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 MAD-2021»

Выполнил студент Мозолевский Александр Дмитриевич

(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 5](#_Toc91026521)

[Глава 1. Спецификация языка программирования 6](#_Toc91026522)

[1.1 Характеристика языка программирования 6](#_Toc91026523)

[1.2 Алфавит языка 6](#_Toc91026524)

[1.3 Символы сепараторы 6](#_Toc91026525)

[1.4 Применяемые кодировки 6](#_Toc91026526)

[1.5 Типы данных 7](#_Toc91026527)

[1.6 Преобразование типов данных 8](#_Toc91026528)

[1.7 Идентификаторы 8](#_Toc91026529)

[1.8 Литералы 8](#_Toc91026530)

[1.9 Область видимости идентификаторов 8](#_Toc91026531)

[1.10 Инициализация данных 9](#_Toc91026532)

[1.11 Инструкции языка 9](#_Toc91026533)

[1.12 Операции языка 10](#_Toc91026534)

[1.13 Выражения и их вычисления 10](#_Toc91026535)

[1.14 Программные конструкции языка 10](#_Toc91026536)

[1.15 Область видимости 11](#_Toc91026537)

[1.16 Семантические проверки 11](#_Toc91026538)

[1.17 Распределение оперативной памяти на этапе выполнения 11](#_Toc91026539)

[1.18 Стандартная библиотека и её состав 11](#_Toc91026540)

[1.19 Ввод и вывод данных 11](#_Toc91026541)

[1.20 Точка входа 11](#_Toc91026542)

[1.21 Препроцессор 12](#_Toc91026543)

[1.22 Соглашения о вызовах 12](#_Toc91026544)

[1.23 Объектный код 12](#_Toc91026545)

[1.24 Классификация сообщений транслятора 12](#_Toc91026546)

[1.25 Контрольный пример 12](#_Toc91026547)

[Глава 2. Структура транслятора 13](#_Toc91026548)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 13](#_Toc91026549)

[2.2 Перечень входных параметров 14](#_Toc91026550)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 14](#_Toc91026551)

[Глава 3. Разработка лексического анализатора 15](#_Toc91026552)

[3.1 Структура лексического анализатора 15](#_Toc91026553)

[3.2 Контроль входных символов 15](#_Toc91026554)

[3.3 Удаление избыточных символов 16](#_Toc91026555)

[3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов 16](#_Toc91026556)

[3.5 Основные структуры данных 17](#_Toc91026557)

[3.6 Принцип обработки ошибок 18](#_Toc91026558)

[3.7 Структура и перечень сообщений лексического анализатора 18](#_Toc91026559)

[3.8 Параметры лексического анализатора и режимы его работы 18](#_Toc91026560)

[3.9 Алгоритм лексического анализа 18](#_Toc91026561)

[3.10 Контрольный пример 18](#_Toc91026562)

[Глава 4. Разработка синтаксического анализатора 19](#_Toc91026563)

[4.1 Структура синтаксического анализатора 19](#_Toc91026564)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 19](#_Toc91026565)

[4.3 Построение конечного магазинного автомата 21](#_Toc91026566)

[4.4 Основные структуры данных 22](#_Toc91026567)

[4.5 Описание алгоритма синтаксического разбора 22](#_Toc91026568)

[4.6 Структура и перечень сообщений синтаксического анализатора 22](#_Toc91026569)

[4.7 Параметры синтаксического анализатора и режимы его работы 23](#_Toc91026570)

[4.8 Принцип обработки ошибок 23](#_Toc91026571)

[4.9 Контрольный пример 23](#_Toc91026572)

[Глава 5. Разработка семантического анализатора 24](#_Toc91026573)

[5.1 Структура семантического анализатора 24](#_Toc91026574)

[5.2 Функции семантического анализатора 24](#_Toc91026575)

[5.3 Структура и перечень сообщений семантического анализатора 24](#_Toc91026576)

[5.4 Принцип обработки ошибок 24](#_Toc91026577)

[5.5 Контрольный пример 25](#_Toc91026578)

[Глава 6. Преобразование выражений 26](#_Toc91026579)

[6.1 Выражения, допускаемые языком 26](#_Toc91026580)

[6.2 Польская запись 26](#_Toc91026581)

[6.3 Программная реализация обработки выражений 27](#_Toc91026582)

[6.4 Контрольный пример 27](#_Toc91026583)

[Глава 7. Генерация кода 28](#_Toc91026584)

[7.1 Структура генератора кода 28](#_Toc91026585)

[7.2 Представление типов данных в оперативной памяти 28](#_Toc91026586)

[7.3 Статическая библиотека 28](#_Toc91026587)

[Глава 8. Тестирование транслятора 30](#_Toc91026588)

[8.1 Тестирование фазы проверки на допустимость символов 30](#_Toc91026589)

[8.2 Тестирование лексического анализатора 30](#_Toc91026590)

[8.3 Тестирование синтаксического анализатора 30](#_Toc91026591)

[8.4 Тестирование семантического анализатора 31](#_Toc91026592)

[Cписок использованной литературы 32](#_Toc91026593)

[Приложение А 33](#_Toc91026594)

[Приложение Б 35](#_Toc91026595)

[Приложение В 38](#_Toc91026596)

[Приложение Г 41](#_Toc91026597)

[Приложение Д 43](#_Toc91026598)

[Приложение Е 45](#_Toc91026599)

[Приложение Ж 47](#_Toc91026600)

[Приложение З 48](#_Toc91026601)

# **Введение**

Задачей данного курсового проекта была поставлена разработка транслятора своего языка программирования MAD-2021. Он предназначен для выполнения простейших операций и арифметических действий над строками и числами.

Главной задачей транслятора заключается в том, чтобы сделать исходный код на данном языке программирования понятной компьютеру. Для решения этой задачи был выбран способ трансляции исходного кода моего языка программирования в исходный код на языке ассемблера. Язык ассемблера – это машинно-ориентированный язык, представляющий формат записи машинных команд, которые понятны для восприятия человеком.

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 на язык ассемблера;

– 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 язык ассемблера с помощью таблиц лексем и идентификаторов

В восьмой главе описывается тестирование транслятора

# **Глава 1. Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

Язык программирования MAD-2021 – это строго типизируемый, процедурный язык высокого уровня, который транслируется в язык ассемблера.

## **1.2 Алфавит языка**

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

Язык программирования MAD-2021 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 MAD-2021, 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 |

## **1.4 Применяемые кодировки**

Для написания исходного кода на языке программирования MAD-2021 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.

