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 RNA-2022»

Выполнил студент Фамилия Имя Отчество

(Ф.И.О.)

Руководитель проекта асс. Мущук Артур Николаевич

(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 асс. Мущук Артур Николаевич

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



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

Нормоконтролер асс. Мущук Артур Николаевич

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

Курсовой проект защищен с оценкой

Минск 2022

**Содержание**

[Введение 5](#_Toc121771121)

[1. Спецификация языка программирования 6](#_Toc121771122)

[1.1 Характеристика языка программирования 6](#_Toc121771123)

[1.2 Определение алфавита языка программирования 6](#_Toc121771125)

[1.4 Применяемые кодировки 6](#_Toc121771126)

[1.5 Типы данных 7](#_Toc121771128)

[1.6 Преобразование типов данных 8](#_Toc121771129)

[1.8 Литералы 9](#_Toc121771130)

[1.9 Объявление данных 10](#_Toc121771131)

[1.10 Инициализация данных 10](#_Toc121771132)

[1.11 Инструкции языка 11](#_Toc121771133)

[1.13 Выражения и их вычисление 12](#_Toc121771134)

[1.14 Конструкции языка 12](#_Toc121771135)

[1.15 Области видимости идентификаторов 13](#_Toc121771136)

[1.16 Семантические проверки 13](#_Toc121771137)

[1.17 Распределение оперативной памяти на этапе выполнения 13](#_Toc121771138)

[1.18 Стандартная библиотека и её состав 14](#_Toc121771139)

[1.19 Ввод и вывод данных 14](#_Toc121771140)

[1.20 Точка входа 15](#_Toc121771141)

[1.21 Препроцессор 15](#_Toc121771142)

[1.22 Соглашения о вызовах 15](#_Toc121771144)

[1.23 Объектный код 15](#_Toc121771145)

[1.24 Классификация сообщений транслятора 15](#_Toc121771146)

[1.25 Контрольный пример 15](#_Toc121771147)

[2. Структура транслятора 16](#_Toc121771148)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 16](#_Toc121771149)

[2.2 Перечень входных параметров транслятора 17](#_Toc121771161)

[2.3 Протоколы, формируемые транслятором 17](#_Toc121771163)

[3. Разработка лексического анализатора 19](#_Toc121771165)

[3.1 Структура лексического анализатора 19](#_Toc121771166)

[3.2 Контроль входных символов 19](#_Toc121771167)

[3.3 Удаление избыточных символов 20](#_Toc121771168)

[3.4 Перечень ключевых слов 20](#_Toc121771169)

[3.5 Основные структуры данных 22](#_Toc121771170)

[3.6 Структура и перечень сообщений лексического анализатора 23](#_Toc121771171)

[3.7 Принцип обработки ошибок 23](#_Toc121771172)

[3.8 Параметры лексического анализатора 23](#_Toc121771173)

[3.9 Алгоритм лексического анализа 23](#_Toc121771174)

[3.10 Контрольный пример 24](#_Toc121771175)

[4. Разработка синтаксического анализатора 25](#_Toc121771176)

[4.1 Структура синтаксического анализатора 25](#_Toc121771177)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 25](#_Toc121771178)

[4.3 Построение конечного магазинного автомата 28](#_Toc121771179)

[4.4 Основные структуры данных 29](#_Toc121771180)

[4.5 Описание алгоритма синтаксического разбора 29](#_Toc121771181)

[4.6 Структура и перечень сообщений синтаксического анализатора 29](#_Toc121771182)

[4.7. Параметры синтаксического анализатора и режимы его работы 29](#_Toc121771183)

[4.8. Принцип обработки ошибок 30](#_Toc121771184)

[4.9. Контрольный пример 30](#_Toc121771185)

[5. Разработка семантического анализатора 31](#_Toc121771186)

[5.1 Структура семантического анализатора 31](#_Toc121771187)

[5.2 Функции семантического анализатора 31](#_Toc121771188)

[5.3 Структура и перечень сообщений семантического анализатора 31](#_Toc121771189)

[5.4 Принцип обработки ошибок 32](#_Toc121771190)

[5.5 Контрольный пример 32](#_Toc121771191)

[6. Вычисление выражений 34](#_Toc121771192)

[6.1 Выражения, допускаемые языком 34](#_Toc121771193)

[6.2 Польская запись и принцип её построения 34](#_Toc121771194)

[6.3 Программная реализация обработки выражений 35](#_Toc121771195)

[6.4 Контрольный пример 36](#_Toc121771196)

[7. Генерация кода 37](#_Toc121771197)

[7.1 Структура генератора кода 37](#_Toc121771198)

[7.2 Представление типов данных в оперативной памяти 37](#_Toc121771199)

[7.3 Статическая библиотека 37](#_Toc121771200)

[7.4 Особенности алгоритма генерации кода 38](#_Toc121771201)

[7.5 Входные параметры генератора кода 39](#_Toc121771202)

[7.6 Контрольный пример 39](#_Toc121771203)

[8. Тестирование транслятора 40](#_Toc121771204)

[8.1 Общие положения 40](#_Toc121771205)

[8.2 Результаты тестирования 40](#_Toc121771206)

[Заключение 42](#_Toc121771207)

[Список использованных литературных источников 45](#_Toc121771208)

[Приложение А 46](#_Toc121771209)

[Приложение Б 48](#_Toc121771210)

[Приложение В 51](#_Toc121771211)

[Приложение Г 53](#_Toc121771212)

[Приложение Д 55](#_Toc121771213)

[Приложение Е 57](#_Toc121771214)

[Приложение Ж 59](#_Toc121771215)

[Приложение З 60](#_Toc121771216)

# **Введение**

Целью курсового проектирования является разработка транслятора для языка программирования RNA-2022. Транслятор реализуется на языке С++, при этом трансляция осуществляется в низкоуровневый язык ассемблера.

Транслятор состоит из следующих частей:

– лексический анализатор;

– семантический анализатор;

– синтаксический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были поставлены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка и преобразование выражений;

– генерация кода на язык ассемблера;

– тестирование транслятора.

**1. Спецификация языка программирования**

* 1. **Характеристика языка программирования**

Язык программирования RNA-2022 является высокоуровневым, процедурным, не объектно-ориентированным, компилируемым. Имеет нестрогую статическую типизацию.

**1.2 Определение алфавита языка программирования**

Алфавит языка RNA-2022 базируется на 8-битной кириллической кодировке Windows-1251. Исходный код RNA-2022 может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9 и некоторые специальные символы для операций и строковых литералов. Кириллические символы разрешены только в строковых литералах и комментариях.

**1.3 Применяемые сепараторы**

Символы-сепараторы служат в качестве разделителей цепочек языка во время обработки исходного текста программы. Используемые в языке RNA-2022 сепараторы представлены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| Символ(ы) | Назначение и область применения |
| ‘пробел’ | Разделитель лексем языка. Допускается везде кроме названий идентификаторов и ключевых слов |
| { … } | Блок функции или условной конструкции/цикла |
| ( … ) | Установка приоритета арифметических операций |
| **,** | Разделитель параметров функций |
| > < == ^=  >= <= | Логические операции (операции сравнения: больше, меньше, равно, не равно, больше или равно, меньше или равно), используемые в условии цикла/условной конструкции, а также в выражениях. |
| + - \* / % | Арифметические операции (сложение, вычитание, умножение, деление, остаток от деления). Используются в выражениях. |
| ; | Разделитель программных конструкций |
| **=** | Оператор присваивания |

**1.4 Применяемые кодировки**

Для написания исходного кода программы на языке RNA-2022 используется кодировка Windows-1251 (представлена на рисунке 1.1).



Рисунок 1.1 – Кодировка Windows-1251

**1.5 Типы данных**

В языке RNA-2022 используются три фундаментальных типа данных: целочисленный, логический и строковый. Описание типов приведено в таблице 1.2.

Таблица 1.2 Типы данных RNA-2022

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| Целочисленный тип данных **number** | Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных. В памяти занимает 2 байта. Максимальное значение: 32767.  Минимальное значение: -32768.  Инициализация по умолчанию: значение 0.  Поддерживаемые операции:  **+** (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  **\*** (бинарный) – оператор умножения;  **/** (бинарный) – оператор целочисленного деления; |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
|  | % (бинарный) – оператор остатка от деления;  **=** (бинарный) – оператор присваивания;  **>** (бинарный) – оператор «больше»;  **<** (бинарный) – оператор «меньше»;  **==**(бинарный) – оператор «равно»;  ^=(бинарный) – оператор «не равно»;  >=(бинарный) – оператор «больше или равно»;  <=(бинарный) – оператор «меньше или равно».  Может использоваться в качестве аргумента в функциях, условия для оператора цикла и условного оператора. |
| Логический тип данных **bool** | Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных. В памяти занимает 1 байт.  Значения: true (истина), false (ложь).  Инициализация по умолчанию: false.  Поддерживаемые операции:  **+** (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  **\*** (бинарный) – оператор умножения;  **/** (бинарный) – оператор целочисленного деления;  % (бинарный) – оператор остатка от деления;  **=** (бинарный) – оператор присваивания. |
| Строковый тип данных **line** | Фундаментальный тип данных. Предусмотрен для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 256.  Инициализация по умолчанию: строка нулевой длины “”. Поддерживаемые операции:  **=** (бинарный) – оператор присваивания. |

**1.6 Преобразование типов данных**

В языке RNA-2022 предусмотрены неявные преобразования между логическими (bool) и целочисленными (number) типами. Значение логического типа true преобразуется к 1, значение false – к 0. К логическому значению true преобразуются все значения типа number кроме 0, он преобразуется к false.

Логический тип преобразуется к целочисленному в следующих случаях:

* при присваивании результата логического выражения переменной типа number;
* при присваивании значения логического идентификатора или литерала переменной типа number;
* при вычислении арифметических выражений;
* при передаче логического типа в качестве целочисленного аргумента функции.

Целочисленный тип преобразуется к логическому в следующих случаях:

* при присваивании результата арифметического выражения переменной типа bool;
* при присваивании значения целочисленного идентификатора или литерала переменной типа bool;
* при передаче целочисленного типа в качестве логического аргумента функции;
* при передаче целочисленного идентификатора или литерала в качестве условия для оператора цикла или условного оператора.

Преобразование строкового типа line в какой-либо другой фундаментальный тип не производится.

**1.7 Идентификаторы**

Идентификаторы в языке RNA-2022 используются для именования функций, процедур, параметров функций и переменных. Идентификаторы, объявленные внутри блока функции или процедуры получают префикс, идентичный имени функции/процедуры для разрешения области видимости. Идентификаторы не должны совпадать с ключевыми словами языка, а также с именами команд ассемблера (за исключением идентификаторов переменных и параметров функций/процедур). Максимальная длина идентификатора не должна превышать 10 символов.

Формальное описание идентификатора приведено на рисунке 1.2.

|  |
| --- |
| *<идентификатор> ::= <буква> | <идентификатор><буква>*  *<буква>::= "A"|"B"|"C"|"D"|"E"|"F"|"G"|"H"|"I"|"J"|"K"|"L"|"M"|"N"|"O"|"P"|*  *"Q"|"R"|"S"|"T"|"U"|"V"|"W"|"X"|"Y"|"Z"|"a"|"b"|"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|*  *"l"|"m"|n"|"o"|"p"|"q"|"r"|"s"|"t"|"u"|"v"|"w"|"x"|"y"|"z"* |

Рисунок 1.2 – Формальное описание идентификатора (БНФ)

Примеры правильных идентификаторов: numA, count, factorial, getName, result, i, isNegative.

Примеры неправильных идентификаторов: num1, number, give, getFourthNumberFibonacci, числоА, number one, result+.

Примеры неправильных идентификаторов для функций или процедур: byte, offset, mov, push.

**1.8 Литералы**

Литералы – это записи в исходном тексте программы, представляющие фиксированное значение.

В языке RNA-2022 существует три типа литералов: целочисленные, логические и строковые. Они используются для инициализации переменных, внутри условия оператора цикла или условного оператора, а также могут выступать в качестве аргумента функции или процедуры. Все литералы являются rvalue. Краткое описание литералов представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Литералы | Описание |
| Целочисленные | Последовательность цифр 0…9 с предшествующим |

Продолжение таблицы 1.3

|  |  |
| --- | --- |
| литералы | знаком минус или без него.  Диапазоны значений: от -32768 до 32767. |
| Логические литералы | Значения true, false. |
| Строковые литералы | Последовательность символов, заключенная в двойные кавычки. Длина может варьироваться в пределах от 1 до 255. Использование двойных кавычек внутри строкового литерала не допускается. |

Примеры правильных литералов: 34, -9, 0, “Hello, World!”, “Привет, Мир!”, true, false.

