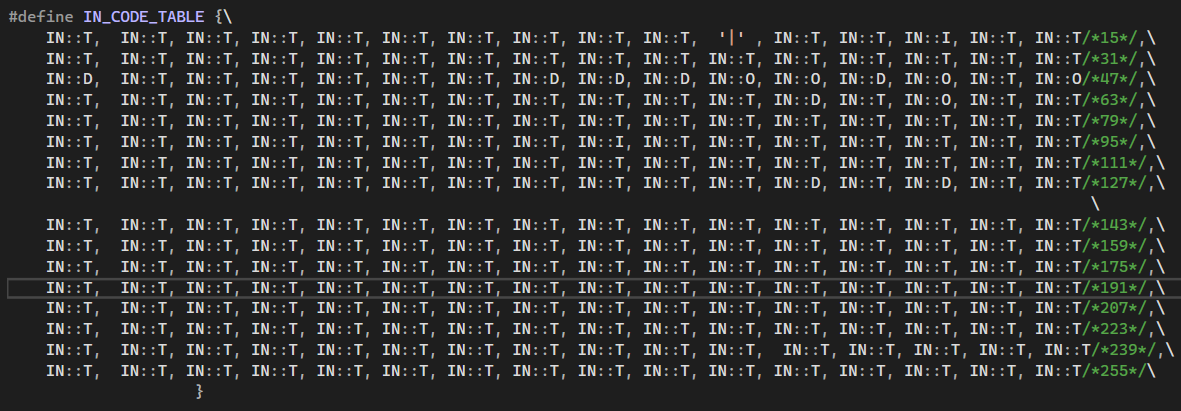


Рисунок 3.2. Таблица контроля входных символов

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции, пробела и все символы, идущие после символа начала комментария (#), если он не включен в строковый литерал. Избыточные символы удаляются перед этапом разбиения исходного кода на лексемы.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем исходный код, занесенный в структуру In.

2. Встреча пробела, табуляции является своего рода встречей символа-сепаратора.

3. В отличие от других символов-сепараторов не заносим их в таблицу лексем, то есть они игнорируются.

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

Лексемы – это специальные символы, которые соответствуют ключевым словам, символам арифметических и логических операций, сепараторам и т.д. Они необходимы для упрощения дальнейшего анализа исходного кода программы. Соответствие цепочек символов и лексем приведено в таблице 3.1.

Таблица 3.1 Соответствие цепочек с лексемами

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | declare | d |
| int, string, vacuum | t |
| main | m |
| function | f |
| return | r |
| print | p |
| echo | w |
| Данные | Лексема | l |
| Идентификатор | i |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| Операции | + | + |
| - | - |
| \* | \* |
| / | / |
| % | % |
| < | < |
| > | > |
| Присваивания (=) | = |

Пример реализации таблицы лексем представлен в приложении Б.

Также в приложении В находятся конечные автоматы ключевых слов.

## **3.5 Основные структуры данных**

Основная структура данных, используемая для хранения таблицы идентификаторов представлена в листинге 3.1.

|  |
| --- |
| struct Entry  {  char parrent\_function[ID\_MAXSIZE + 5];  int firstApi;  char id[ID\_MAXSIZE + 5];  IDDATATYPE iddatatype;  IDTYPE idtype;  union  {  int vint;  char operation = '\0';  struct  {  unsigned char len;  char str[TI\_STR\_MAXSIZE];  } vstr;  } value; |

Листинг 3.1. — Структура таблицы идентификаторов

Основная структура данных, используемая для хранения таблицы лексем представлена в листинге 3.2

|  |
| --- |
| struct Entry // строка таблицы лексем  {  char lexema; // лексема  int sn; // номер строки в исходном тексте  int idxTI; // индекс в таблице идентификаторов или LT\_TI\_NULLIDX  Entry();  Entry(const char lex, int str\_n, int idxTI);  };  struct LexTable // экзепляр таблицы лексем  {  int maxsize; // ёмкость таблицы лексем < LT\_MAXSIZE  int size; // текущий размер таблицы лексем < maxsize  Entry\* table; // массив строк таблицы лексем  Entry GetEntry( // получить строку таблицы лексем  int n // номер получаемой строки  );  void PrintLexTable(const wchar\_t\* in);  LexTable();  }; |

Листинг 3.2. — Структура таблицы лексем

## **3.6 Принцип обработки ошибок**

При нахождении ошибки останавливается работа транслятора и выводится сообщение в консоль либо заносится информация об ошибке в файл протокола.

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

Перечень сообщений лексического анализатора представлен в листинге 3.3.

|  |
| --- |
| ERROR\_ENTRY(120, "Таблица лексем: превышен допустимый размер таблицы"),  ERROR\_ENTRY(121, "Превышена длина имя переменной или недопустимое имя переменной"),  ERROR\_ENTRY(122, "Таблица идентификаторов: превышен допустимый размер таблицы"),  ERROR\_ENTRY(123, "Лексический анализ: нераспознанная лексема"),  ERROR\_ENTRY(124, "Лексический анализ: нарушен формат строкового литерала"),  ERROR\_ENTRY(125, "Лексический анализ: перезапись идентификатора"),  ERROR\_ENTRY(126, "Лексический анализ: неизвестная переменная"),  ERROR\_ENTRY(127, "Лексический анализ: превышена длина строкового литерала"),  ERROR\_ENTRY(128, "Не удалось создать файл с таблицами лексем и идентификаторов"),  ERROR\_ENTRY(129, "Лексический анализ: найдено несколько точек входа (main)"),  ERROR\_ENTRY(130, "Лексический анализ: отсутствие ключевого слова declare при объявлении переменной"), |