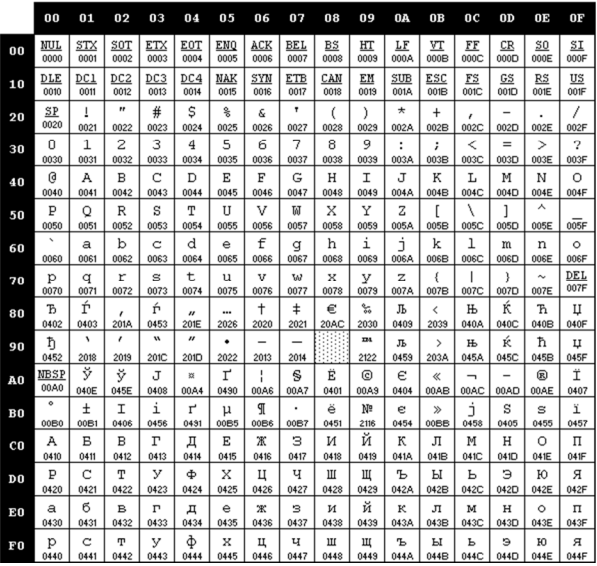


Рисунок 1.1 – Кодировка Windows-1251

## **1.5 Типы данных**

В языке программирования MAD-2021 используются три основных типа данных, которые описываются в таблице 1.2.

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). |
| bool | Фундаментальный тип данных, используемый для объявления логической переменной, которая принимает одно из двух значений: true или false. Без явно указанной инициализации переменной, присваивается нулевое значение (false). |

## **1.6 Преобразование типов данных**

В языке программирования MAD-2021 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. Зарезервированные идентификаторы не поддерживаются. Предусмотрены несколько правил составления идентификатора:

– состоит из символов латинского алфавита любого регистра и цифр;

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

– максимальная длина идентификатора равна 15. При превышении длины идентификатора она будет урезаться;

– 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 MAD-2021 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, только rvalue. Могут быть представлены в десятичной и шестнадцатеричной системах. Для представления в шестнадцатеричной системе необходимо сделать префикс 0х. |
| Строковые литералы | Символы, заключенные в одинарные кавычки (‘’), только rvalue. Максимальное число символов в строковом литерале – 255. |

## **1.9 Область видимости идентификаторов**

Область видимости в языке программирования MAD-2021 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**

Способы инициализации переменных языка программирования MAD-2021 представлены в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| let <тип данных> <идентификатор>; | Автоматическая инициализация переменной. int – инициализируется нулем, string – пустой строкой, bool – false. |
| let <тип данных> <идентификатор> = <выражение>; | Инициализация переменной с присваиванием значения. |

## **1.11 Инструкции языка**

Все возможные инструкции языка программирования MAD-2021 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования MAD-2021

|  |  |
| --- | --- |
| Инструкция | Запись |
| Объявление переменной | let <тип данных> <идентификатор>;  let <тип данных> <идентификатор> = <выражение>; |
| Присваивание | <идентификатор> = <выражение >; |
| Объявление функции | <тип данных> function <идентификатор> ([<тип данных> <идентификатор>][, <тип данных> <идентификатор>]\*) {…} |
| Блок инструкций | {  …  } |
| Возврат из подпрограммы | ret <выражение>; |
| Вывод данных | write <выражение>;  writeln <выражение>; |
| Оператор цикла | repeat (<условие>)  {…} |
| Условный оператор | where (<условие>)  {…} |
| Условный оператор с блоком else | where (<условие>)  {…}  else  {…} |

## **1.12 Операции языка**

Язык программирования MAD-2021 может выполнять арифметические и логические операции, представленные в таблице 1.6.

Таблица 1.6 – 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 MAD-2021 представлены в таблице 1.7

Таблица 1.7 – программные конструкции

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

## **1.15 Область видимости**

В языке программирования MAD-2021 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 1.8.

Таблица 1.8 – 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 стеке. 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**

Функции стандартной библиотеки с их описанием приведены в таблице 1.9. Стандартная библиотека реализована на языке C++.

## **1.19 Ввод и вывод данных**

В языке программирования MAD-2021 ввод данных не поддерживается. Вывод данных происходит с помощью функции write(<выражение>); или writeln(<выражение>);

## **1.20 Точка входа**

Точкой входа в языке программирования MAD-2021 является функция main.

## **1.21 Препроцессор**

В языке программирования MAD-2021 имеется препроцессор. Директива $include[<имя библиотеки>] подключает выбранную библиотеку языка MAD-2021.

## **1.22 Соглашения о вызовах**

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

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## **1.23 Объектный код**

Язык программирования MAD-2021 транслируется в язык ассемблера.

## **1.24 Классификация сообщений транслятора**

В случае возникновения ошибки в исходном коде программы на языке программирования MAD-2021 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 |
| 200-299 | Ошибки лексического анализа |
| 600-699 | Ошибки синтаксического анализа |
| 300-399 | Ошибки семантического 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 MAD-2021 в программу на языке ассемблера. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на языке ассемблера. 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 MAD-2021, в котором сепараторами были разделены слова. 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 ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует кода на языке программирования MAD-2021 в код на языке Ассемблера.

## **2.2 Перечень входных параметров**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 Входные параметры транслятора языка MAD-2021

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя файла> | Входной файл с расширением .txt, в котором содержится исходный код на языке MAD-2021 | Не предусмотрено |
| -log:<имя файла> | Файл для записи протокола работы транслятора | <имя файла>.log |
| -out:<имя файла> | Файл для записи результата работы транслятора | <имя файла>.out.asm |
| -tokens | Ключ для вывода промежуточного представления кода | Отсутствует |
| -lex | Ключ для вывода таблицы лексем в консоль | Отсутствует |
| -id | Ключ для вывода таблицы идентификаторов в консоль | Отсутствует |
| -rules | Ключ для вывода трассировки синтаксического анализатора в консоль | Отсутствует |

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

Перечень протоколов, формируемых транслятором языка программирования MAD-2021 и их назначением представлен в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка MAD-2021

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала с параметром <log> | Содержит информацию о времени выполнения приложения; входных параметрах в приложение; код на языке программирования MAD-2021 с сепараторами и без избыточных пробелов, табуляций и переходов на новую строку; таблицы лексем и идентификаторов; 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> | Содержит сгенерированный код на языке Ассемблера |

# **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 MAD-2021, которые предварительно был обработан, т. е. были удалены пробелы, табуляция и комментарии, а символы перехода строки заменены прямой чертой. 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 MAD-2021