Примеры неправильных литералов: -9999999, 1002938584, TRUE, “”, “Компания “Рассвет””.

**1.9 Объявление данных**

При объявлении переменной используется ключевое слово **def**, за которым следует указание типа данных и имени идентификатора. Допускается инициализировать данные при объявлении. Для объявления функции используется ключевое слово **function** после которого также следует указание типа и имени идентификатора. При объявлении процедуры используется ключевое слово **procedure** после которого сразу следует имя процедуры без указания типа.

**1.10 Инициализация данных**

Способы инициализации данных представлены в таблице 1.4.

Таблица 1.4 – Способы инициализации данных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| def <тип данных> <идентификатор>; | Инициализация по умолчанию при объявлении: переменные типа bool – инициализируются false, переменные типа number инициализируются нулём, переменные типа line – пустой строкой. |
| def <тип данных> <идентификатор> = <значение>|<выражение>;  <значение>::= <литерал>| <идентификатор>|<вызов функции> | Присваивание значения идентификатора, литерала, результата вычисления выражения или функции при объявлении. |
| <идентификатор> = <значение>|  <выражение>;  <значение>::= <литерал>| <идентификатор>|<вызов функции> | Присваивание значения при повторном использовании. |

**1.11 Инструкции языка**

Инструкции языка программирования RNA-2022 представлены в таблице 1.5.

Таблица 1.5 – Инструкции языка RNA-2022

|  |  |  |
| --- | --- | --- |
| Инструкция | Запись на языке RNA-2022 | Пример |
| Объявление переменной | **def** <тип данных> <идентификатор>; | def number num; |
| Объявление переменной с явной инициализацией | **def** <тип данных> <идентификатор> = <значение>|<выражение>;  <значение>::= <литерал>| <идентификатор>|<вызов функции> | def bool b = true;  def number num = 3 \* 9; |
| Присваивание | <идентификатор> = <значение>|  <выражение>;  <значение>::= <литерал>| <идентификатор>|<вызов функции> | num = 23;  num = result;  num = factorial[9]; |
| Вызов внешней функции или процедуры | <идентификатор> [ <идентификатор>|  <литерал>, …]; | factorial[a];  factorial[9];  max[6, 11]; |
| Возврат из  подпрограммы | **give** <идентификатор> / <литерал>; | give result;  give 6; |
| Вывод данных | **out** [<идентификатор> / <литерал>]; | out[“Hello”]; |
| Вывод данных с переходом на новую строку | **outln** [<идентификатор> / <литерал>]; | outln[result] |
| Условный оператор | **when** [<условие>]  { … }  **otherwise**  { … } | when [i > 0]  {  outln[“Positive”];  }  otherwise  {  outln[“Negative”];  } |
| Оператор цикла | **cycle** [<условие>]  { … } | cycle[i < 5]  {  out[i];  i = i + 1;  } |

**1.12 Операции языка**

В языке программирования RNA-2022 предусмотрены логические и арифметические операции, представленные в таблице 1.6.

Таблица 1.6 – Операции языка RNA-2022

|  |  |  |
| --- | --- | --- |
| Тип операций | Операции | Приоритет |
| Логические операции | > – больше  < – меньше  == – равно  ^= – не равно  >= – больше или равно  <= – меньше или равно | -1  -1  -1  -1  -1  -1 |
| Арифметические операции | + – сложение  - – вычитание  \* – умножение  / – деление  % –остаток от деления | 2  2  3  3  3 |
| Операция запятая | , | 1 |

Максимальным значением приоритетности является “3”, минимальным “0” соответственно. Логические операции приоритета не имеют. Для смены приоритета арифметических операций можно использовать оператор круглые скобки ().

**1.13 Выражения и их вычисление**

Выражения в языке RNA-2022 составляются согласно следующим правилам:

– Допускается использование оператора смены приоритета только в арифметических выражениях;

– Использование логических и арифметических операторов в одном выражении не допустимо;

– Использование двух и более идущих подряд операторов (за исключением оператора смены приоритета) запрещено;  
– Использование двух и более логических операторов в одном выражении запрещено;

– Вызов функции могут содержать только арифметические выражения или выражения присваивания;

– Вычисление сложных выражений (как минимум с одним оператором) внутри оператора возврата, в аргументах функции или процедуры, внутри условия цикла или условного оператора (за исключением логических операций) не производится.

**1.14 Конструкции языка**

Ключевые программные конструкции языка программирования RNA-2022 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка RNA-2022

|  |  |
| --- | --- |
| Главная функция | **program**  { … } |

Продолжение таблицы 1.7

|  |  |
| --- | --- |
| Процедура | **procedure** <идентификатор> [<тип данных> <идентификатор>, …]  {  …  } |
| Функция | **function** <тип данных> <идентификатор> [<тип данных> <идентификатор>, …]  {  …  give <выражение>  } |

**1.15 Области видимости идентификаторов**

В языке RNA-2022 переменные обязаны находится внутри программного блока функций или процедур. Переменные, объявленные в одной функции, недоступны в другой. Внутри разных областей видимости разрешено объявление переменных с одинаковыми именами. Все переменные и параметры внутри области видимости получают префикс, который отображается в таблице идентификаторов. Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости также не предусмотрено.

**1.16 Семантические проверки**

В языке программирования RNA-2022 выполняются следующие семантические проверки:

– Наличие функции **program** – точки входа в программу;

– Единственность точки входа;

– Переопределение идентификаторов;

– Использование идентификаторов без их объявления;

– Проверка соответствия типа функции и возвращаемого параметра;

– Правильность передаваемых в функцию параметров: количество, типы;

– Правильность строковых выражений;

– Превышение размера строковых и целочисленных литералов;

– Деление на ноль в арифметических операциях;

– Проверка на вызов функции в логических выражениях;

– Корректность использования операторов в выражениях.

**1.17 Распределение оперативной памяти на этапе выполнения**

Транслированный в язык ассемблера исходный код использует две области памяти. В сегмент констант заносятся все литералы. В сегмент данных заносятся переменные и параметры функций. Локальная область видимости в исходном коде регулируется префиксами идентификаторов, что и обуславливает их локальность на уровне исходного кода несмотря на то, что в языке ассемблера все переменные имеют глобальную область видимости.

**1.18 Стандартная библиотека и её состав**

Функции стандартной библиотеки языка RNA-2022 реализованы на языке C++. Описание функций приведено в таблице 1.8.

Таблица 1.8 – Состав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| concat | line | line a, line b | Функция производит конкатенацию строк a и b, возвращает строку. |
| linelen | number | line a | Функция вычисляет длину строки a. |
| random | number | number min, number max | Функция возвращает случайно сгенерированное число в диапазоне [min, max]. |
| sqrt | number | number a | Функция возвращает результат вычисление квадратного корня из числа a. |
| OutLine | - | line a | Функция выводит в консоль строку a. |
| OutNumber | - | number a | Функция выводит в консоль число a. |
| OutBool | - | bool a | Функция выводит в консоль булево значение a. |
| OutLineLn | - | line a | Функция выводит в консоль строку a с переносом на новую строку. |
| OutNumberLn | - | number a | Функция выводит в консоль число a с переносом на новую строку. |
| OutBoolLn | - | bool a | Функция выводит в консоль булево значение a с переносом на новую строку. |

**1.19 Ввод и вывод данных**

Вывод данных осуществляется с помощью функций **out** и **outln**. Допускается использование функций с литералами и идентификаторами.

В зависимости от типа параметра определяется функция: OutLine, OutBool, OutNumber, OutLineLn, OutBoolLn, OutNumberLn которые входят в состав стандартной библиотеки и описаны в таблице 1.8.

Функции ввода данных в языке RNA-2022 не предусмотрены.

**1.20 Точка входа**

Точкой входа является функция с именем program.

**1.21 Препроцессор**

Препроцессор в языке программирования RNA-2022 не предусмотрен.

**1.22 Соглашения о вызовах**

В языке RNA-2022 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**

Исходный код на языке RNA-2022 транслируется в язык ассемблера, а затем в объектный код.

**1.24 Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке RNA-2022 и выявления её транслятором в текущий файл протокола выводится сообщение. Классификация сообщений приведена в таблице 1.9.

Таблица 1.9 Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-110 | Системные ошибки, ошибки параметров |
| 200-299 | Ошибки лексического анализа |
| 300-399 | Ошибки семантического анализа |
| 600-699 | Ошибки синтаксического анализа |
| 400 – 499, 700 – 999 | Зарезервированные коды ошибок |

**1.25 Контрольный пример**

Контрольный пример демонстрирует главные особенности языка RNA-2022: функции, процедуры, фундаментальные типы, основные инструкции и операции, использование функций статической библиотеки. Исходный код контрольного примера представлен в приложении А.

**2. Структура транслятора**

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

В языке программирования RNA-2022 исходный текст программы транслируется в язык ассемблера. Основными компонентами транслятора являются лексический, синтаксический и сематический анализаторы, а также генератор кода на язык ассемблера. На этапе лексического анализа создаются таблицы лексем и идентификаторов, которые идут на вход синтаксического и семантического анализаторов. После прохождения проверки исходного кода всеми перечисленными анализаторами, таблицы лексем и идентификаторов идут на вход генератора кода. Структура транслятора языка RNA-2022 представлена на рисунке 2.1.



Рисунок 2.1 Структура транслятора RNA-2022

Первой стадией работы транслятора является лексический анализ, который выполняется лексическим анализатором. Входными данными для лексического анализатора является исходный текст программы, преобразованный в единую последовательность символов. Анализатор осуществляет разбор текста, разбивая его на отдельные слова. Далее он заменяет лексические единицы их внутренним представлением – лексемами, которые используются для промежуточного представления исходного текста программы. Все лексемы записываются в таблицу лексем. Также формируется таблица идентификаторов, в которой хранится информация о имени идентификатора, его типе данных, значении, а также его первом вхождении в таблицу лексем. Таблица лексем и таблица идентификаторов являются входом для следующей фазы транслятора – синтаксического анализа.

Назначение лексического анализатора:

− убрать все лишние пробелы;

− выполнить распознавание лексем;

− построить таблицу лексем и таблицу идентификаторов;

− при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

Второй стадией трансляции является синтаксический анализ. Синтаксический анализатор выполняет проверку исходного кода на соответствие правилам грамматики языка программирования RNA-2022.

Назначение синтаксического анализатора:

− сопоставить последовательности лексем правилам грамматики;

− построить дерево разбора;

− выявить синтаксические ошибки при их наличии и выдать сообщение об ошибке.

Третьей стадией трансляции является семантический анализ. На вход семантического анализатора поступаются таблицы лексем и идентификаторов. Анализатор отслеживает ошибки, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Семантические проверки, осуществляемые анализатором, описаны в пункте 1.16 раздела 1.

Четвертой стадией трансляции является генерация ассемблерного кода на основе полученных данных на предыдущих этапах трансляции (таблица лексем, таблица идентификаторов). Более подробно структура генератора описана в главе 7.

**2.2 Перечень входных параметров транслятора**

Входные параметры транслятора представлены в таблице 2.1.

Таблица 2.1 Входные параметры транслятора языка RNA-2022

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с расширением .txt, в котором содержится исходный код на RNA-2022 | Не предусмотрено |
| -log:<имя\_файла> | Файл для записи полного протокола работы транслятора | <имя\_файла>.log |
| -out:<имя\_файла> | Файл для записи результата работы транслятора | <имя\_файла>.out.asm |
| -poliz | Ключ для вывода на консоль промежуточного представления кода после преобразования в польскую инверсную запись | По умолчанию отсутствует |
| -lt | Ключ для вывода таблицы лексем на консоль | По умолчанию отсутствует |
| -id | Ключ для вывода таблицы идентификаторов на консоль | По умолчанию отсутствует |

**2.3 Протоколы, формируемые транслятором**

В ходе трансляции формируются протоколы работы лексического, синтаксического и семантического анализаторов. Перечень протоколов, формируемых транслятором языка программирования их описание представленные в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка RNA-2022

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, заданный параметром -log | Содержит информацию о времени выполнения приложения; входных параметрах в приложение; код на языке YNK-2020 с сепараторами и без избыточных пробелов, табуляций и переходов на новую строку; таблицу идентификаторов; таблицу лексем; промежуточное представление кода; трассировку синтаксического анализа; дерево разбора, время выполнения разбора; промежуточное представление кода после приведения его к польской нотации. |
| Выходной файл, заданный параметром -out: | Содержит сгенерированный код на языке Ассемблера. |

# **3.** **Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор — это часть компилятора, которая читает исходную программу и выделяет в ее тексте лексемы входного языка. На вход лексического анализатора поступает текст исходной программы. Результатом работы лексического анализатора является перечень всех найденных в тексте исходной программы лексем. Этот перечень лексем представляется в виде таблицы, называемой таблицей лексем. Таблица лексем фактически содержит весь текст исходной программы, обработанный лексическим анализатором. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 Структура лексического анализатора

## **3.2 Контроль входных символов**

Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. Таблица входных символов представлена на рисунке 3.2, категории входных символов представлены в таблице 3.1.