Листинг 3.3 – Перечень ошибок лексического анализатора

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

Входным параметром функции лексического анализатора является исходный текст программы, написанный на языке MSV-2022, а также файлы протокола. Исходный текст представляется в виде таблиц лексем и идентификаторов для последующего анализа и генерации кода кода. В файлы протокола вносится информация об работе транслятора(время выполнения, список входных параметров, список ошибок)

## **3.9 Алгоритм лексического анализа**

Лексически анализ выполняется программой, входящей в состав транслятора, называемой лексическим анализатором. Цель лексического анализатора – распознавание цепочек символов языка и деление их на лексемы. Распознавание цепочек происходит благодаря конечным автоматам.

Пример. Конечный автомат для распознавания ключевого слова main приведен на рисунке 3.3. S0 – начальное состояние, S4 – конечное состояние автомата.

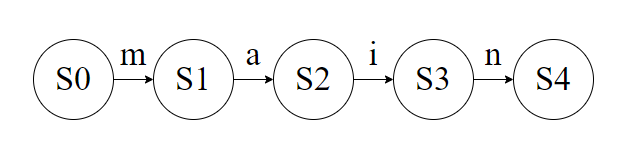


Рисунок 3.3 — Граф переходов для цепочки “main”

## **3.10 Контрольный пример**

Результат работы лексического анализатора – вывод файла протокола, построение таблиц лексем и идентификаторов, вывод таблиц в файлы – представлен в приложении Б.

# **Глава 4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ – фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций. На входе синтаксический анализатор получает таблицы лексем и идентификаторов, которые были созданы на фазе лексического анализа. На выходе при успешном разборе выводится дерево разбора. Структура синтаксического анализа представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора

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

В синтаксическом анализаторе транслятора языка MSV-2022 используется контекстно-свободная грамматика , где

T – множество терминальных символов (было описано в разделе 1.2 данной пояснительной записки),

N – множество нетерминальных символов (первый столбец таблицы 4.1),

P – множество правил языка (второй столбец таблицы 4.1),

S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика имеет нормальную форму Грейбах, т.к. она не леворекурсивная (не содержит леворекурсивных правил) и правила  имеют вид:

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

Правила языка MSV-2022 представлены в приложении Г.

TS – терминальные символы, которыми являются сепараторы, знаки арифметических операций и некоторые строчные буквы.

NS – нетерминальные символы, представленные несколькими заглавными буквами латинского алфавита.

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов MSV-2022

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | i{NrE;};  tfi(F){NrE;}S  tfi(F){NrE;}  tfi(F){N}  tfi(F){N}S | Проверка правильности структуры программы |
| N | dti;  i=E;  rE;  dtfi(F);  dti;N  rE;N  i=E;N  dtfi(F);N  pE;  pE;N  w(Z){K};  w(Z){K};N  c(i,P,P);  c(i,P,P);N  C(i,P);  C(i,P);N | Проверка на правильность построения конструкции операторов |
| E | i  l  (E)  (E,E)  i(W)  i()  iM  lM  (E)M  i(W)M | Выражение |
| W | i  l  i,w  l,w | Подвыражение |
| F | ti  ti,F | Параметры функции |
| M | +E  -E  \*E  /E  %E  +EM  -EM  \*EM  /EM  %EM | Выражение |
| Z | i  l  i<l  i>l  l>i  l<i  l<l  l>l | Условие для цикла |
| K | dti;  i=E;  dtfi(F);  dti;N  i=E;N  dtfi(F);N  pE;  pE;N  w(Z){K};  w(Z){K};N  c(i,P,P);  c(i,P,P);N  C(i,P);  C(i,P);N | Операторы цикла |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| P | i  l | Параметры встроенных функций |

## **4.3 Построение конечного магазинного автомата**

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

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ А) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

## **4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающей правила языка MSV-2022. Данные структуры представлены в приложении Д.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата:

1. В магазин записывается стартовый символ;
2. На основе полученных ранее таблиц формируется входная лента;
3. Запускается автомат;
4. Выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;//в обратном ли
5. Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала;
6. Если в магазине встретился нетерминал, переходим к пункту 4;
7. Если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

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

Перечень сообщений синтаксического анализатора представлен в листинге 4.1

|  |
| --- |
| ERROR\_ENTRY(600,"Неверная структура программы"),  ERROR\_ENTRY(601,"Ошибочный оператор"),  ERROR\_ENTRY(602,"Ошибка в выражении"),  ERROR\_ENTRY(603,"Ошибка при указании подвыражения"),  ERROR\_ENTRY(604,"Ошибка в параметрах вызываемой функции"),  ERROR\_ENTRY(605,"Ошибка в условии цикла"),  ERROR\_ENTRY(606,"Ошибка в выражении в цикле"),  ERROR\_ENTRY(607,"Ошибка. ожидался идентификатор или литерал") |

Листинг 4.1 – Перечень сообщений синтаксического анализатора

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

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

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

## **4.8 Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

1. Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
2. Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка.
3. Соответствующее сообщение выводится в консоль
4. В случае нахождения ошибки, после всей процедуры трассировки в протокол будет выведено диагностическое сообщение.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода предоставлен в приложении Е в виде фрагмента трассировки и дерева разбора исходного кода.