## **3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2.

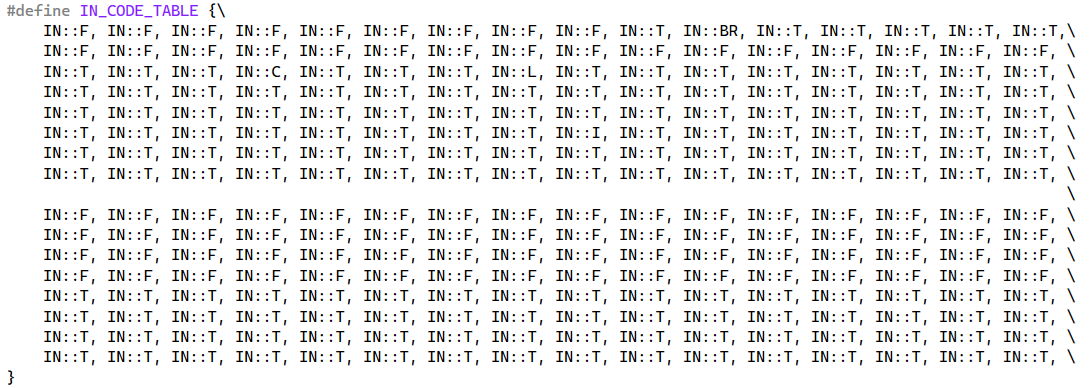


Рисунок 3.2. Таблица контроля входных символов

## **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 или символов после начала комментария вне пределов строкового литерала 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 | let | d |
| int, string, bool | t |
| main | m |
| function | f |
| procedure | p |
| ret | r |
| write | o |
| writeln | b |
| repeat | u |
| where | w |
| else | ! |
| Данные | Лексема | l |
| Идентификатор | i |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| Операции | Арифметические (+,-,/,\*,%) и логические (>, <, ~, !) | v |
| 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 {  int idxFirstLE; // индекс первой строки в ТЛ  char visibility[ID\_MAXSIZE]; // область видимости  char id[ID\_MAXSIZE]; // идентификатор  IDDATATYPE iddatatype; // тип данных  IDTYPE idtype; // тип идентификатора  int countOfPar = 0; // количество параметров  union {  int vint = 0; // целочисленное значение  struct {  int len; // длина строки  char str[TI\_STR\_MAXSIZE - 1]; // символы строки  } vstr; // строковое значение  } value; // значение идентификатора  };  struct IdTable {  int maxsize; // ёмкость ТИ  int size; // текущий размер ТИ  Entry\* table; // массив строк ТИ  }; |

Листинг 3.1. — Структура таблицы идентификаторов

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

|  |
| --- |
| struct Entry {  char lexema; // лексема  int line; // строка лесемы  int idxTI; // индекс в ТИ  int priority; // приоритет операции  operations op; // операция  };  struct LexTable {  int maxsize; // ёмкость ТЛ  int size; // текущий размер ТЛ  Entry\* table; // массив строк ТЛ  }; |

Листинг 3.2. — Структура таблицы лексем

## **3.6 Принцип обработки ошибок**

При нахождении ошибок, транслятор пытается продолжить свою работу до завершения текущего этапа, чтобы вывести набор ошибок. При нахождении критической ошибки работа транслятора прекращается.

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

Перечень сообщений лексического анализатора представлен в листинге 3.3.

|  |
| --- |
| ERROR\_ENTRY(200, "[ LEXICAL ] Недопустимый символ в исходном файле (-in)"),  ERROR\_ENTRY(201, "[ LEXICAL ] Превышен размер таблицы лексем"),  ERROR\_ENTRY(202, "[ LEXICAL ] Переполнение таблицы лексем"),  ERROR\_ENTRY(203, "[ LEXICAL ] Превышен размер таблицы идентификаторов"),  ERROR\_ENTRY(204, "[ LEXICAL ] Переполнение таблицы идентификаторов"),  ERROR\_ENTRY(205, "[ LEXICAL ] Неизвестная последовательность символов"), |

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 MAD-2021, а также файл протокола.

## **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.

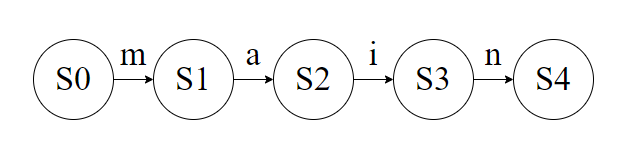


Рисунок 3.3 — Граф переходов для цепочки “main”

## **3.10 Контрольный пример**

Результат работы лексического анализатора – вывод в протокол таблицы лексем и идентификаторов – 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 MAD-2021 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 MAD-2021 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 MAD-2021

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | tfiFBS  piFUS  m{N}  tfiFB  piFU | Проверка правильности структуры программы |
| F | (P)  () | Проверка наличия параметров функции |
| P | Ti  ti,P | Проверка на правильность параметров функции при её объявлении |
| B | {NrI;}  {rE;}  {N} | Проверка наличия тела функции |
| I | i  l | Проверка на недопустимое выражение |
| N | dti;N  dti=E;N  i=E;N  u(R){X}N  w(R){X}N  w(R){X}!{X}N  oI;N  bI;N  rE;N  iK;N  dti;  dti=E;  i=E;  u(R){X}  w(R){X}  w(R){X}!{X}  oE;  bE;  rE;  iK; | Проверка на правильность конструкции в теле функции |
| R | i  ivi  ivl  lvi | Проверка на правильность в условном выражении |
| K | (W)  () | Проверка на правильность вызова функции |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| E | i  l  (E)  iK  iM  lM  (E)M  iKM | Проверка на правильность арифметического выражения |
| W | i  l  i,W  l,W | Проверка на правильность параметров вызываемой функции |
| M | vE  vEM | Проверка на правильность арифметических действий |
| X | dti;X  dti=E;X  i=E;X  oI;X  bE;X  rE;X  iK;X  dti;  dti=E;  i=E;  oE;  bE;  rE;  iK; | Проверка на правильность конструкции в теле цикла или условного выражения |

## **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 MAD-2021. 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;
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, "[ SYNTAX ] Неверная структура программы"),  ERROR\_ENTRY(601, "[ SYNTAX ] Отсутствует список параметров функции"),  ERROR\_ENTRY(602, "[ SYNTAX ] Ошибка в параметрах функции"),  ERROR\_ENTRY(603, "[ SYNTAX ] Отсутствует тело функции"),  ERROR\_ENTRY(604, "[ SYNTAX ] Недопустимое выражение"),  ERROR\_ENTRY(605, "[ SYNTAX ] Отсутствует тело условия или цикла"),  ERROR\_ENTRY(606, "[ SYNTAX ] Неверная конструкция в теле функции"),  ERROR\_ENTRY(607, "[ SYNTAX ] Ошибка в условном выражении"),  ERROR\_ENTRY(608, "[ SYNTAX ] Ошибка в вызове функции"),  ERROR\_ENTRY(609, "[ SYNTAX ] Ошибка в арифметическом выражении"),  ERROR\_ENTRY(610, "[ SYNTAX ] Ошибка в списке параметров при вызове функции"),  ERROR\_ENTRY(611, "[ SYNTAX ] Ошибка в арифметическом выражении"),  ERROR\_ENTRY(612, "[ SYNTAX ] Неверная конструкция в теле цикла или условия"), |

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. Все ошибки записываются в общую структуру ошибок.
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 анализ в трансляторе выделен в отдельную фазу, идущую после синтаксического анализа. Он имеет ряд семантических проверок, которые помогают определить логические несоответствия конструкций языка программирования MAD-2021.

## **5.2 Функции семантического анализатора**

За семантический анализ отвечает функция Analyze. На вход она принимает таблицы лексем и идентификаторов, а также поток вывода в протокол.