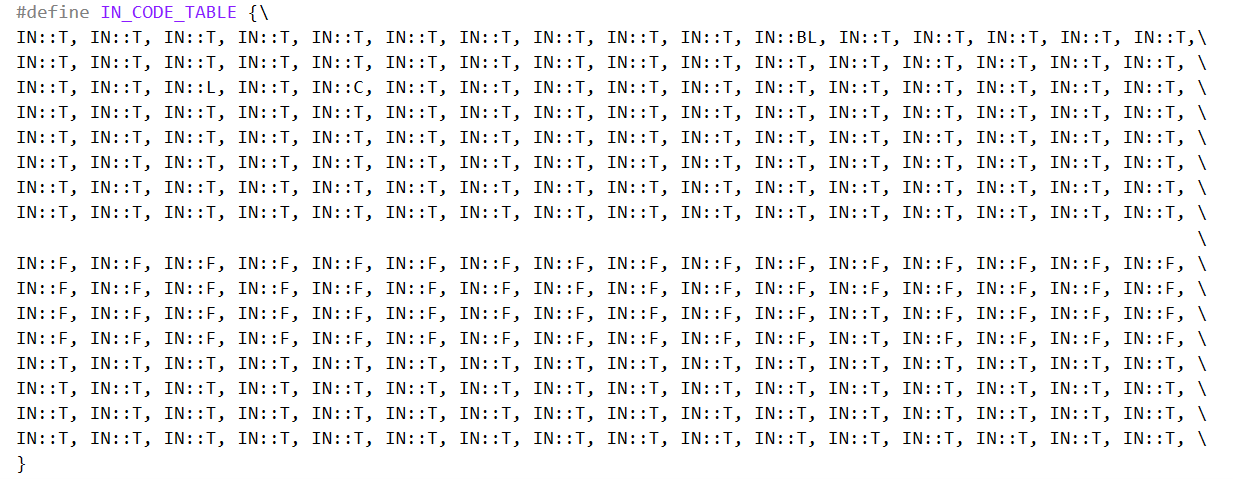


Рисунок 3.2. Таблица контроля входных символов

Таблица 3.1 Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Символы | Значение в таблице входных символов |
| T | Разрешенный |
| F | Запрещенный |
| BL | Перевод строки |
| C | Признак начала комментария |
| L | Строковый литерал |

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на лексемы.

Описание алгоритма удаления избыточных символов:

1. Посимвольное считывание исходного кода, занесенного в структуру In.
2. Встреча пробела или знака табуляции вне пределов строкового литерала или комментария является встречей символа-сепаратора.
3. В отличие от других символов-сепараторов табуляции и пробелы игнорируются, т.е. не записываются в таблицу лексем.

## **Перечень ключевых слов**

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие ключевых слов и лексем приведено в таблице 3.2.

Таблица 3.2 Соответствие ключевых слов, символов операций и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | def | d |
| number, line, bool | t |
| program | p |
| function | f |
| procedure | s |
| give | r |
| out | o |
| cycle | u |
| when | w |
| otherwise | ! |
| Иное | Идентификатор | i |
| Литерал | l |
| Функции стандартной библиотеки | concat | + |
| linelen | % |
| sqrt | q |
| outln | b |

Продолжение таблицы 3.2

|  |  |  |
| --- | --- | --- |
|  | random | z |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| [ | [ |
| ] | ] |
| Операторы | Арифметические (+, -, \*, /, %) | v |
| Логические (== != > < >= <=) | g |
| Присваивание (=) | = |

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Структура конечного автомата и пример графа перехода конечного автомата изображены на рисунках 3.3 и 3.4 соответственно.

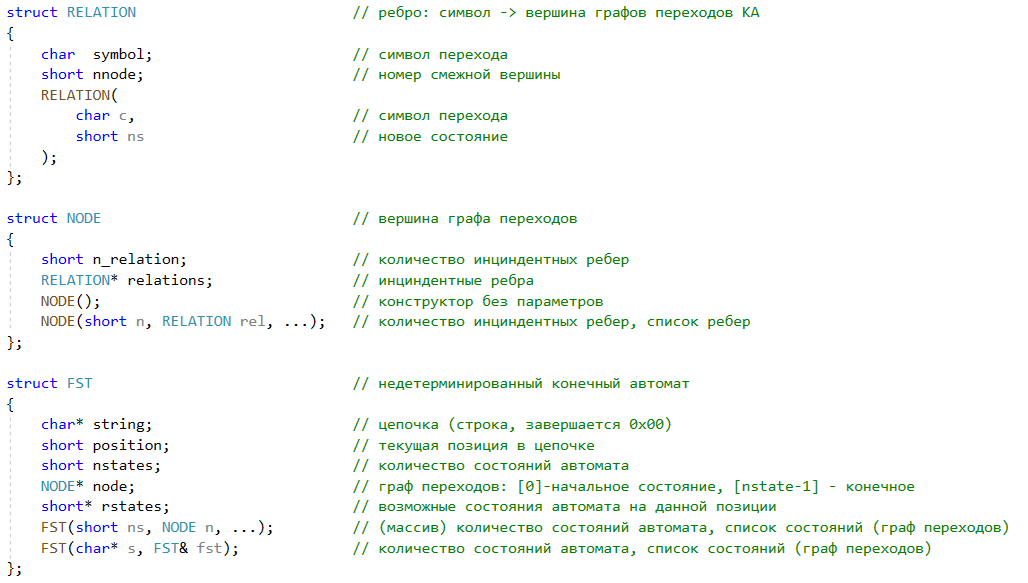


Рисунок 3.3 Структура конечного автомата

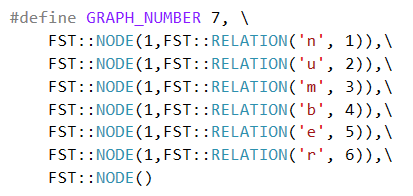


Рисунок 3.4 Граф конечного автомата для ключевого слова number

В приложении Б представлены графы конечных автоматов для других слов языка.

## **Основные структуры данных**

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Код на языке C++ со структурой таблицы лексем представлен на рисунке 3.3. Код на языке C++ со структурой таблицы идентификаторов представлен на рисунке 3.4.

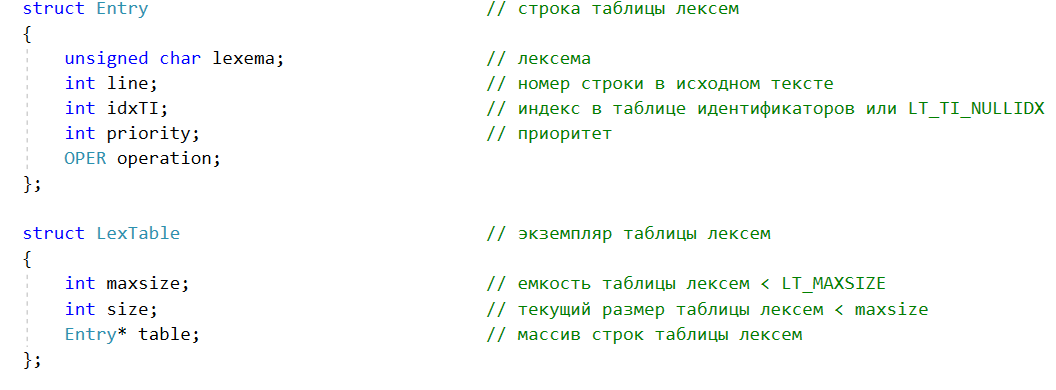


Рисунок 3.3 Структура таблицы лексем

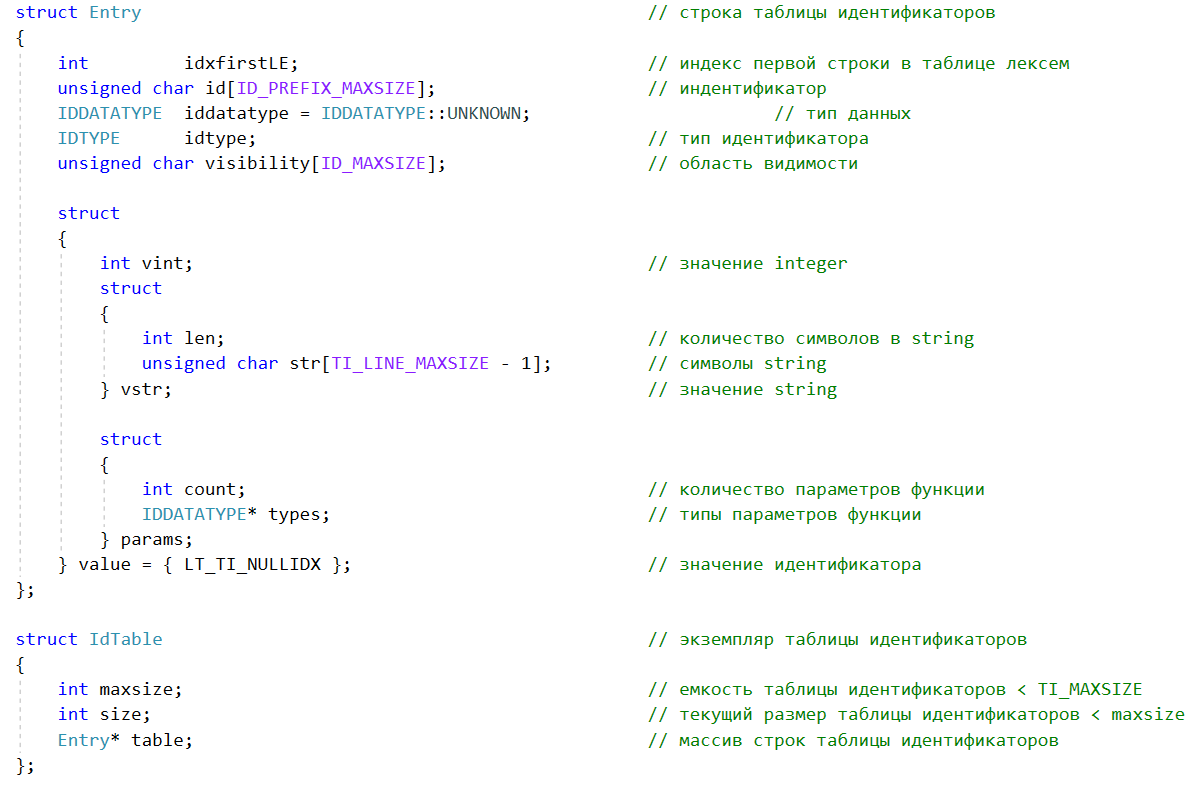


Рисунок 3.4 Структура таблицы идентификаторов

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

Перечень сообщений лексического анализатора представлен на рисунке 3.5.

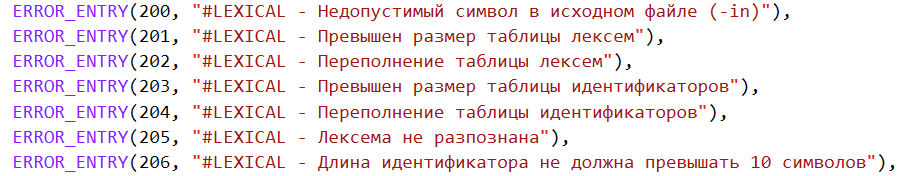


Рисунок 3.5 Сообщения лексического анализатора

## **Принцип обработки ошибок**

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. При возникновении сообщения, лексический анализатор останавливает работу и далее сообщение выводится в файл протокола.