# **Глава 5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

## Семантический анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Некоторые проверки (такие как проверка на единственность точки входа, проверка на предварительное объявление переменной) осуществляются в процессе лексического анализа. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1 – Структура семантического анализатора

## **5.2 Функции семантического анализатора**

## Семантический анализатор проверяет правильность составления программных конструкций. При невозможности подобрать правило перехода будет выведен код ошибки, а так же код этой ошибки. Информация об ошибках выводится в протокол работы.

## **5.3 Структура и перечень сообщений семантического анализатора**

Сообщения, формируемые семантическим анализатором, представлены в листинге 5.1.поменять

|  |
| --- |
| ERROR\_ENTRY(308, "[SEM]# Кол-во ожидаемых функцией и передаваемых параметров не совпадают"),  ERROR\_ENTRY(309, "[SEM]# Несовпадение типов передаваемых параметров"),  ERROR\_ENTRY(314, "[SEM]# Типы данных в выражении не совпадают"),  ERROR\_ENTRY(315, "[SEM]# Тип функции и возвращаемого значения не совпадают"),  ERROR\_ENTRY(316, "[SEM]# Попытка вернуть значениее другого типа"),  ERROR\_ENTRY(317, "[SEM]# Неверное условное выражение"),  ERROR\_ENTRY(318, "[SEM]# Деление на ноль"),  ERROR\_ENTRY(320,"[SEM]# Попытка произвести арифметическую операцию со строкой"),  ERROR\_ENTRY(321,"[SEM]# Неверно указаны параметры для concatstr"),  ERROR\_ENTRY(322,"[SEM]# Неверно указаны параметры для copystr"), |

Листинг 5.1 – Перечень сообщений семантического анализатора

## **5.4 Принцип обработки ошибок**

## Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением. Анализ останавливается после того,как встретит ошибку.

## **5.5 Контрольный пример**

Соответствие примеров некоторых ошибок в исходном коде и 28 диагностических сообщений об ошибках приведено в таблице 5.1.

Таблица 5.1 – Тестирование функций

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| vacuum function name(integer a)  {  return a;  } | Ошибка 306: Попытка вернуть значение в процедуре |
| main  {  let integer i = 5;  let string str = i;  print str;  return 0;  }; | Ошибка 312: Типы данных в выражении не совпадают  Строка 4 позиция -1 |
| main  {  let integer a = 5;  let integer b = i/0;  print b;  return 0;  }; | Ошибка 318: Деление на ноль  Строка 5 позиция -1 |

# **Глава 6. Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке MSV-2022 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, /, % (остаток от деления),и (), а также вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке MSV-2022

|  |  |
| --- | --- |
| Приоритет | Операция |
| 0 | ( |
| 1 | , |
| 2 | + - |
| 3 | \* / % |
| 4 | ) |

## **6.2 Польская запись**

Выражения в языке MSV-2022 преобразовываются к обратной польской записи.

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

Обратная польская запись — это форма записи математических выражений, в которой операции расположены после своих операндов. Выражение в обратной польской нотации читается слева направо: операция выполняется над двумя операндами, непосредственно стоящими перед знаком этой операции.

Алгоритм построения:

– исходная строка: выражение;

– результирующая строка: польская запись;

– стек: пустой;

– исходная строка просматривается слева направо;

– операнды переносятся в результирующую строку в порядке их следования;

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

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

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

– отрывающая скобка помещается в стек;

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

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

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

–символы, исключенные из строки записываются в конец строки как “&”

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |  |
| --- | --- | --- | --- |
| Символ | Операция | Стек | Выходная строка |
| i | добавить к выходной строке |  | i |
| + | поместить в стек | + | i |
| i | добавить к выходной строке | + | i, i |
| \* | поместить в стек | +, \* | i, i |
| ( | поместить в стек | +, \*, ( | i, i |
| l | добавить к выходной строке | +, \*, ( | i, i, l |
| + | поместить в стек | +, \*, (, + | i, i, l |
| l | добавить к выходной строке | +, \*, (, + | i, i, l, l |
| ) | присоединить содержимое стека до скобки в обратном порядке к выходной строке и удалить скобку из стека. | +, \* | i, i, l, l, + |

## **6.3 Программная реализация обработки выражений**

Программная реализация функции перевода в обратную польскую инверсию содержится в цикле, который проходит по таблице лексем. При нахождении символа присваивания (=) вызывает функцию polishNotation и преобразует последующее выражение до конца строки.

## **6.4 Контрольный пример**

Пример преобразования выражения к польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления.

# **Глава 7. Генерация кода**

## **7.1 Структура генератора кода**

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

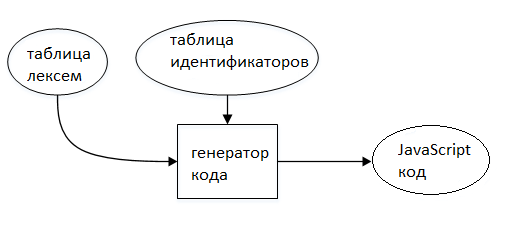


Рисунок 7.1 Структура генератора кода

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

интерпретатор JavaScript динамически выделяет необходимую память при объявлении переменных и функций

Таблица 7.1 – Соответствия типов идентификаторов языка MSV-2022 и JavaScript

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке MSV-2022 | Тип идентификатора на JavaScript | Пояснение |
| int | number | числа в JavaScript хранятся в 64-битном формате IEEE-754 |
| string | string | Строки хранятся в формате UTF-16 |

## **7.3 Статическая библиотека**

Статическая библиотека отсутствует и встроенные функции хранятся в списке лексем . Во время работы лексического анализатора в таблицу лексем помещается соответствующая лексема , при генерации кода MSV-2022 в JavaScript код данная лексема преобразовывается в функцию.