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

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

Сообщения, формируемые семантическим анализатором, представлены в листинге 5.1.

|  |
| --- |
| ERROR\_ENTRY(300, "[ SEMANTIC ] Имеется не закрытый строковый литерал"),  ERROR\_ENTRY(301, "[ SEMANTIC ] Имеется более одной точки входа в main"),  ERROR\_ENTRY(302, "[ SEMANTIC ] Не имеется точки входа в main"),  ERROR\_ENTRY(303, "[ SEMANTIC ] Превышен размер строкового литерала"),  ERROR\_ENTRY(304, "[ SEMANTIC ] Объявление переменной без ключевого слова let"),  ERROR\_ENTRY(305, "[ SEMANTIC ] Необъявленный идентификатор"),  ERROR\_ENTRY(306, "[ SEMANTIC ] Объявление переменной без указания типа"),  ERROR\_ENTRY(307, "[ SEMANTIC ] Попытка реализовать существующую функцию"),  ERROR\_ENTRY(308, "[ SEMANTIC ] Объявление функции без указания типа"),  ERROR\_ENTRY(309, "[ SEMANTIC ] Несовпадение типов передаваемых параметров функции"),  ERROR\_ENTRY(310, "[ SEMANTIC ] Несоответствие арифметических операторов"),  ERROR\_ENTRY(311, "[ SEMANTIC ] Невозможно деление на ноль"),  ERROR\_ENTRY(312, "[ SEMANTIC ] Несоответствие типов данных"),  ERROR\_ENTRY(313, "[ SEMANTIC ] Несоответствие открытых и закрытых скобок в выражении"),  ERROR\_ENTRY(314, "[ SEMANTIC ] Функция возвращает неверный тип данных"),  ERROR\_ENTRY(315, "[ SEMANTIC ] Несоответствие количества передаваемых параметров функции"),  ERROR\_ENTRY(316, "[ SEMANTIC ] Попытка переопределить переменную"),  ERROR\_ENTRY(317, "[ SEMANTIC ] Недопустимое значение"), |

Листинг 5.1 – Перечень сообщений семантического анализатора

## **5.4 Принцип обработки ошибок**

Принцип обработки ошибок семантического анализатора основан на перебирании лексем. Когда встречается лексема, последующая цепочка которой может содержать семантическую ошибку, начинается обращение к таблице идентификаторов. Например, при нахождении лексемы «=», анализатор проверяет последующую цепочку до конца строки на ряд семантических ошибок.

## **5.5 Контрольный пример**

Результат работы контрольного примера расположен в приложении Б, где показан результат лексического анализатора, т.к. представленные таблицы лексем и идентификаторов проходят лексическую и семантическую проверки одновременно.

Таблица 5.3 – Тестирование функций

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| {  let int i = 5;  let int res = i + 2;  writeln res;  } | Ошибка 302: [ SEMANTIC ] Не имеется точки входа в main |
| main  {  let int i = 5;  let string str = i;  writeln res;  } | Ошибка 312: [ SEMANTIC ] Несоответсвие типов данных  Строка 5 позиция -1 |
| int function i()  {  ret 0;  }  int function i()  {  ret 1;  }  main  {  let int res = i();  } | Ошибка 307: [ SEMANTIC ] Попытка реализовать существующую функцию  Строка 6 позиция -1 |

# **Глава 6. Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке MAD-2021 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, /, % (остаток от деления), логические операции, как >, <, ! (не равно), ~ (равно) и (), а также вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке MAD-2021

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

## **6.2 Польская запись**

Выражения в языке MAD-2021 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.

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| a+b\*func(i) |  |  |
| +b\*func(i) | a |  |
| b\*func(i) | a | + |
| \*func(i) | ab | + |
| func(i) | ab | +\* |
| (i) | ab | +\* |
| i) | ab | +\* |
| ) | abi | +\* |
|  | abi@1\*+ |  |

## **6.3 Программная реализация обработки выражений**

Программная реализация функции перевода в обратную польскую инверсию содержится в функции startPolish, которая принимает параметром таблицу лексем. Она содержит цикл, который при нахождении символа присваивания (=) вызывает функцию polishNotation и преобразует последующее выражение до конца строки.

После завершение функции startPolish происходит синхронизация индексов таблицы идентификаторов с таблицей лексем, так как лексемы меняют свое положение.

## **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.

В приложении Ж приведено измененное представление промежуточного кода, отображающее результаты преобразования выражений в польский формат.

# **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 ассемблерным кодом.



Рисунок 7.1 Структура генератора кода

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

Некоторые элементы таблицы идентификаторов располагаются в сегментах языка ассемблера. Идентификаторы языка программирования MAD-2021 расположены в сегменте данных (.data). Литералы расположены в сегменте констант (.const). Соответствие между типами данных языка программирования MAD-2021 и языка ассемблера приведены в таблице 7.1. Сгенерированный код приведен в приложении З.

Таблица 7.1 – Соответствия типов идентификаторов языка MAD-2021 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке MAD-2021 | Тип идентификатора на языке ассемблера | Пояснение |
| int | SDWORD | Хранит целочисленный тип данных со знаком. |
| string | BYTE | Каждый символ строки типа string хранится в поле размером 1 байт. |

## **7.3 Статическая библиотека**

Статическая библиотека реализована на языке программирования C++. Её реализация находится в проекте StaticLibrary, в свойствах которого был выбран пункт «статическая библиотека .lib».

В языке программирования MAD-2021 функции поделены на различные библиотеки, которые подключаются в проект использую директиву $include[<имя библиотеки>]. Подключение библиотеки в языке ассемблера происходит с помощью директивы includelib на этапе генерации кода. Далее с помощью оператора EXTRN объявляются имена функций из библиотеки. Оператор EXTRN выполняет две функции. Во-первых, он сообщает ассемблеру, что указанное символическое имя является внешним для текущего ассемблирования. Вторая функция оператора EXTRN состоит в том, что он указывает ассемблеру тип соответствующего символического имени. Так как ассемблирование является очень формальной процедурой, то ассемблер должен знать, что представляет из себя каждый символ. Это позволяет ему генерировать правильные команды. Вышеописанное проиллюстрировано на листинге 7.1.

|  |
| --- |
| out << ".586\n";  out << ".model flat, stdcall\n";  out << "includelib libucrt.lib\n";  out << "includelib kernel32.lib\n";  out << "includelib ../Debug/StaticLibrary.lib\n";  out << "ExitProcess PROTO :DWORD\n\n";  while (!libs.empty()) {  if (libs.top() == "math") {  out << "EXTRN mpow: proc\n";  out << "EXTRN mrand: proc\n";  }  else if (libs.top() == "string") {  out << "EXTRN slen: proc\n";  out << "EXTRN scpy: proc\n";  }  libs.pop();  }  out << "EXTRN BREAKL: proc\n";  out << "EXTRN OutputInt: proc\n";  out << "EXTRN OutputStr: proc\n";  out << "EXTRN OutputIntLn: proc\n";  out << "EXTRN OutputStrLn: proc\n";  out << "\n.stack 4096\n\n"; |

Листинг 7.1 Фрагмент функции генерации кода

# **Глава 8. Тестирование транслятора**

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

В языке программирования MAD-2021 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 |
| int function ك() {ret 0;} | Ошибка 200: [ LEXICAL ] Недопустимый символ в исходном файле (-in)  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  {  let int $err$;  } | Ошибка 205: [ LEXICAL ] Неизвестная последовательность символов  Строка 3 позиция 11 |

## **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 |
| int function fi(int a, int b)  {  ret 0;  }  main  {  let int i = fi(1, );  } | Cтрока 8, [ SYNTAX ] Ошибка в списке параметров при вызове функции |

## **8.4 Тестирование семантического анализатора**

На этапе семантического анализа могут возникнуть ошибки, связанные с семантикой языка. Результаты тестирования семантического анализатора представлены в таблице 8.4.