## **Параметры лексического анализатора**

Входными параметрами для лексического анализатора является исходный текст программы, написанный на языке RNA-2022, а также файл протокола в который записываются выходные данные (таблица лексем и таблица идентификаторов).

## **Алгоритм лексического анализа**

Лексический анализ основывается на работе конечных автоматов, разбирающих регулярные выражения. Регулярные выражения — аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа.

Алгоритм лексического анализа:

− из входного потока символов программы на исходном языке удаляются лишние пробелы и добавляется сепаратор для вычисления номера строки для каждой лексемы;

− формируется массив из слов языка;

− для каждого слова выполняется функция распознавания лексемы;

− при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;

− при неуспешном распознавании выдается сообщение об ошибке;

− формируется протокол работы.

## **Контрольный пример**

Результат работы лексического анализатора в виде таблиц лексем и идентификаторов, соответствующих контрольному примеру, представлен в приложении В.

# **4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализатор – это часть компилятора, выполняющая синтаксический анализ, то есть исходный код проверяется на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные на этапе лексического анализа. Выходной информацией является дерево разбора.

Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора

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

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

T – множество терминальных символов,

N – множество нетерминальных символов,

P – множество правил языка,

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 RNA-2022 реализованные на языке C++ представлены в приложении Г.

Перечень правил, составляющих грамматику языка представлен в таблице 4.1.

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

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | ftiFBS  siFUS  p{N}  ftiFB  siFU | Проверка правильности структуры программы |
| F | [P]  [] | Проверка наличия списка параметров функции |
| P | ti  ti,P | Проверка на ошибку в параметрах функции при ее объявлении |
| B | {Nr[I];}  {r[I];} | Проверка наличия тела функции |
| I | l  i | Проверка на недопустимое выражение (ожидается только литерал или идентификатор) |
| N | dti;N  dti;  dti=R;N  dti=E;N  dti=E;  i=R;N  i=E;N  i=E;  u[R]{X}N  u[R]{X}  w[R]{X}N  w[R]{X}  w[R]{X}!{X}N  w[R]{X}!{X}  %K;N  %K;  +K;N  +K;  qK;N  qK;  zK;N  zK;  o[I];N  o[I];  b[I];N  oK;  bK;  iK;N  iK; | Проверка на неверную конструкцию в теле функции |
| U | {N} | Проверка на ошибку в теле процедуры |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| R | i  l  igi  igl  lgi  lgl | Проверка на ошибку в условном выражении |
| K | [W]  [] | Проверка на ошибку в вызове функции |
| E | i  iM  l  lM  (E)  (E)M  iK  iKM  %K  %KM  +K  +KM  qK  qKM  zK  zKM | Проверка на ошибку в арифметическом выражении |
| W | i  l  i,W  l,W | Проверка на ошибку в параметрах вызываемой функции |
| M | vE  vEM | Проверка арифметических действий |
| X | dti;N  dti;  dti=E;N  dti=E;  dti=R;N  dti=R;  i=R;N  i=R;  i=E;N  i=E;  %K;N  %K;  +K;N  +K;  qK;N | Проверка на неверную конструкцию в теле цикла или условного выражения |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
|  | qK;  zK;N  zK;  o[I];N  oK;  b[I];N  bK;  iK;N  iK;  r[I];N  r[I]; |  |

## **4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку, описание которой представлено в таблице 4.2.

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 представляет из себя множества терминальных и нетерминальных символов. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $). |
|  | Функция переходов автомата | Функция которая представляет из себя множество правил грамматики. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики. |
|  | 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**

Основными структурами данных синтаксического анализатора являются структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка RNA-2022. Данные структуры представлены в приложении Д.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

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 выбирается другая цепочка;
6. Если в магазине встретился нетерминал, переход к пункту 4;
7. Если 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 на рисунке 4.2.

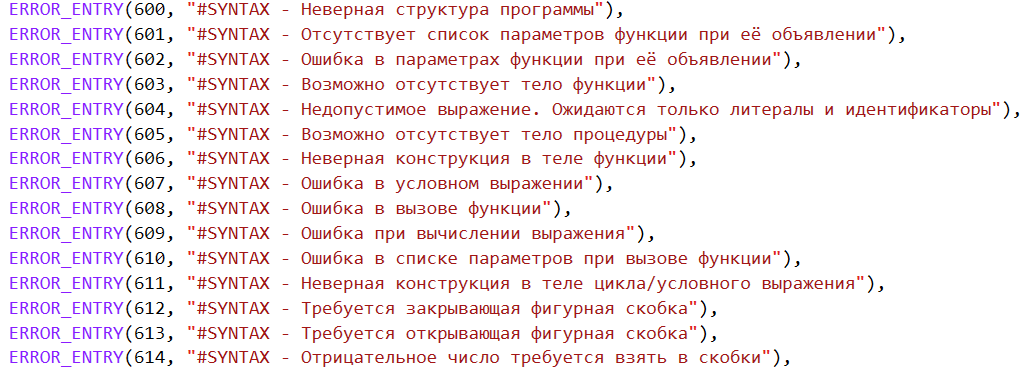


Рисунок 4.3 Сообщения синтаксического анализатора

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

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

Выходными параметрами являются трассировка прохода таблицы лексем и дерево 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.

# **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 семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1. Структура семантического анализатора

## **5.2 Функции семантического анализатора**

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

За семантический анализ отвечает функция AnalyzeSem. Ее входными параметрами является таблица лексем и таблица идентификаторов.

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

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

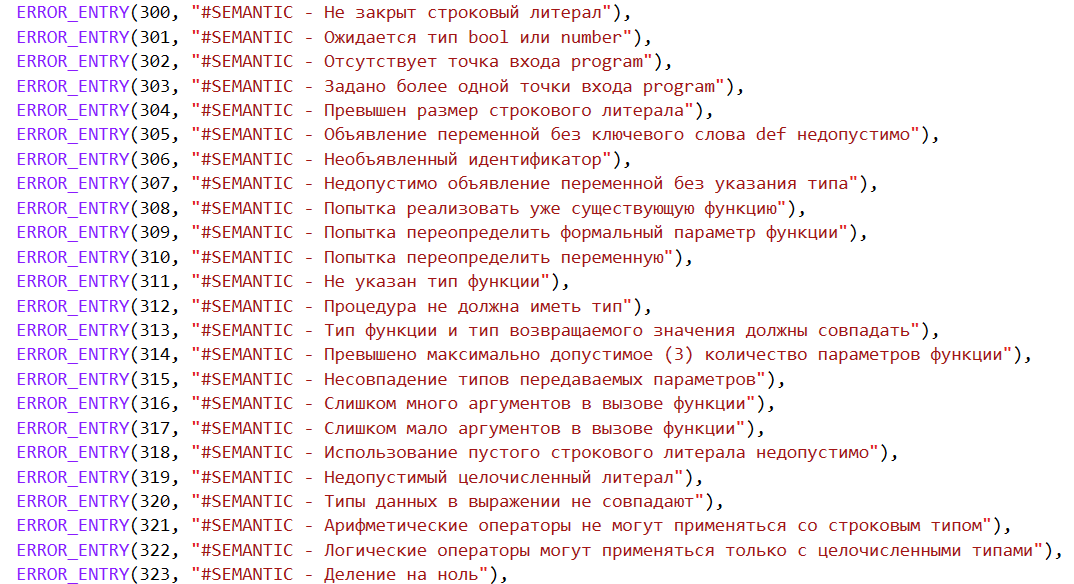


Рис. 5.2 – перечень сообщений семантического анализатора

## **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; семантический анализ останавливается.

## **5.5 Контрольный пример**

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

Таблица 5.1 – Примеры семантических ошибок

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| {  def number a = 5;  outln[a];  } | Ошибка 302: #SEMANTIC - Отсутствует точка входа program  Строка -1 позиция -1 |
| program  {  def line a = "Test;  outln[a];  } | Ошибка 300: #SEMANTIC - Не закрыт строковый литерал  Строка 3 позиция 15 |
| program  {  line a = "Test";  outln[a];  } | Ошибка 305: #SEMANTIC - Объявление переменной без ключевого слова def недопустимо  Строка 3 позиция 7 |

Продолжение таблицы 5.1

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| program  {  def line a = 99999999999;  outln[a];  } | Ошибка 319: #SEMANTIC - Недопустимый целочисленный литерал  Строка 3 позиция 15 |
| program  {  def a = 67;  outln[a];  } | Ошибка 307: #SEMANTIC - Недопустимо объявление переменной без указания типа  Строка 3 позиция 6 |
| program  {  def line a = 67;  outln[a];  } | Ошибка 320: #SEMANTIC - Типы данных в выражении не совпадают  Строка 3 позиция -1 |
| function MyFunc [number a, number b]  {  …  } | Ошибка 311: #SEMANTIC - Не указан тип функции  Строка 1 позиция -1 |
| procedure number MyProc [number a]  {  …  } | Ошибка 312: #SEMANTIC - Процедура не должна иметь тип  Строка 1 позиция -1 |
| program  {  def line a = "Hello, " + "World!";  } | Ошибка 321: #SEMANTIC - Арифметические операторы не могут применяться со строковым типом  Строка 3 позиция -1 |
| function number MyFunc [number a]  {  def line lnA = "Hello";  give[lnA];  }  program  {  …  } | Ошибка 313: #SEMANTIC - Тип функции и тип возвращаемого значения должны совпадать  Строка 4 позиция -1 |
| program  {  def bool b = true < false;  } | Ошибка 322: #SEMANTIC - Логические операторы могут применяться только с целочисленными типами  Строка 3 позиция -1 |

# **6. Вычисление выражений**

## **6.1 Выражения, допускаемые языком**

В языке RNA-2022 допускается вычисление выражений целочисленного типа, также поддерживается вызов функций внутри арифметических выражений. В выражениях могут использоваться арифметические и логические операции. Операции и их приоритет представлен в таблице 6.1.

Таблица 6.1 – Операции и их приоритет

|  |  |  |
| --- | --- | --- |
| Тип операций | Операции | Приоритет |
| Логические операции | > – больше  < – меньше  == – равно  ^= – не равно  >= – больше или равно  <= – меньше или равно | -1  -1  -1  -1  -1  -1 |
| Арифметические операции | + – сложение  - – вычитание  \* – умножение  / – деление  % –остаток от деления | 2  2  3  3  3 |

## Примеры выражений языка RNA-2022 представлены на рисунке 6.1.

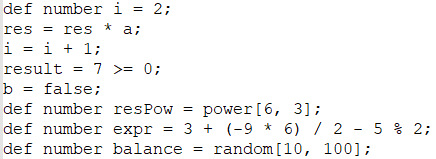


Рисунок 6.1 – Примеры выражений

## **6.2 Польская запись и принцип её построения**

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

Обратная польская запись — это форма записи математических выражений, в которой операторы 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;

– результирующая строка: польская запись;

– исходная строка просматривается слева направо;

– операнды переносятся в результирующую строку в порядке их следования;

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

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

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

– отрывающая скобка помещается в стек;

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

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

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

В таблице 6.2 представлен пример преобразования выражения в обратную польскую запись.

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| z[l,l]vl | - | - |
| [l,l]vl | z | - |
| l,l]vl | z | [ |
| ,l]vl | zl | [ |
| l]vl | zl | [ |
| ]vl | zll | [ |
| vl | zll@2 | - |
| l | zll@2 | v |
| - | zll@2l | v |
| - | zll@2lv | - |

## **6.3 Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений в обратную польскую запись основана на функциях Poliz и StartPoliz. Функция StartPoliz принимает как параметр таблицу лексем и таблицу идентификаторов и содержит цикл, в ходе которого перебираются все лексемы исходного кода. Если последовательность лексем соответствует началу выражения, вызывается функция Poliz, где и проводится преобразование выражений к польской нотации.

## **6.4 Контрольный пример**

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

# **7. Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы в цепочку символов выходного языка. На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом. Структура генератора кода представлена на рисунке 7.1.



Рисунок 7.1 Структура генератора кода

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

Элементы таблицы идентификаторов расположены сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке RNA-2022 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка RNA-2022 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке RNA-2022 | Тип идентификатора на языке ассемблера | Пояснение |
| number | sword | Хранит целочисленный тип данных. |
| bool | sword | Хранит булевый тип данных (в виде целого числа) |
| line | dword | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |

## **7.3 Статическая библиотека**

В языке RNA-2022 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Подключение библиотеки происходит с помощью includelib на этапе генерации кода. C помощью оператора EXTRN объявляются функции из библиотеки. Пример подключения библиотеки в исходном коде на языке ассемблера представлен на рисунке 7.2.

