**Содержание**

[Введение 6](#_Toc90849301)

[1. Спецификация языка программирования 7](#_Toc90849302)

[1.1 Характеристика языка программирования 7](#_Toc90849303)

[1.2 Определение алфавита языка программирования 7](#_Toc90849304)

[1.3 Применяемые сепараторы 7](#_Toc90849305)

[1.4 Применяемые кодировки 7](#_Toc90849306)

[1.5 Типы данных 8](#_Toc90849307)

[1.6 Преобразование типов данных 8](#_Toc90849308)

[1.7 Идентификаторы 9](#_Toc90849309)

[1.8 Литералы 9](#_Toc90849310)

[1.9 Объявление данных 9](#_Toc90849311)

[1.10 Инициализация данных 9](#_Toc90849312)

[1.11 Инструкции языка 10](#_Toc90849313)

[1.12 Операции языка 10](#_Toc90849314)

[1.13 Выражения и их вычисление 11](#_Toc90849315)

[1.14 Конструкции языка 11](#_Toc90849316)

[1.15 Область видимости идентификаторов 12](#_Toc90849317)

[1.16 Семантические проверки 12](#_Toc90849318)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc90849319)

[1.18 Стандартная библиотека и ее состав 12](#_Toc90849320)

[1.19 Ввод и вывод данных 13](#_Toc90849321)

[1.20 Точка входа 13](#_Toc90849322)

[1.21 Препроцессор 13](#_Toc90849323)

[1.22 Соглашение о вызовах 13](#_Toc90849324)

[1.23 Объектный код 13](#_Toc90849325)

[1.24 Классификация сообщений транслятора 13](#_Toc90849326)

[2. Структура транслятора 15](#_Toc90849327)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc90849328)

[2.2 Перечень входных параметров транслятора 16](#_Toc90849329)

[2.3 Протоколы, формируемые транслятором 16](#_Toc90849330)

[3. Разработка лексического анализатора 17](#_Toc90849331)

[3.1 Структура лексического анализатора 17](#_Toc90849332)

[3.2 Контроль входных символов 17](#_Toc90849333)

[3.3 Удаление избыточных символов 18](#_Toc90849334)

[3.4 Перечень ключевых слов 18](#_Toc90849335)

[3.5 Основные структуры данных 19](#_Toc90849336)

[3.6 Структура и перечень сообщений лексического анализатора 19](#_Toc90849337)

[3.7 Принцип обработки ошибок 20](#_Toc90849338)

[3.8 Параметры лексического анализатора 20](#_Toc90849339)

[3.9 Алгоритм лексического анализа 20](#_Toc90849340)

[3.10 Контрольный пример 20](#_Toc90849341)

[4. Разработка синтаксического анализатора 21](#_Toc90849342)

[4.1 Структура синтаксического анализатора 21](#_Toc90849343)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 21](#_Toc90849344)

[4.3 Построение конечного магазинного автомата 22](#_Toc90849345)

[4.4 Основные структуры данных 23](#_Toc90849346)

[4.5 Описание алгоритма синтаксического разбора 23](#_Toc90849347)

[4.6 Структура и перечень сообщений синтаксического анализатора 24](#_Toc90849348)

[4.7 Параметры синтаксического анализатора и режимы его работы 24](#_Toc90849349)

[4.8 Принцип обработки ошибок 24](#_Toc90849350)

[4.9 Контрольный пример 24](#_Toc90849351)

[5. Разработка семантического анализатора 25](#_Toc90849352)

[5.1 Структура семантического анализатора 25](#_Toc90849353)

[5.2 Функции семантического анализатора 25](#_Toc90849354)

[5.3 Структура и перечень сообщений семантического анализатора 25](#_Toc90849355)

[5.4 Принцип обработки ошибок 26](#_Toc90849356)

[5.5 Контрольный пример 26](#_Toc90849357)

[6. Вычисление выражений 27](#_Toc90849358)

[6.1 Выражения, допускаемые языком 27](#_Toc90849359)

[6.2 Польская запись и принцип её построения 27](#_Toc90849360)

[7. Генерация кода 28](#_Toc90849361)

[7.1 Структура генератора кода 28](#_Toc90849362)

[7.2 Представление типов данных в оперативной памяти 28](#_Toc90849363)

[7.3 Статическая библиотека 29](#_Toc90849364)