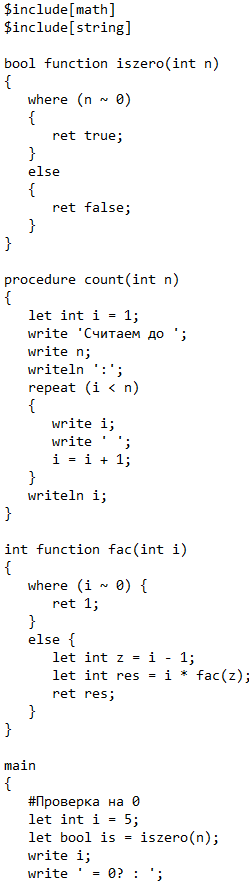
Таблица 8.4 – Тестирование семантического анализатора

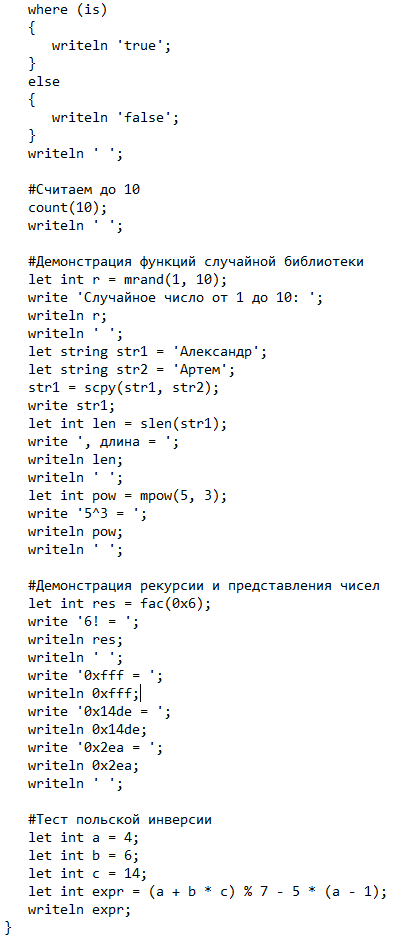
|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  let int i = 1;  let int i = 2;  } | Ошибка 316: [ SEMANTIC ] Попытка переопределить переменную  Строка 3 позиция -1 |
| main  {  let int i = 1;  i = 1 + a;  } | Ошибка 305: [ SEMANTIC ] Необъявленный идентификатор  Строка 4 позиция -1 |
| int function test (int a, string b)  {  ret 0;  }  main  {  test(1, 2);  } | Ошибка 309: [ SEMANTIC ] Несовпадение типов передаваемых параметров функции  Строка 8 позиция -1 |
| int function test (int a, string b)  {  ret 0;  }  main  {  test(1, 'test', '1');  } | Ошибка 315: [ SEMANTIC ] Несоответствие количества передаваемых параметров функции  Строка 8 позиция -1 |

# **Cписок использованной литературы**

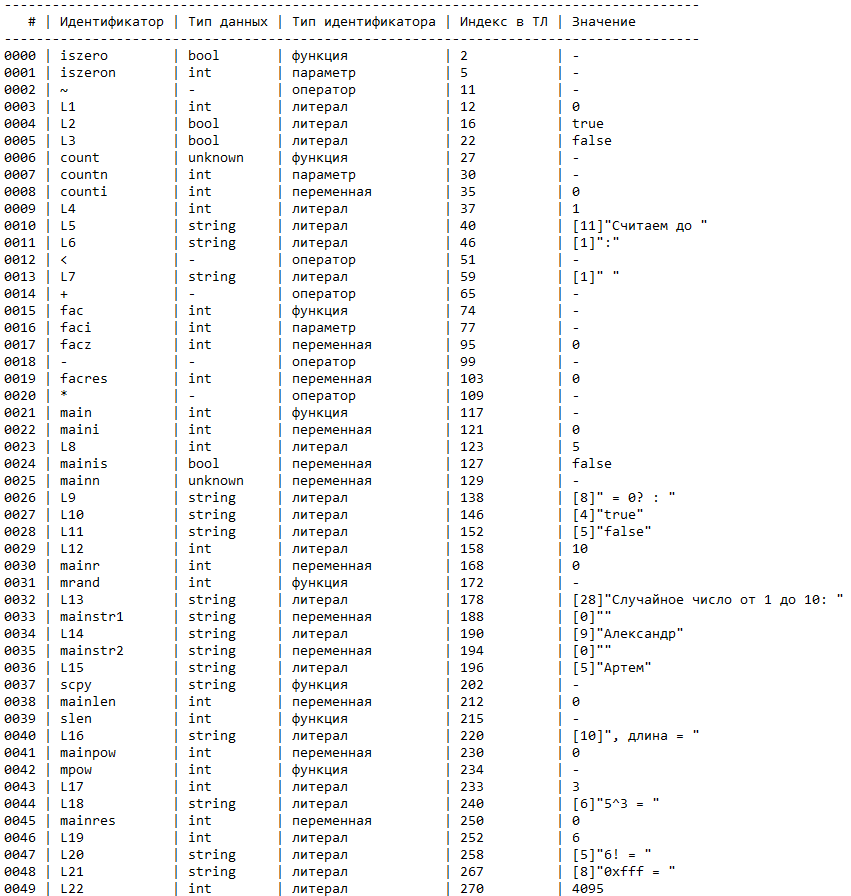
1. Ахо А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
2. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.
3. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.
4. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.