## **7.4 Особенности алгоритма генерации кода**

В процессе генерации программа посимвольно проходит по таблице лексем и в зависимости от лексемы генерируется конструкция на языке JavaScript , при встрече символа, отвечающего за литерал или функцию, программа берет значение из таблицы идентификаторов.

## **7.5 Особенности алгоритма генерации кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке MSV-2022. Результатом работы генератора является JavaScript код.

## **7.6 Контрольный пример**

Результат генерации JavaScript кода на основе контрольного примера из

приложения А приведен в приложении . Результат работы контрольного примера

приведён в приложении .

# **Глава 8. Тестирование транслятора**

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

В языке программирования MSV-2022 не разрешается использовать запрещенные входным алфавитом символы. Результат использования запрещенного символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  declare int a Ä;    } | Ошибка 123: [LEX]#: нераспознанная лексема Строка 1 позиция 14 |

## **8.2 Тестирование лексического анализатора**

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

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  declare int a;  appp = 1+1\*(1 + 1011);;;  } | Ошибка 126: [LEX]#: неизвестная переменная  Строка: 4  Столбец: -1 |

## **8.3 Тестирование синтаксического анализатора**

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

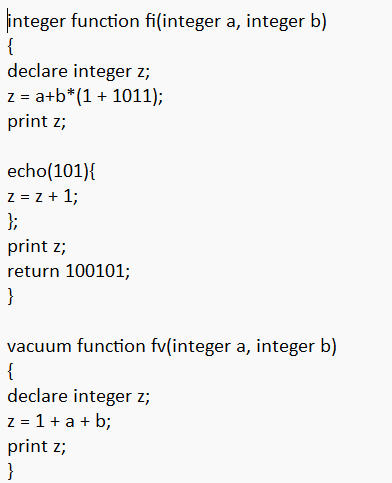
Таблица 8.3 – Тестирование синтаксического анализатора

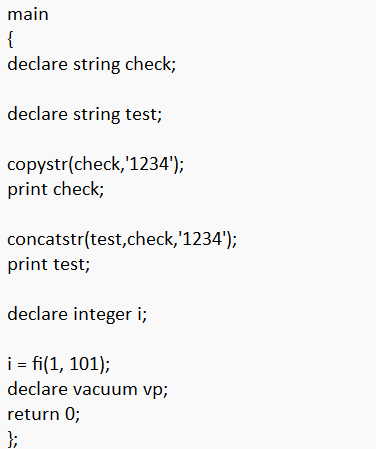
|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  declare int a;  a = 1+1\*(1 + 1011);;;  } | -662: строка 6,Отсутствие подходящей цепочки |

## **8.4 Тестирование семантического анализатора**

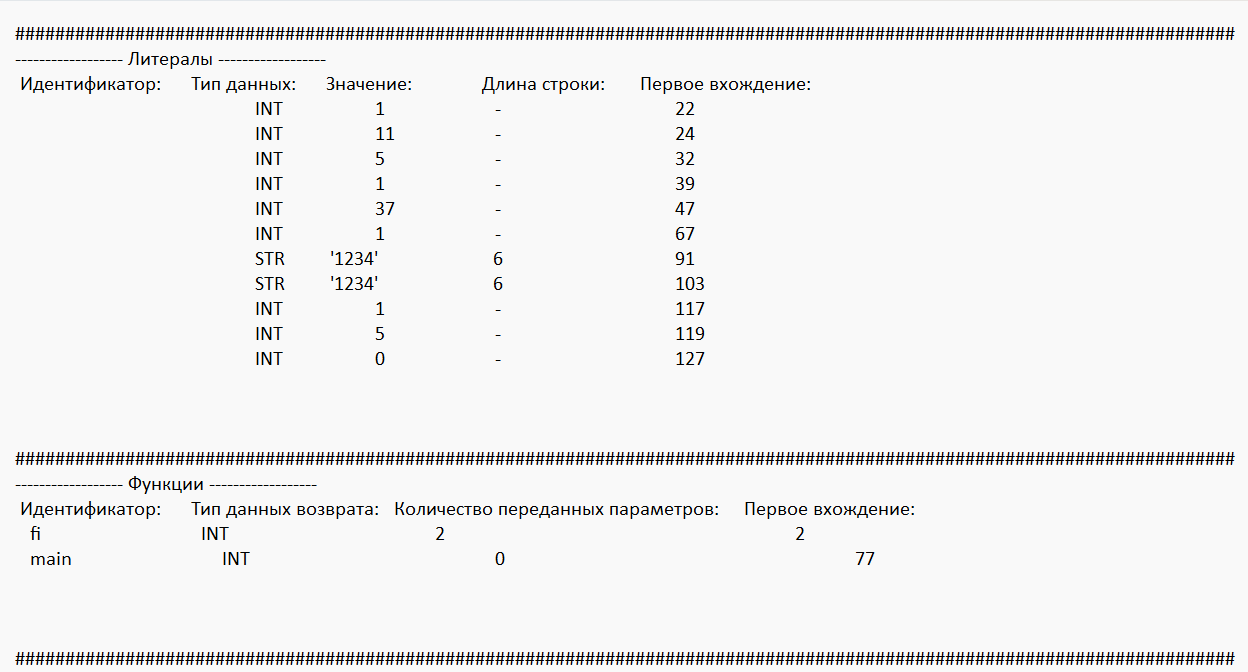
Итоги тестирования семантического анализатора приведены в пункте 5.5.