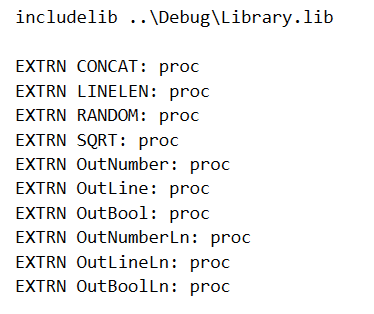


Рисунок 7.2 – Подключение статической библиотеки

## **7.4 Особенности алгоритма генерации кода**

В языке RNA-2022 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.3.

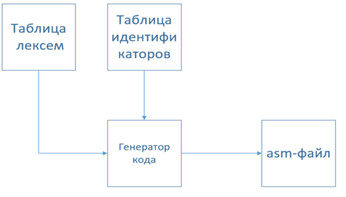


Рисунок 7.3 – Схема работы генератора кода

В таблице 7.2 представлены прототипы функций, осуществляющих генерацию года, и их описание.

Таблица 7.2 – Прототипы функций, осуществляющих генерацию кода

|  |  |
| --- | --- |
| void Generation(LT::LexTable, IT::IdTable, wchar\_t) | Основная функция. Формирует поток выходного файла и вызывает другие генерирующие функции. |
| void Head(ofstream\*); | Функция, генерирующая заголовок ассемблерного файла (подключение библиотек, указание прототипов функций и т.д.). |
| void ConstSegment(IT::IdTable, ofstream\*); | Функция, генерирующая сегмент констант. |
| void DataSegment (LT::LexTable, IT::IdTable, ofstream\*) | Функция, генерирующая сегмент данных. |

Продолжение таблицы 7.2

|  |  |
| --- | --- |
| void CodeSegment (LT::LexTable, IT::IdTable, ofstream\*) | Функция, генерирующая сегмент кода. |

## **Входные параметры генератора кода**

На вход генератору кода поступают таблицы лексем и идентификаторов. Результаты работы генератора кода выводятся в файл с расширением .asm.

## **7.6 Контрольный пример**

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении З. Результат работы контрольного примера приведён на рисунке 7.4.

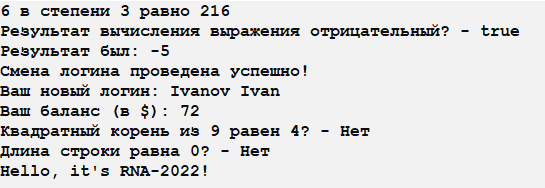


Рисунок 7.4 – Результат работы программы на языке RNA-2022

**8. Тестирование транслятора**

## **8.1 Общие положения**

В ходе работы транслятора могут возникать ошибки на разных этапах трансляции: анализ исходного текста программы, лексических анализ, синтаксический анализ, семантический анализ. Возникшие ошибки отслеживаются транслятором и заносятся в файл протокола, с указанием идентификатора ошибки, сообщения, строки и позиции в исходном тексте программы. Как правило, после возникновения ошибки работа транслятора прекращается, поскольку ошибка на одном этапе трансляции может вызвать ошибки на последующих этапах (за исключением синтаксического анализатора).

## **8.2 Результаты тестирования**

В таблице 8.1 приведены результаты тестов для разных этапов трансляции.

Таблица 8.1 – Результаты тестов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| Проверка на допустимость символов | |
| pro№gram  {  } | Ошибка 200: #LEXICAL - Недопустимый символ в исходном файле (-in)  Строка 1 позиция 4 |
| Лексический анализ | |
| function number func! [number a, number b]  { …} | Ошибка 205: #LEXICAL - Лексема не разпознана  Строка 1 позиция 17 |
| function number MyLongNamedFunction [number a]  {…} | Ошибка 206: #LEXICAL - Длина идентификатора не должна превышать 10 символов  Строка 1 позиция 17 |
| Синтаксический анализ | |
| program def number a  {…} | Ошибка 600: строка 1, #SYNTAX - Неверная структура программы |
| function number myFunc  {  give[0];  }  program  {…} | Ошибка 601: строка 2, #SYNTAX - Отсутствует список параметров функции при её объявлении |
| function number func [number]  {  give[8];  }  program  {…} | Ошибка 602: строка 1, #SYNTAX - Ошибка в параметрах функции при её объявлении |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| function number myFunc []  program  {…} | Ошибка 603: строка 3, #SYNTAX - Возможно отсутствует тело функции |
| function number myFunc []  {  give[1 + 1];  }  program  {…} | Ошибка 604: строка 3, #SYNTAX - Недопустимое выражение. Ожидаются только литералы и идентификаторы |
| procedure myProc []  program  {…} | Ошибка 605: строка 3, #SYNTAX - Возможно отсутствует тело процедуры |
| function number myFunc []  {  program  {  def number a = 9;  }  } | Ошибка 606: строка 3, #SYNTAX - Неверная конструкция в теле функции |
| program  {  when[1 > 8 > 9]  {  def number a = 1;  }  } | Ошибка 607: строка 3, #SYNTAX - Ошибка в условном выражении |
| function number myFunc [number a]  {  a = a + 2;  give[a];  }  program  {  def number num = myFunc;  } | Ошибка 608: #SYNTAX - Ошибка в вызове функции  Строка 9 позиция -1 |
| program  {  def number num = 9 \* 8);  } | Ошибка 609: строка 3, #SYNTAX - Ошибка при вычислении выражения |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| function number myFunc[number a, number b]  {  def number res = a + b;  give[res];  }  program  {  def number res = myFunc[2,,3];  } | Ошибка 610: строка 8, #SYNTAX - Ошибка в списке параметров при вызове функции |
| def number i = 0;  when[i ^= 8]  {  cycle[i < 7]  {  i = i + 1;  }  }  } | Ошибка 611: строка 6, #SYNTAX - Неверная конструкция в теле цикла/условного выражения |
| program  def number a = 9;  } | Ошибка 613: #SYNTAX - Требуется открывающая фигурная скобка |
| program  {  def number a; | Ошибка 612: #SYNTAX - Требуется закрывающая фигурная скобка |
| …  def number a = 8 - -9; | Ошибка 614: #SYNTAX - Отрицательное число требуется взять в скобки  Строка 3 позиция -1 |
| Семантический анализ | |
| when["line"]  {  out[true];  } | Ошибка 301: #SEMANTIC - Ожидается тип bool или number  Строка 3 позиция -1 |
| …  when[a == 1] {…} | Ошибка 306: #SEMANTIC - Необъявленный идентификатор  Строка 3 позиция -1 |
| function number MyFunc[number a]{…}  function number MyFunc[number a, number b]{…} | Ошибка 308: #SEMANTIC - Попытка реализовать уже существующую функцию  Строка 5 позиция -1 |
| function number MyFunc[number a, number b]  { def number a = 9; } | Ошибка 309: #SEMANTIC - Попытка переопределить формальный параметр функции  Строка 3 позиция -1 |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| def line ln = "hello" + "world"; | Ошибка 321: #SEMANTIC - Арифметические операторы не могут применяться со строковым типом  Строка 3 позиция -1 |
| def number a = 9 / 0; | Ошибка 323: #SEMANTIC - Деление на ноль  Строка 3 позиция -1 |

Некоторые сообщения семантического анализатора были описаны в подразделе 5.5.

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования RNA-2022. Были выполнены основные задачи курсового проекта:

– разработана спецификация языка программирования;

– разработана структура транслятора;

– разработан лексический анализатор;

– разработан синтаксический анализатор;

– разработан семантический анализатор;

– разработан генератор кода на язык ассемблера;

– проведено тестирование транслятора.

Итоговая версия языка RNA-2022 включает:

* 3 типа данных;
* поддержку операторов вывода;
* Возможность вызова функций стандартной библиотеки;
* Наличие 5 арифметических операторов для вычисления выражений;
* Наличие 6 логических операторов для вычисления выражений;
* Поддержка функций, процедур, операторов цикла и условия;

Таким образом, в ходе выполнения курсового проекта были получены новые знания и навыки в проектировании систем программирования и в разработке программного обеспечения для систем программирования.

# **Список использованных литературных источников**

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

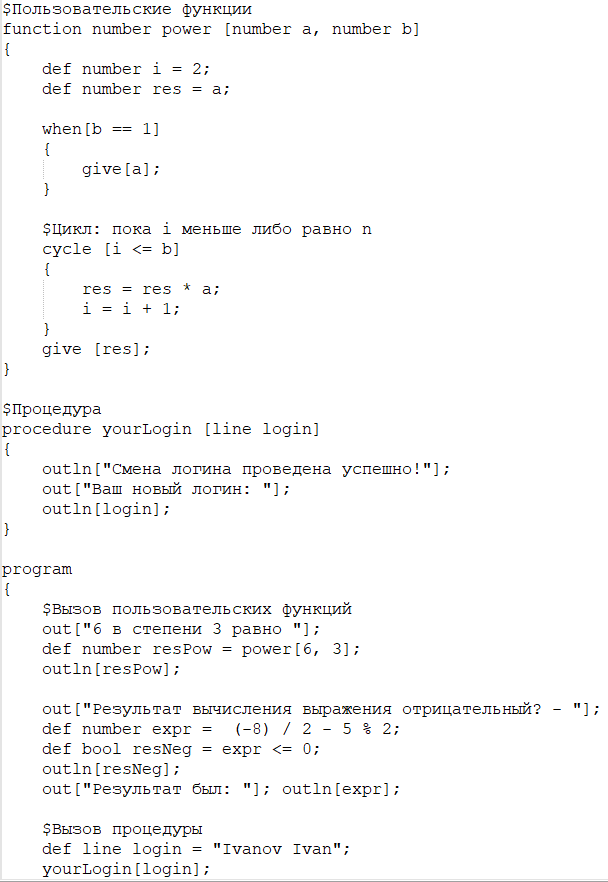
2. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.

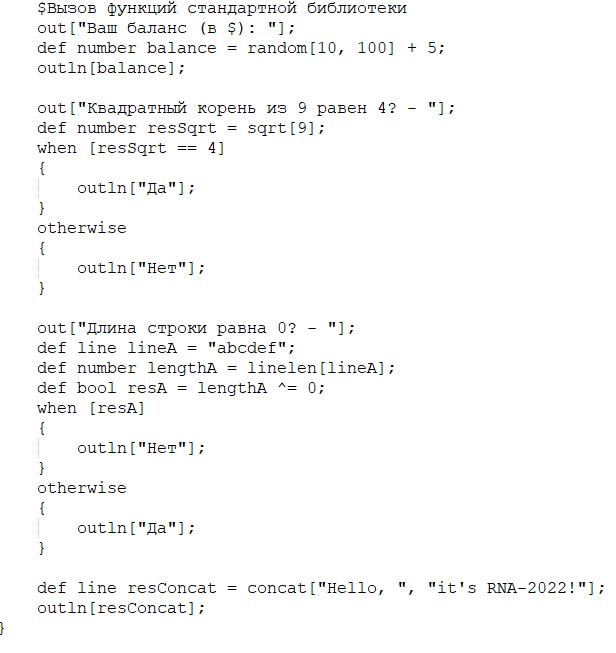
3. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.