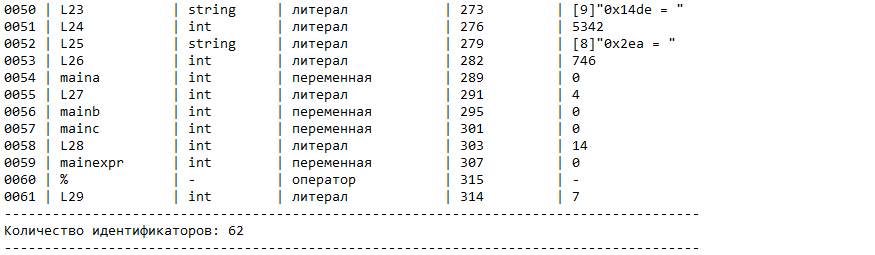
# **Приложение А**

****

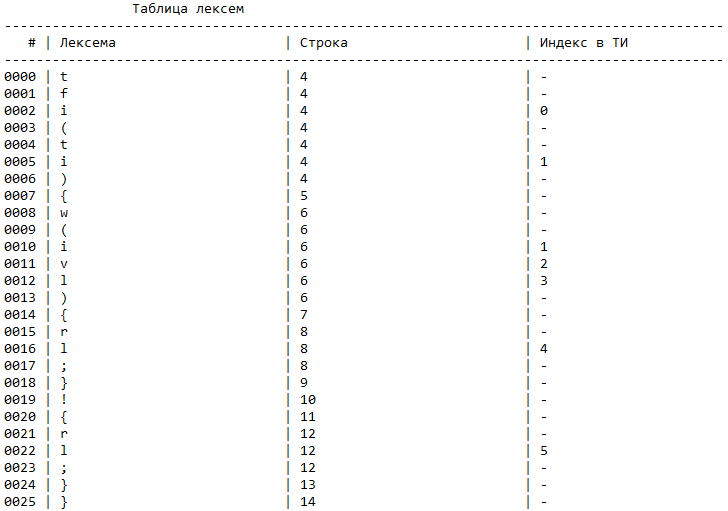
****

# **Приложение Б**

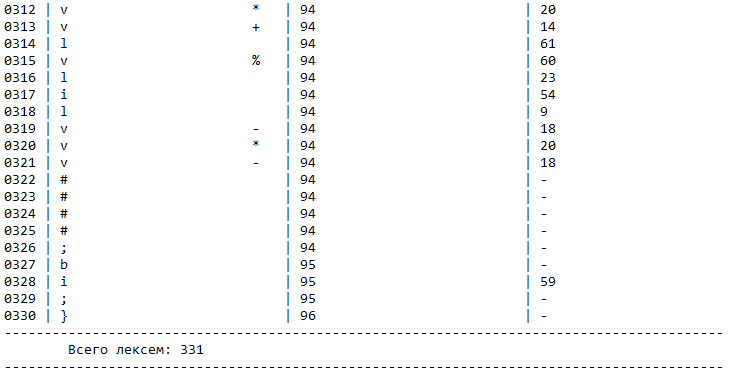
****

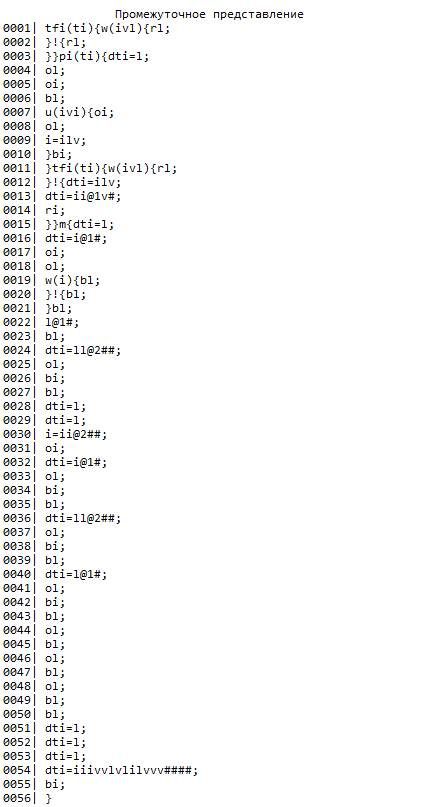
****

Начало таблицы лексем:

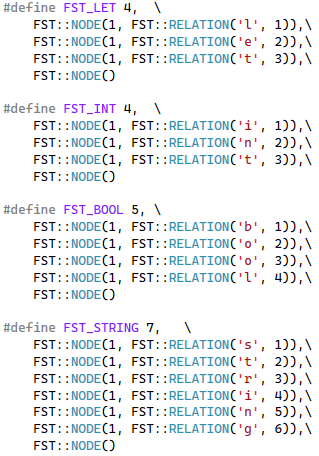
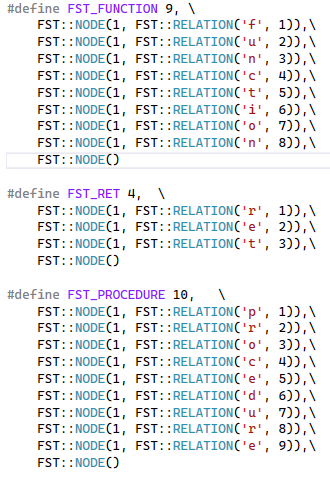
****

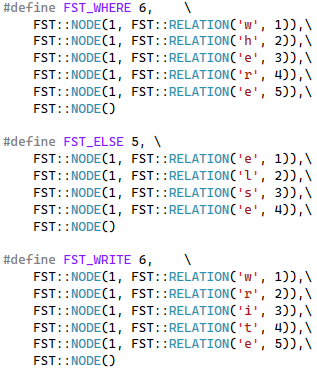
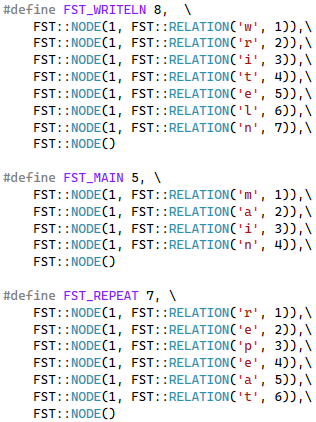
Окончание таблицы лексем:

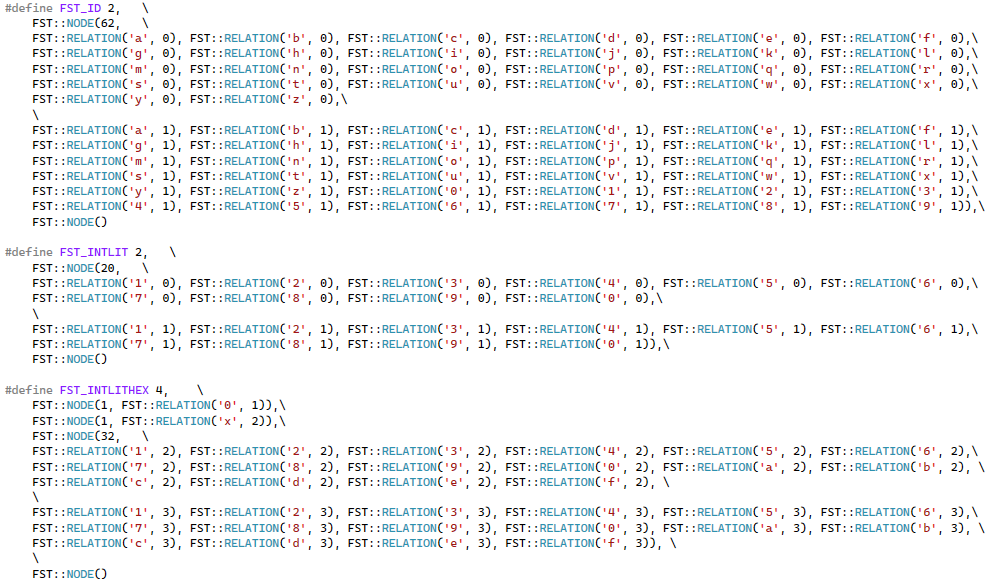


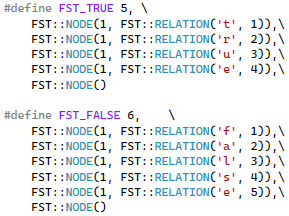
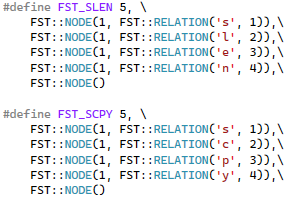


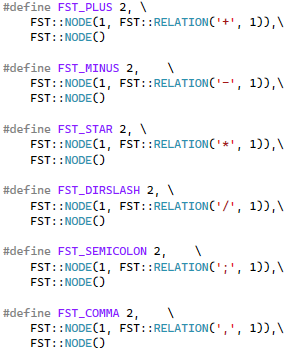
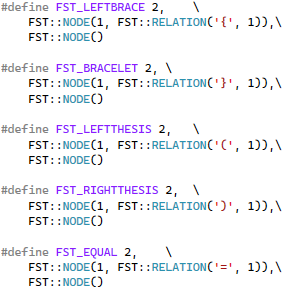
# **Приложение В**

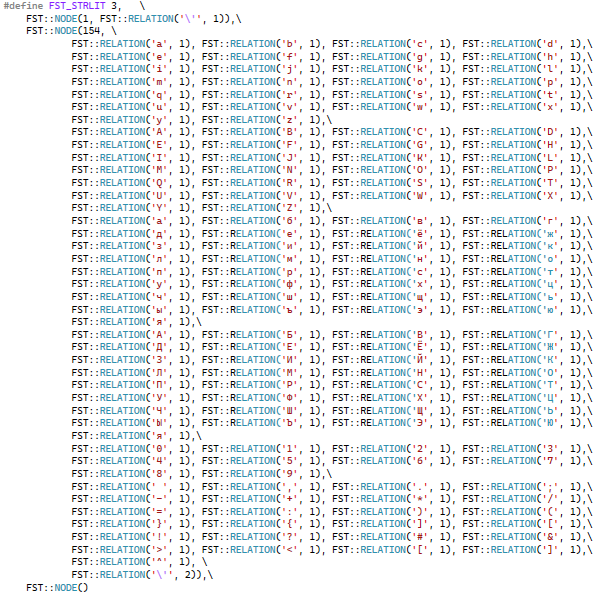
 

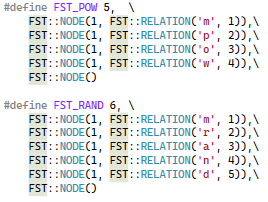


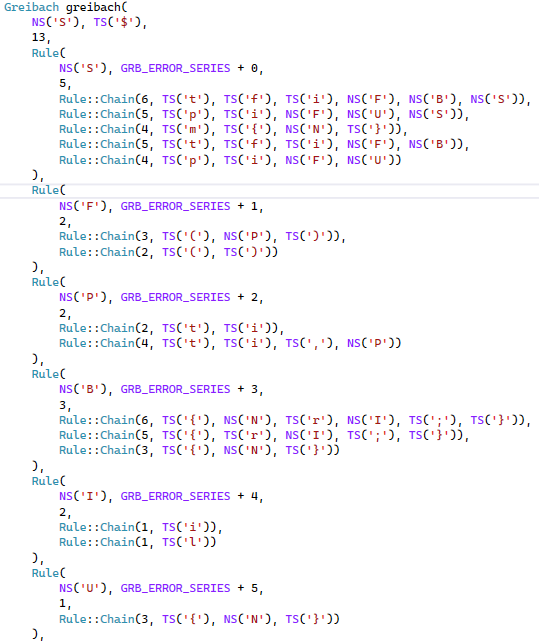
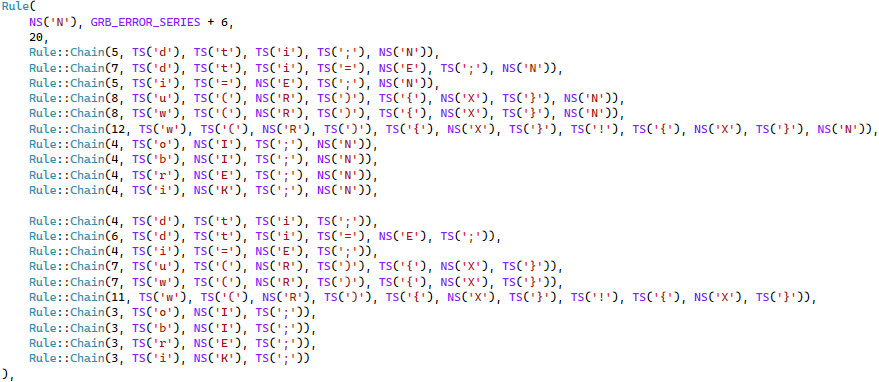
 

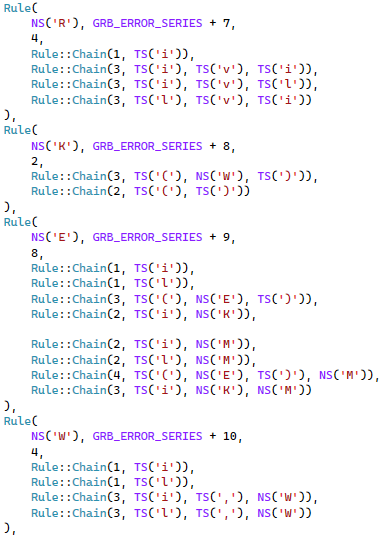
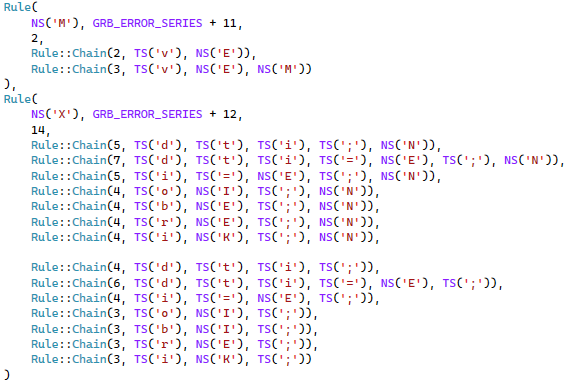