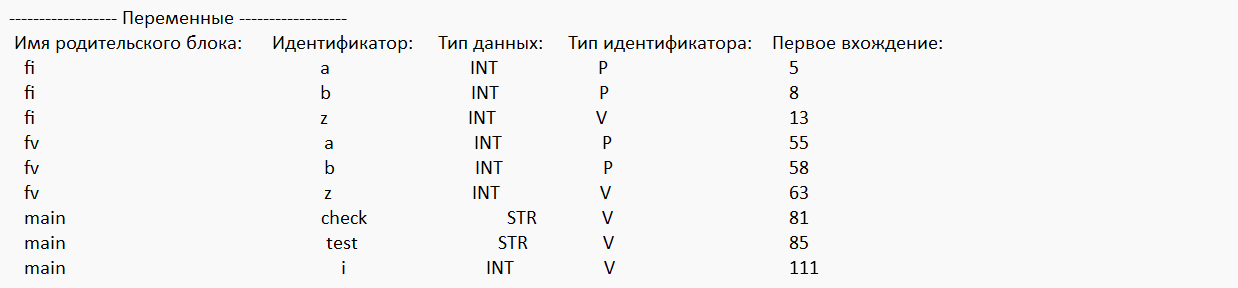
# **Приложение А**

****

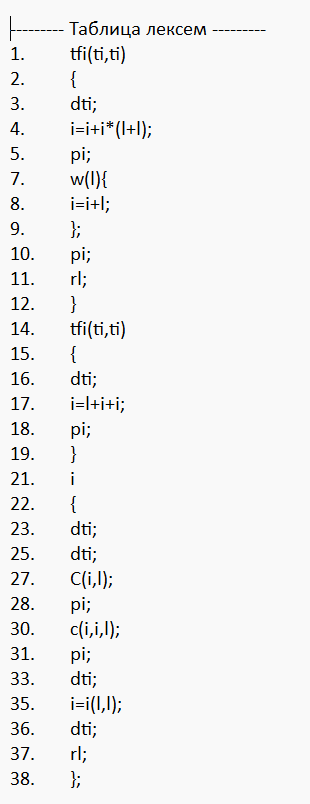
****

# **Приложение Б**

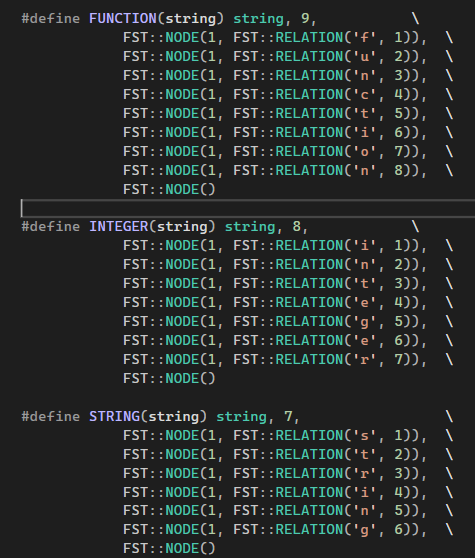
****

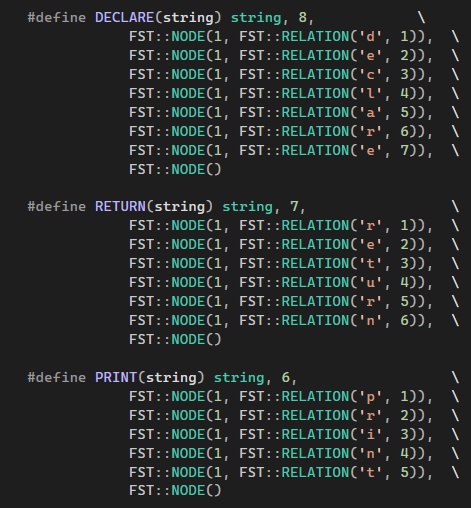
****

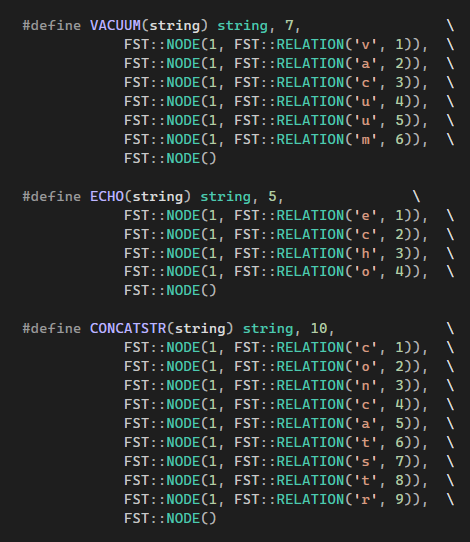
Начало таблицы лексем:

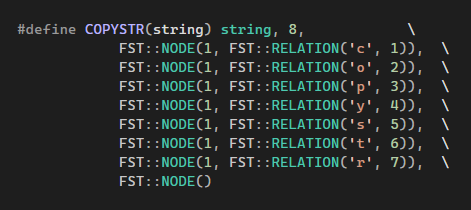
****

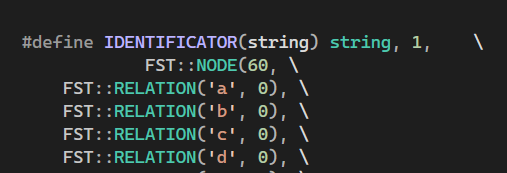
# **Приложение В**

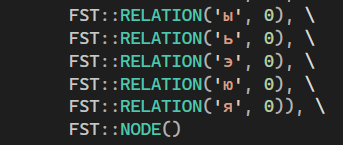


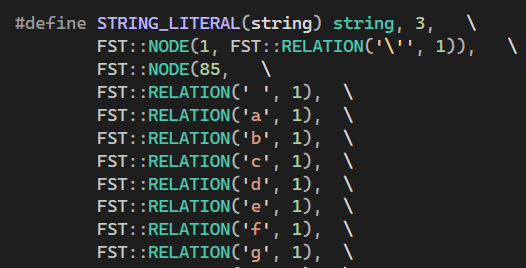


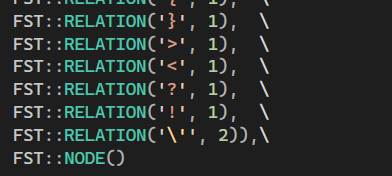


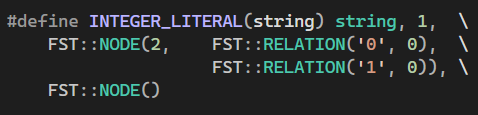




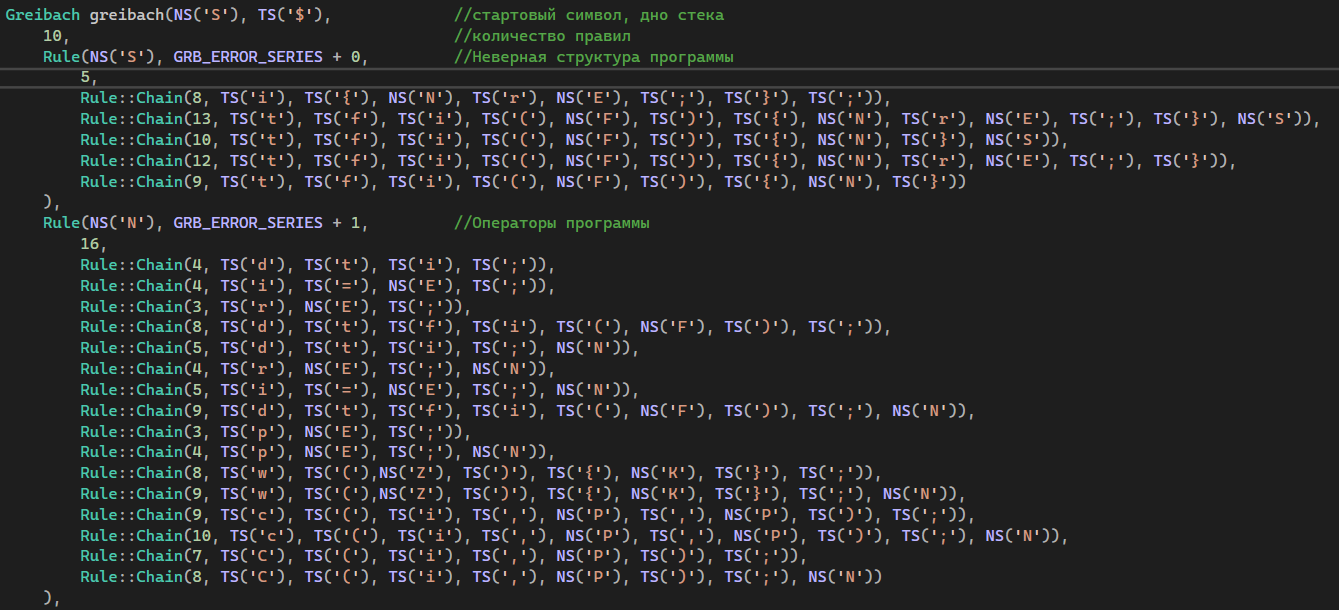


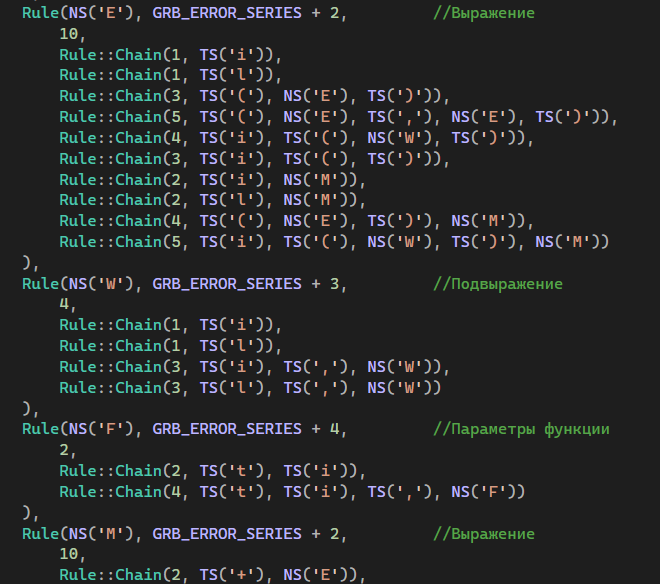