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

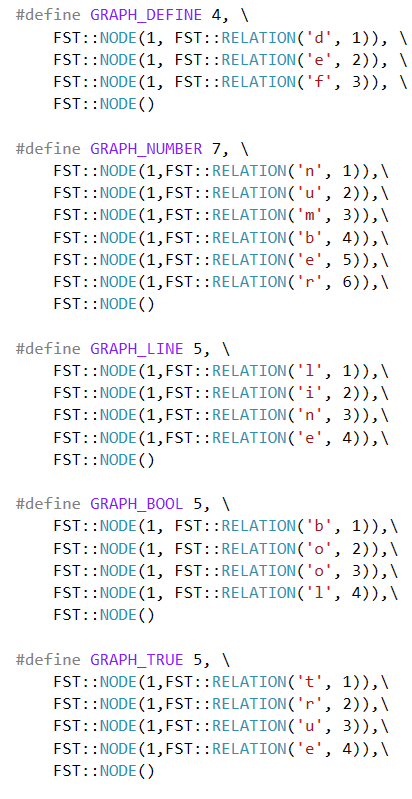
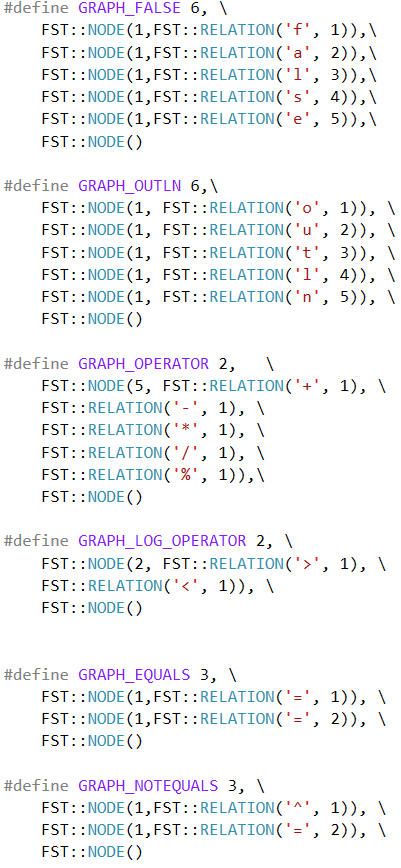
5. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.

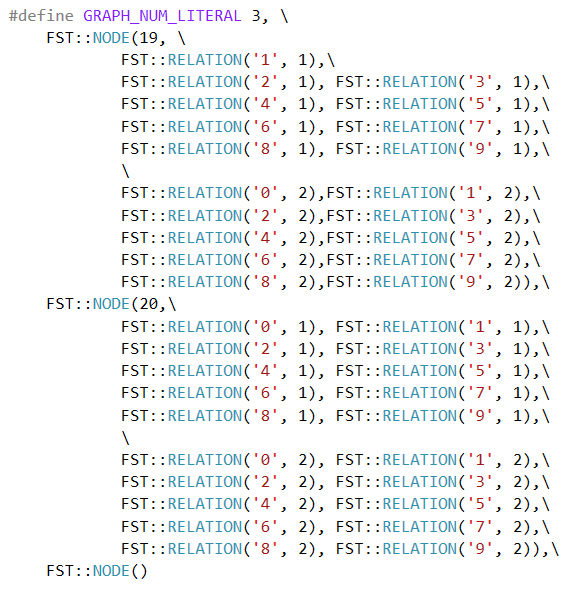
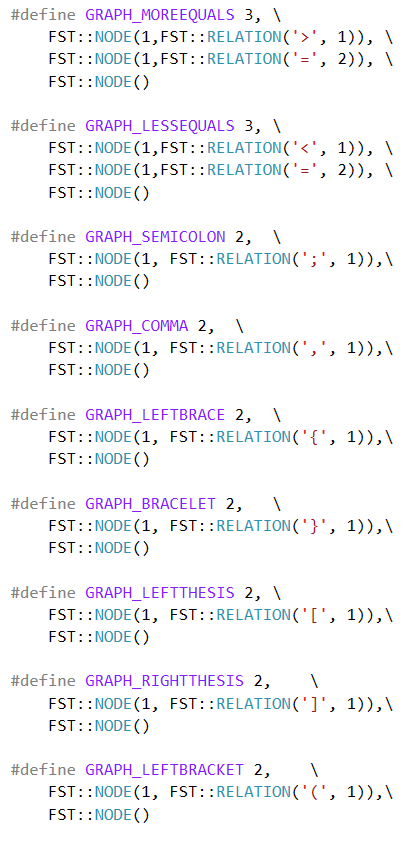
**Приложение А**

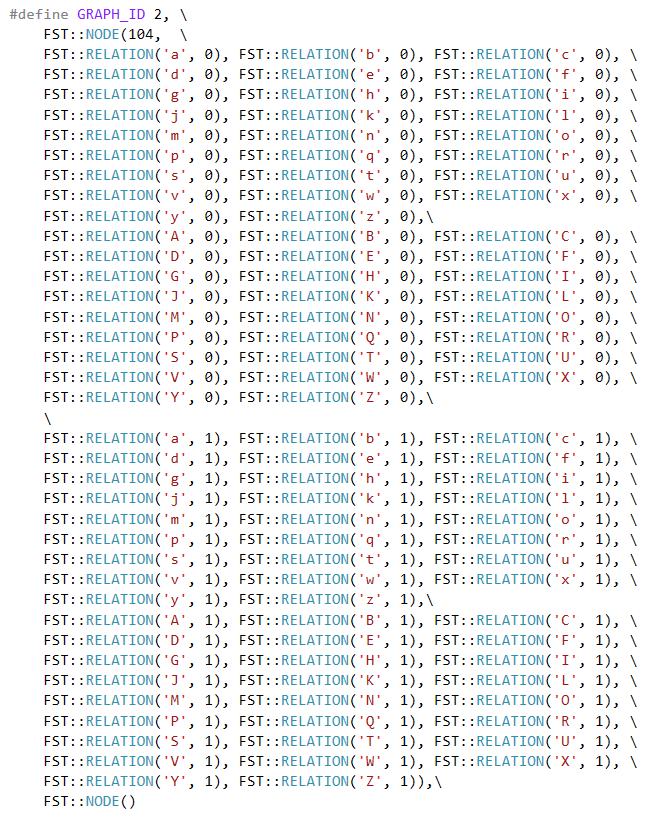
****

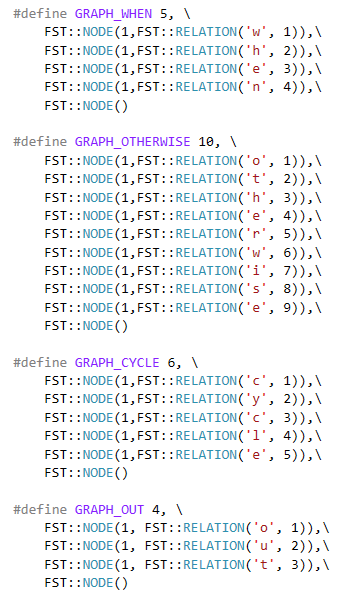
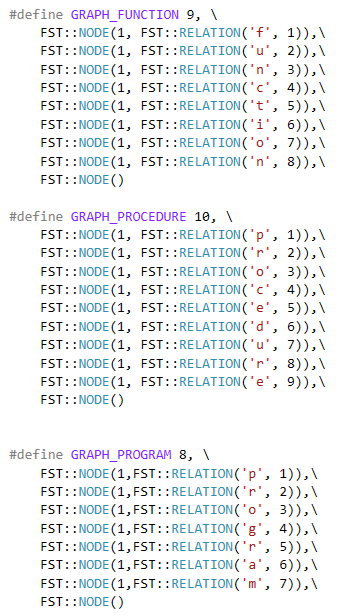
****

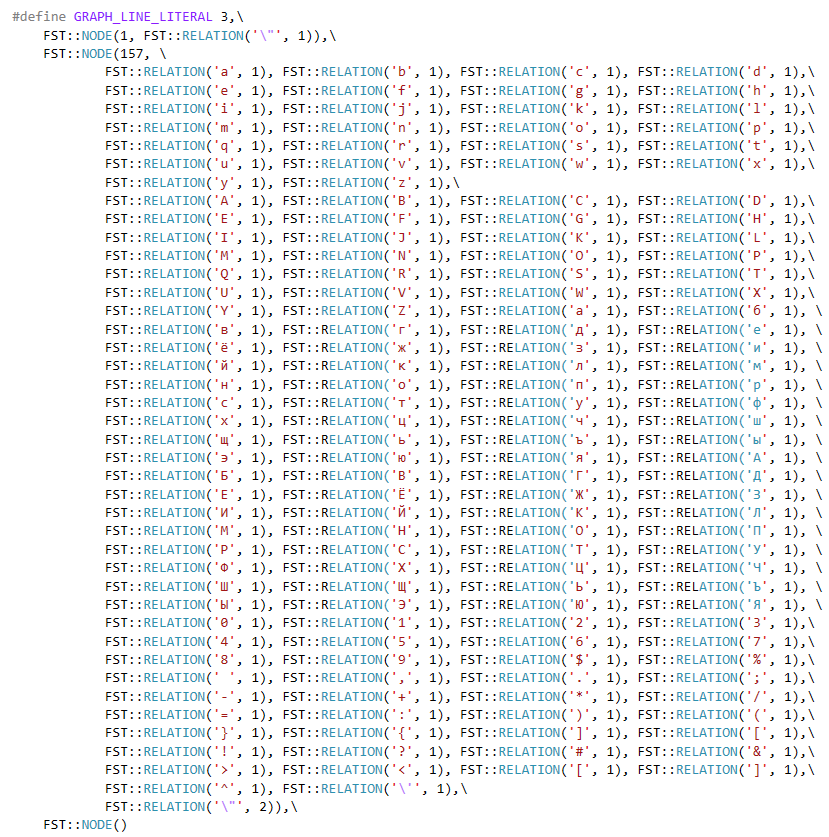
**Приложение Б**

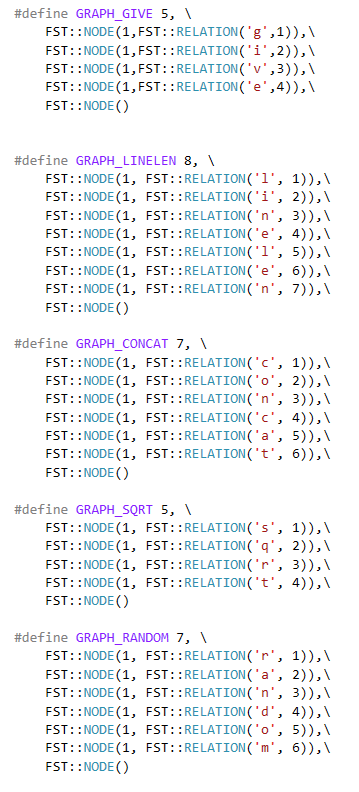
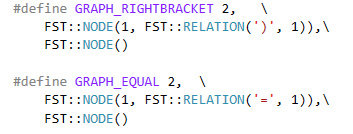
** **