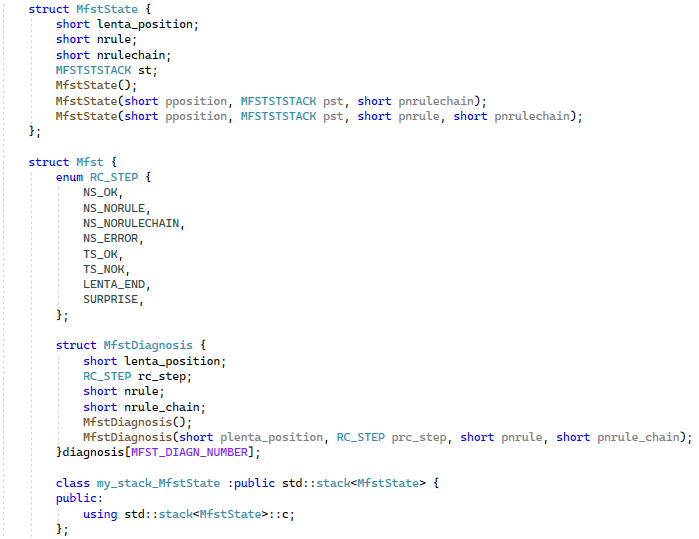
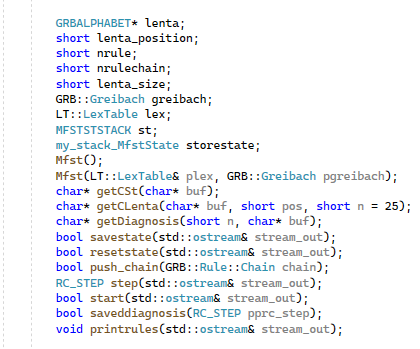


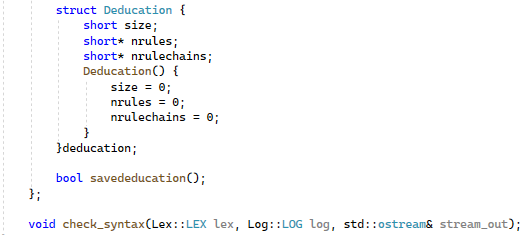
# **Приложение Г**

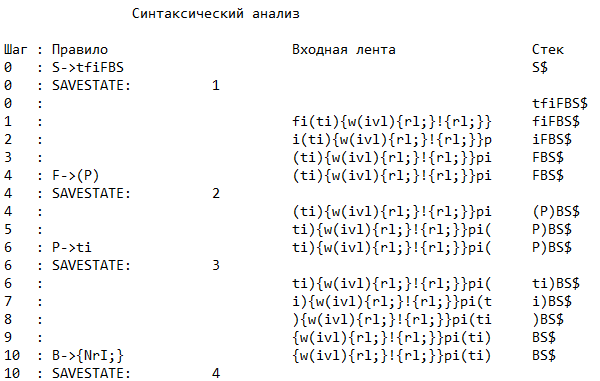
# **Приложение Д**

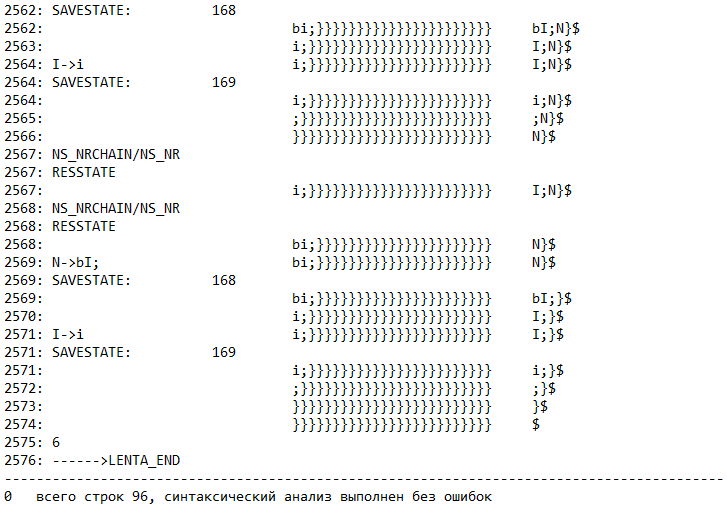


# **Приложение Е**

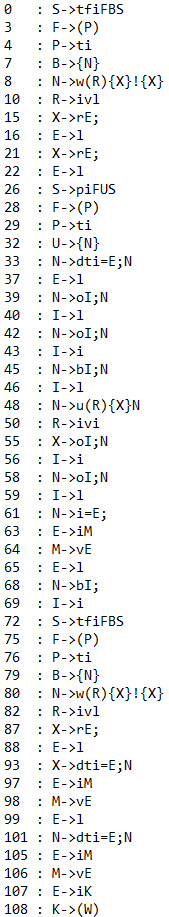
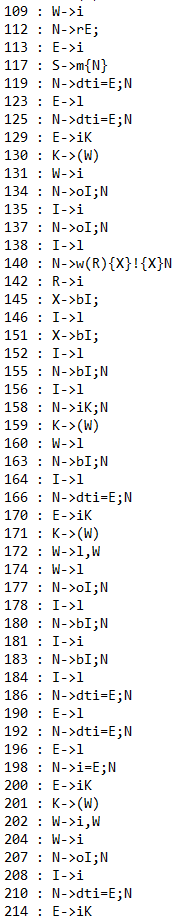
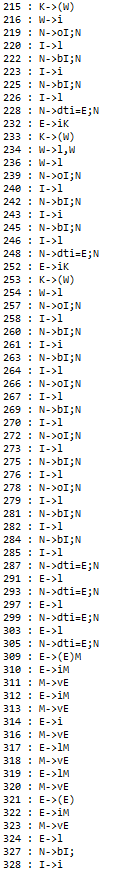
Начало разбора



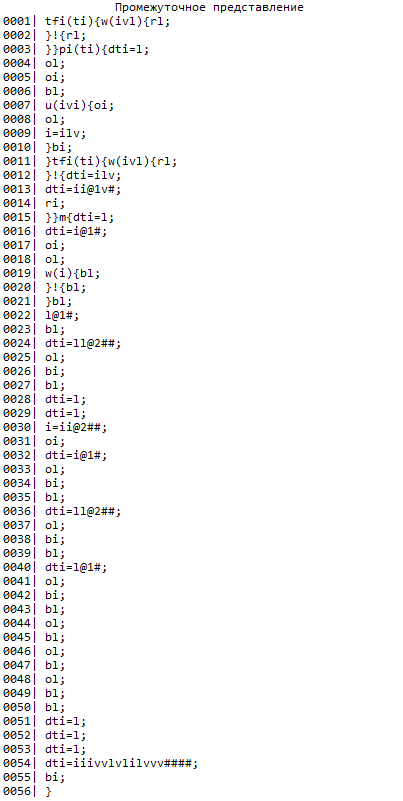
Конец разбора



Дерево разбора

# **Приложение Ж**



# **Приложение З**