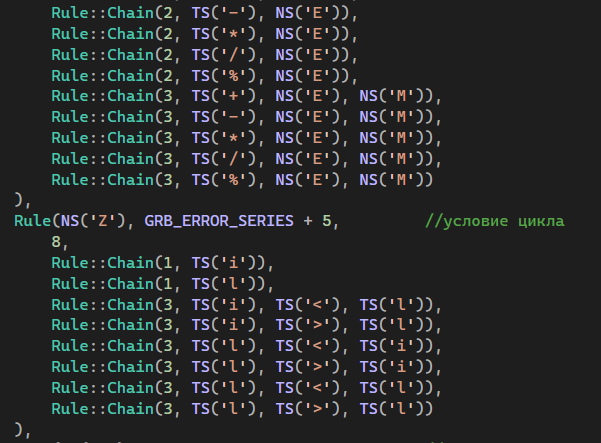


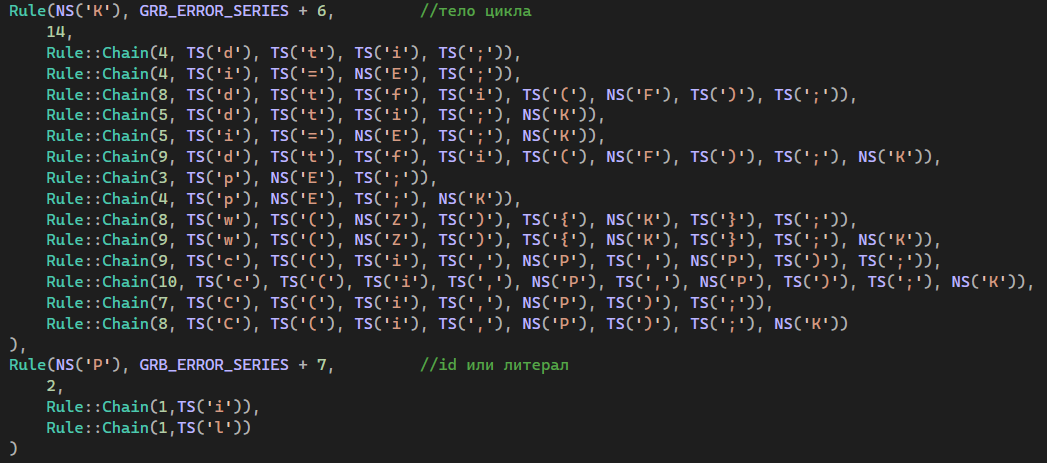


# **Приложение Г**

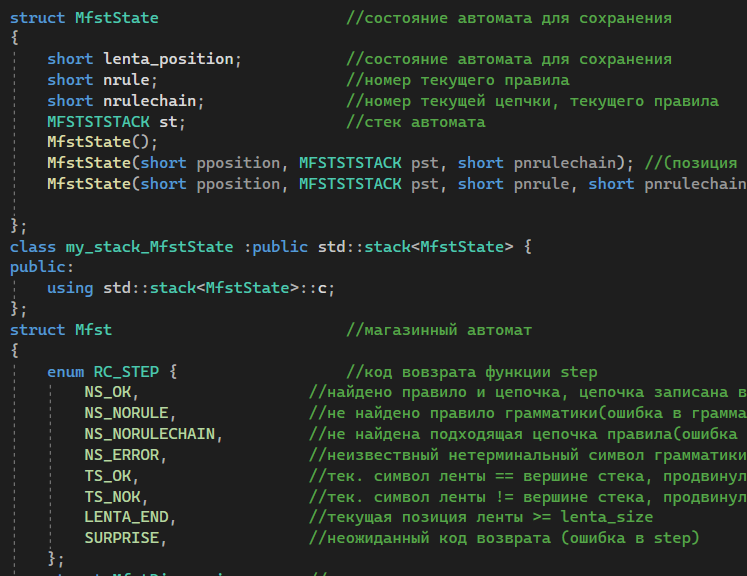


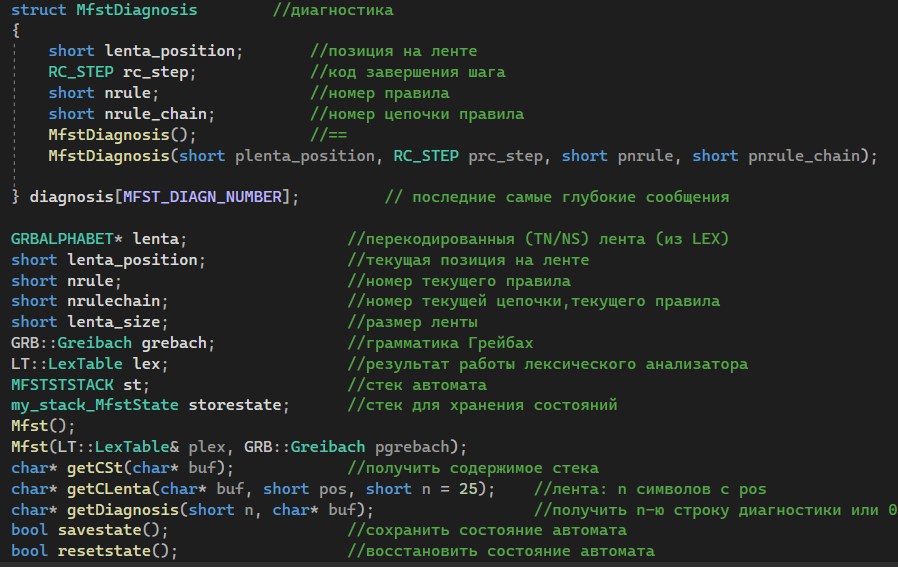


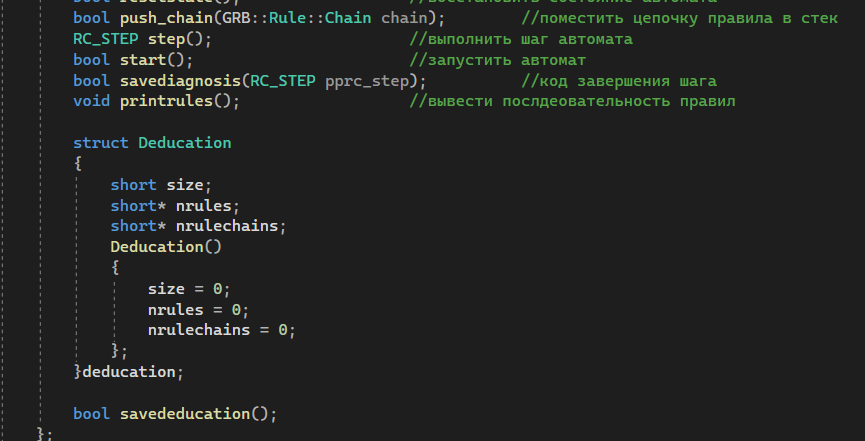




# **Приложение Д**

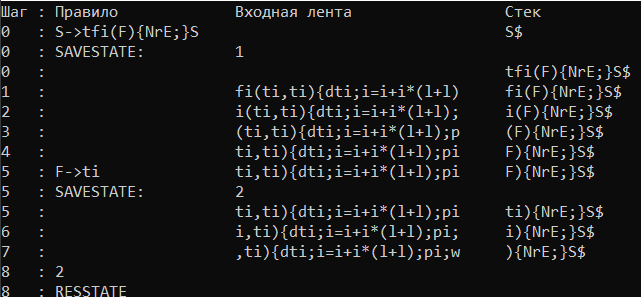




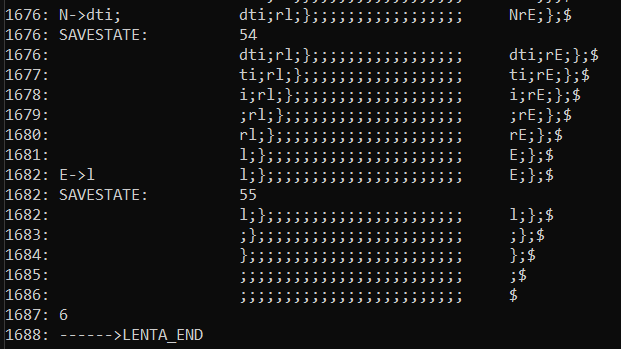


# **Приложение Е**

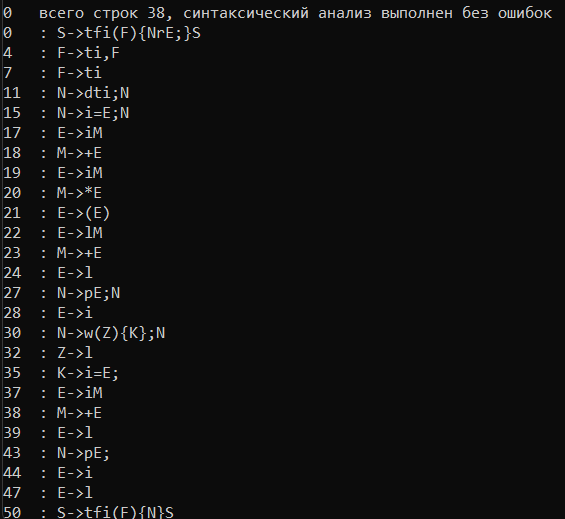