** **

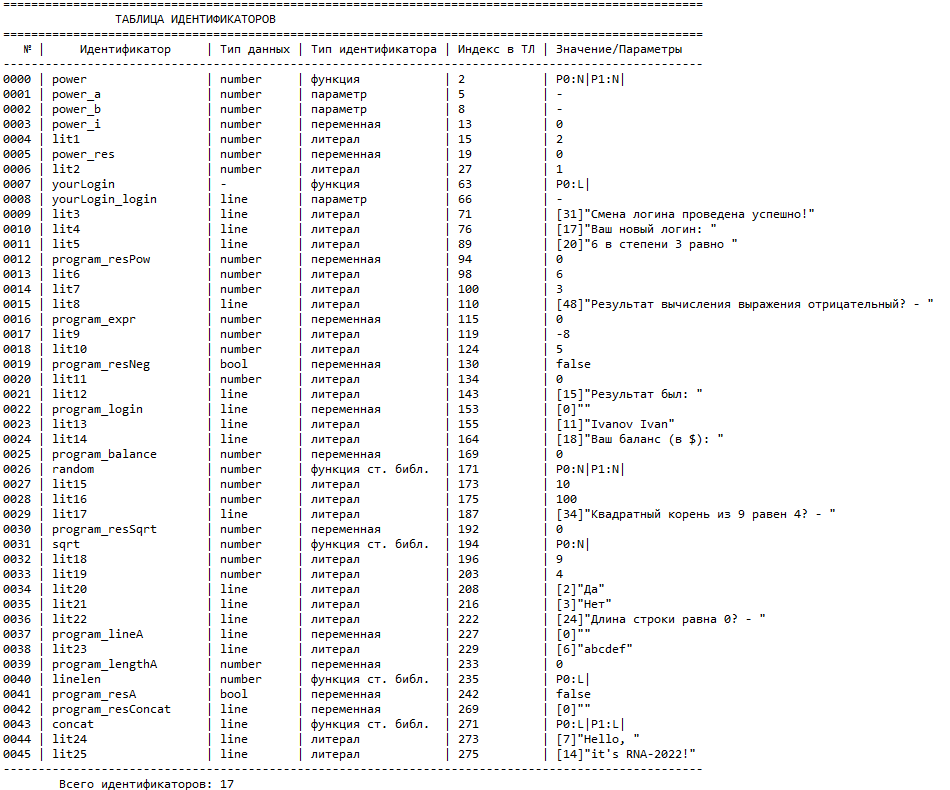
****

****

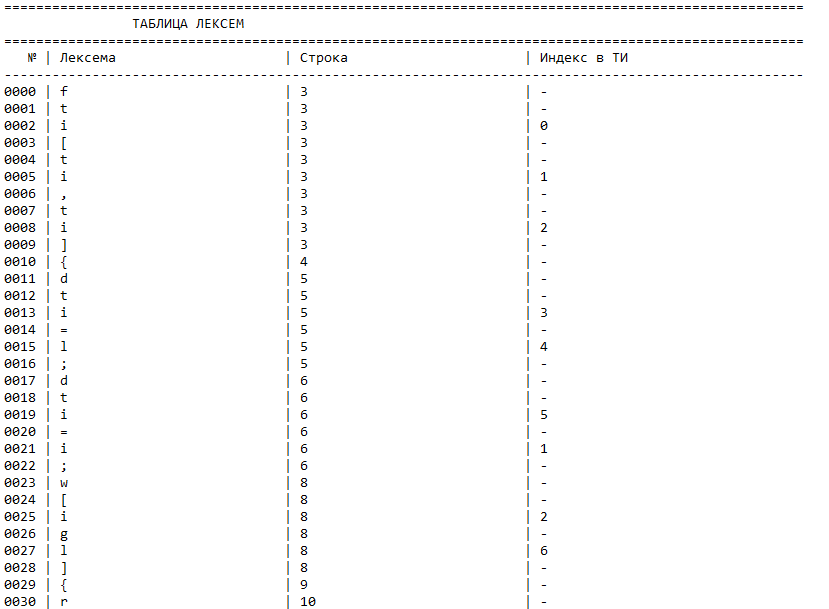
****

****

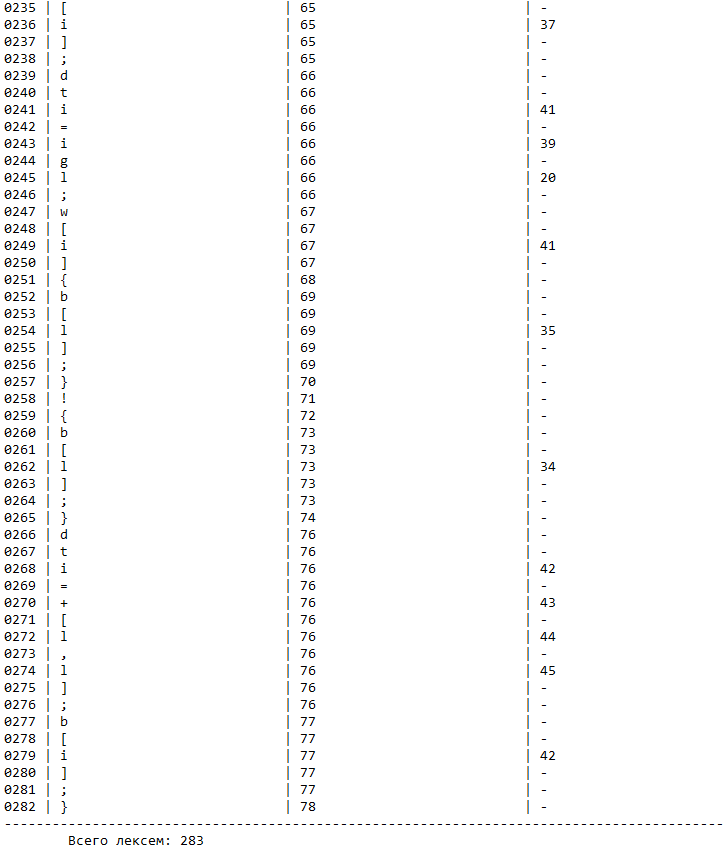
**Приложение В**

****

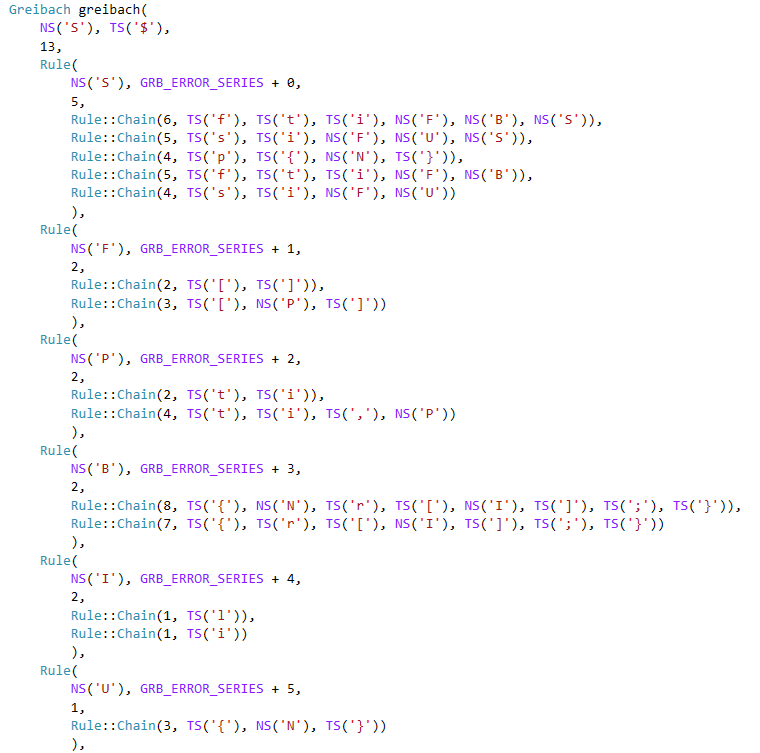
Начало таблицы лексем

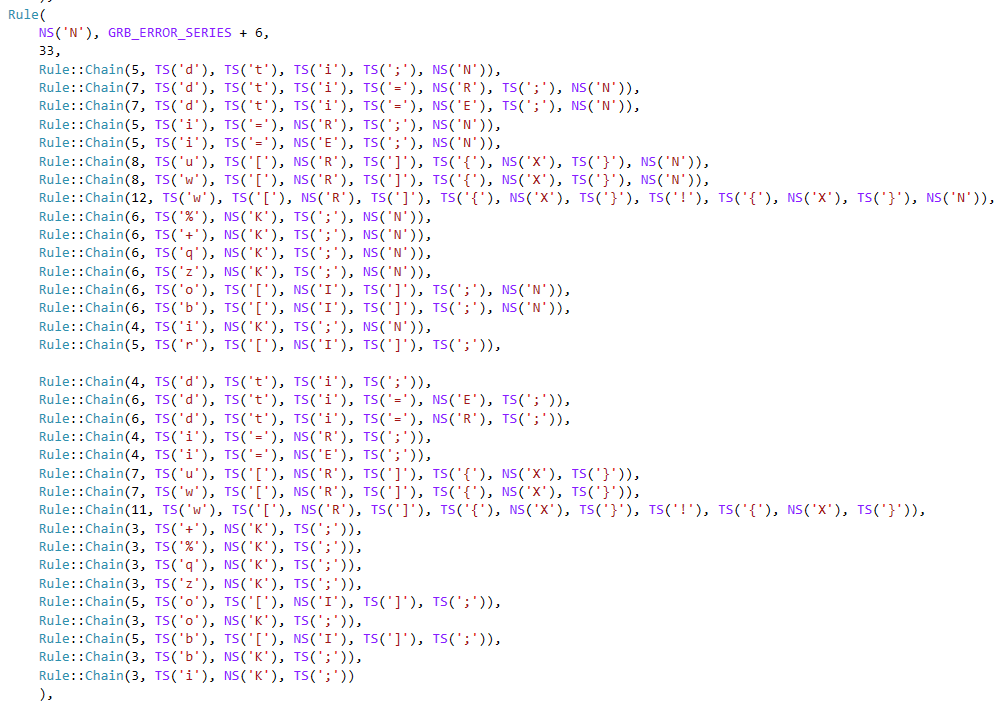
****

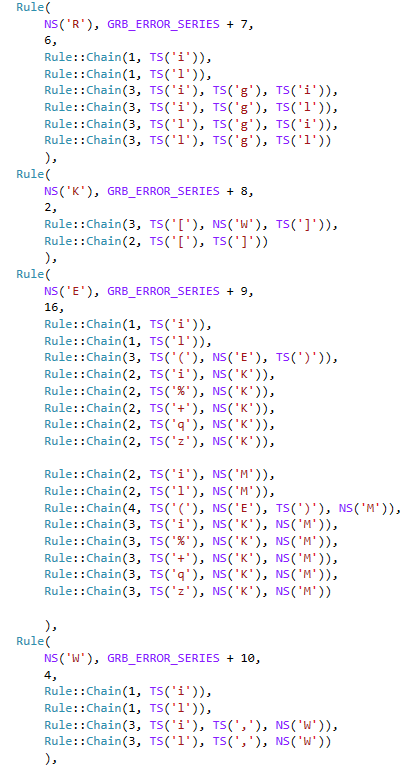
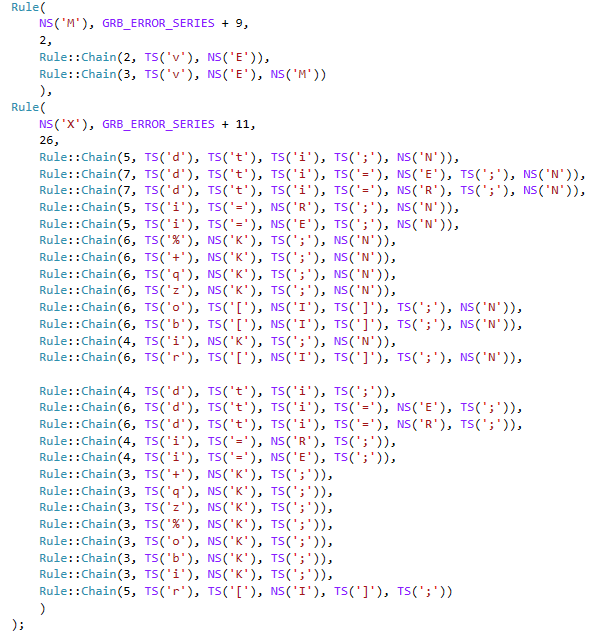
Окончание таблицы лексем