Начало разбора

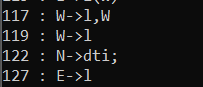
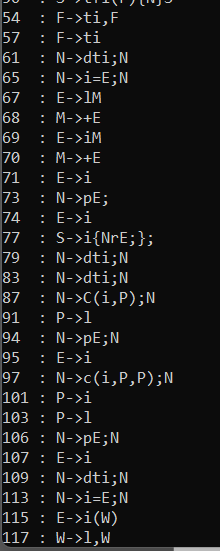


Конец разбора

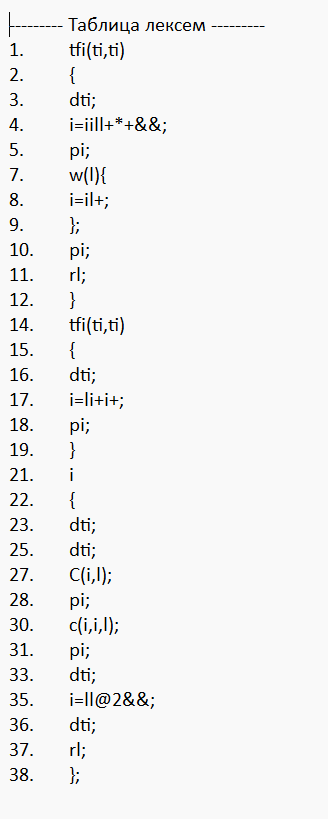


Дерево разбора





# **Приложение Ж**



# **Литература**

1. Ахо А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
2. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.
3. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.