****

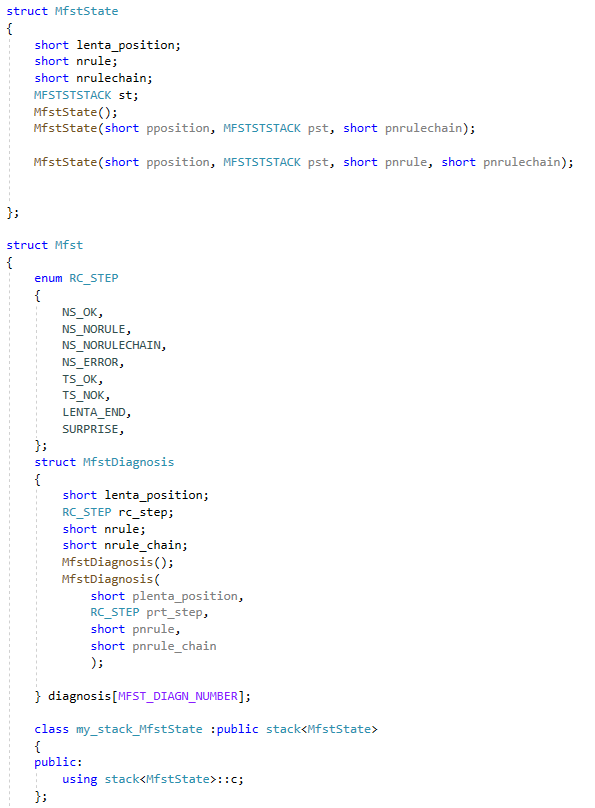
**Приложение Г**

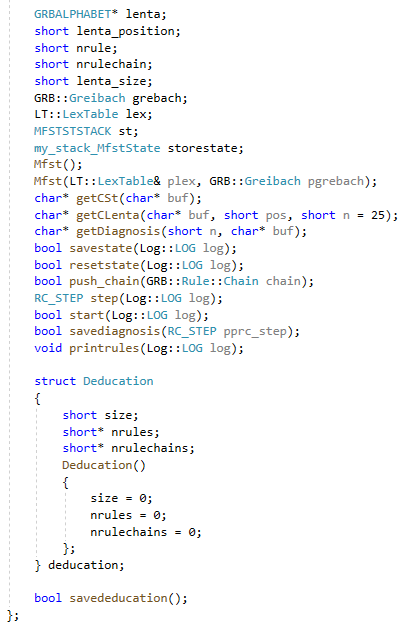
****

****

** **

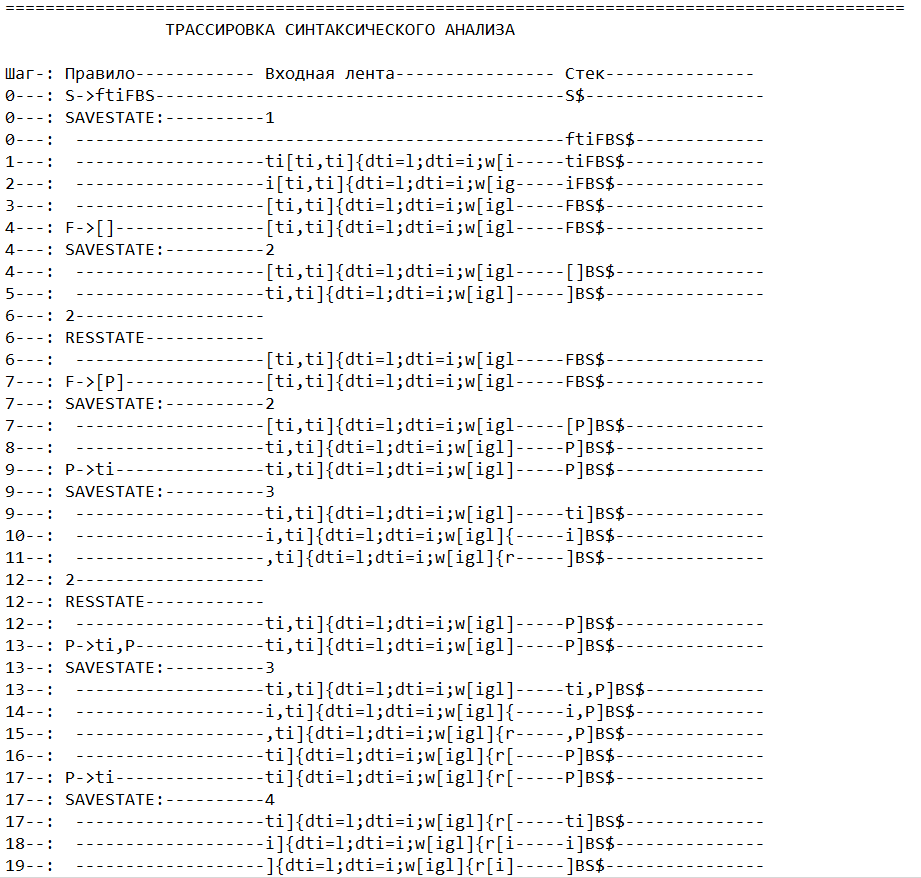
**Приложение Д**

****

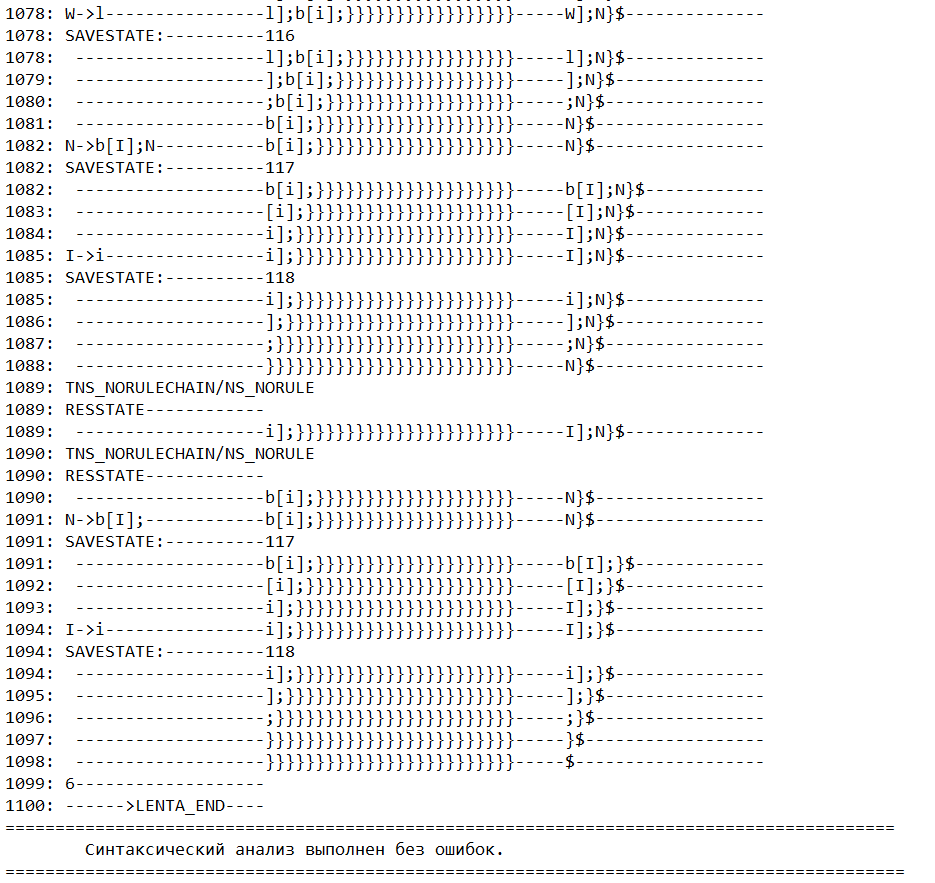
****

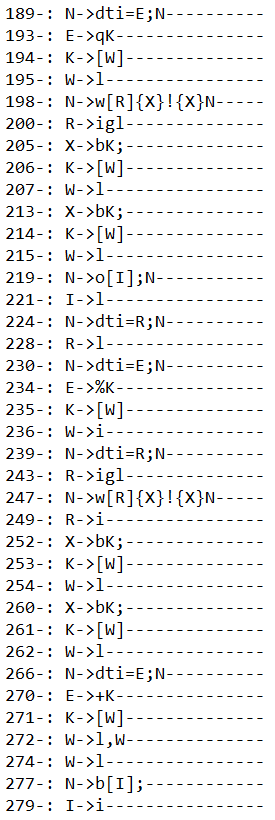
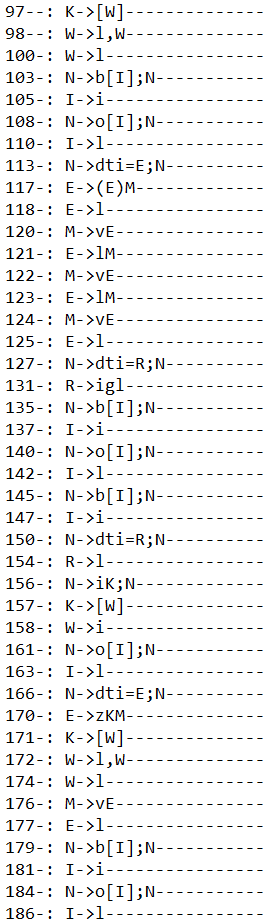
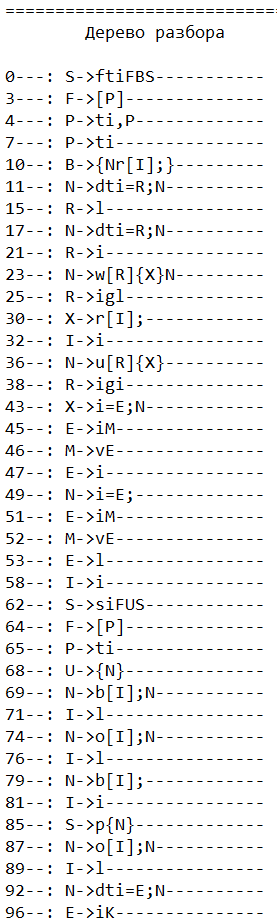
**Приложение Е**

Начало разбора

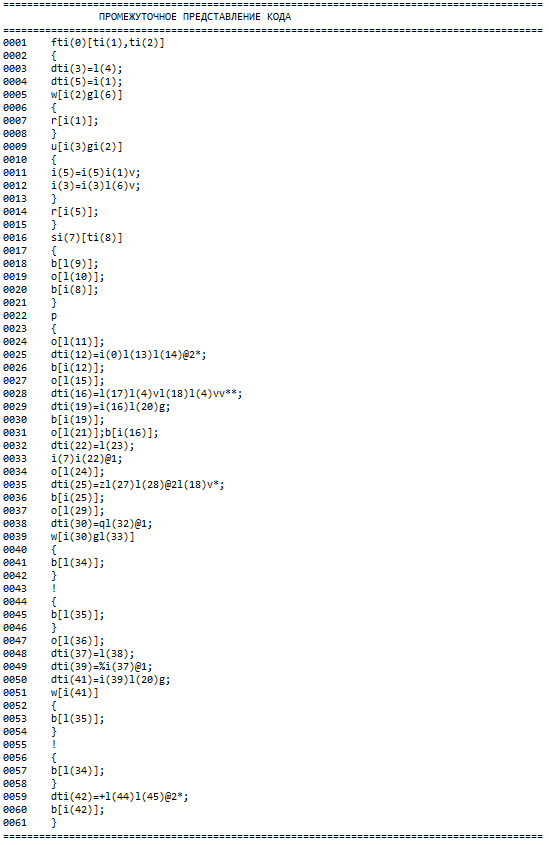


Конец разбора

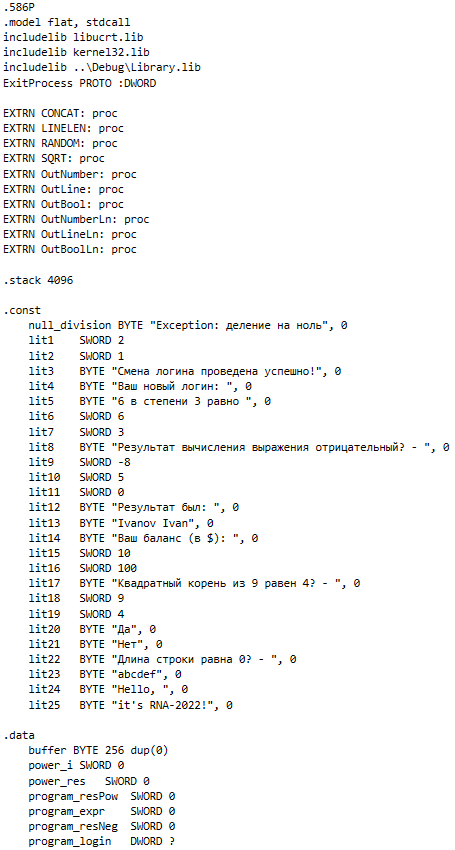


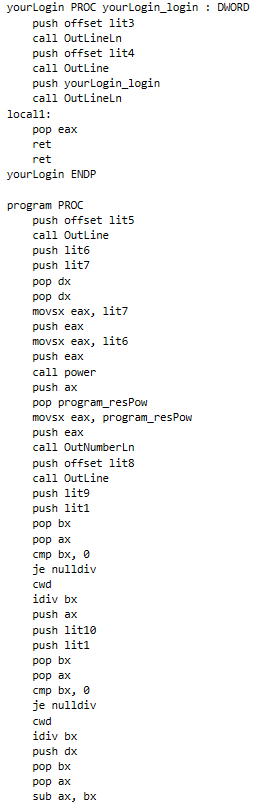
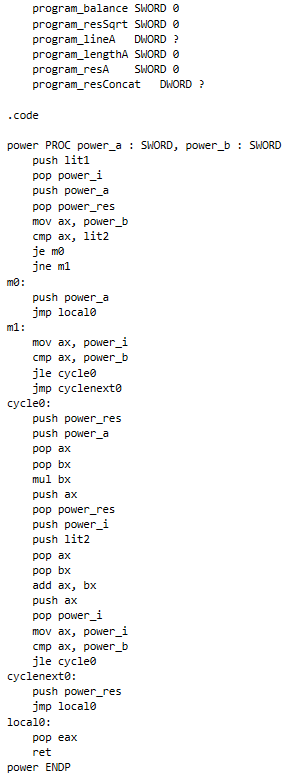
****

**Приложение Ж**

****

**Приложение З**

****

****