[7.4 Особенности алгоритма генерации кода 29](#_Toc90849365)

[7.5 Входные параметры генератора кода 29](#_Toc90849366)

[7.6 Контрольный пример 29](#_Toc90849367)

[8. Тестирование транслятора 31](#_Toc90849368)

[8.1 Общие положения 31](#_Toc90849369)

[8.2 Результаты тестирования 31](#_Toc90849370)

[Список использованных источников 37](#_Toc90849371)

[Приложение А 38](#_Toc90849372)

[Приложение Б 40](#_Toc90849373)

[Приложение В 45](#_Toc90849374)

[Приложение Г 47](#_Toc90849375)

[Приложение Д 48](#_Toc90849376)

[Приложение Е 50](#_Toc90849377)

[Приложение Ж 52](#_Toc90849378)

# **Введение**

Целью курсового проекта является разработка транслятора для языка программирования RAA-2022. Данный язык программирования предназначен для выполнения сравнений беззнаковых чисел и строк.

Исходя из цели курсового проекта, можно определить следующие задачи:

* Разработка спецификации языка программирования RAA-2022;
* Разработка лексического анализатора;
* Разработка синтаксического анализатора;
* Разработка семантического анализатора;
* Разработка генератора кода;
* Тестирование транслятора.

Пояснительная записка описывает правила и требования в использовании, принцип работы, реализацию разработанного языка программирования и компилятора.

# **1. Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

Язык программирования RAA-2022 предназначен для выполнения сравнения целых чисел и операций над строками, также операций сложения и вычитания двух чисел и возведение в степень. Является процедурным, строго типизированным, не объектно-ориентированным, компилируемым.

**1.2 Определение алфавита языка программирования**

Исходный код может содержать в себе символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, символы пробела, табуляции, перевода строки, спецсимволы [], символы операторов: “< > = !+-” и символы сепараторов: , ; {}().

**1.3 Применяемые сепараторы**

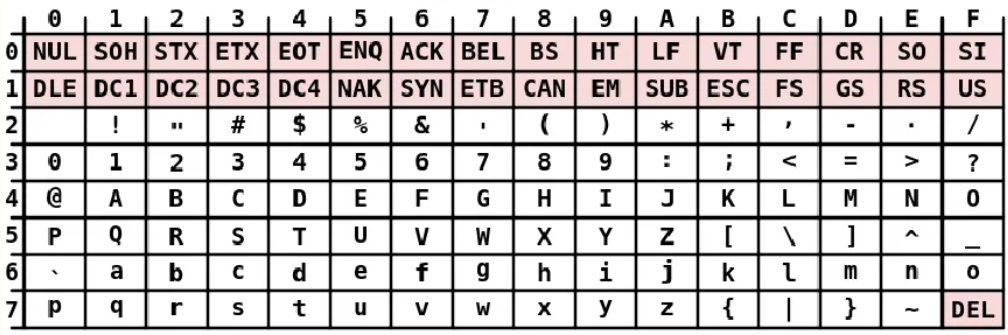
Применяемые сепараторы в языке программирования RAA-2022, приведены в таблице 1.1.

Таблица 1.1 — Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение сепаратора |
| ; | Разделитель инструкций |
| { } | Программный блок |
| ( ) | Параметры |
| пробел | Служит для разделения. Допускается везде, кроме идентификаторов и ключевых слов |
| , | Разделитель параметров в функции |

## **1.4 Применяемые кодировки**

Для написания исходного кода на языке программирования RAA-2022 используется кодировка ASCII, которая представлена на рисунке 1.1.

Рисунок 1.1 – Таблица кодировки ASCII

## **1.5 Типы данных**

В языке программирования RAA-2022 есть 2 типа данных: целочисленный (ushort) и строковый (line). Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2. Пользовательские типы данных не поддерживаются.

Таблица 1.2 – Типы данных языка RAA-2022

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| ushort | Фундаментальный тип данных. Предусмотрен для объявления беззнаковых целочисленных данных (2 байта).  Автоматически инициализируется нулевым значением.  Операции:  <= бинарная операция сравнения меньше либо равно;  >= бинарная операция сравнения больше либо равно;  == бинарная операция сравнения на равенство;  != бинарная операция сравнения на не равенство;  < бинарная операция сравнения меньше;  > бинарная операция сравнения больше;  + арифметическая операция сложения;  - арифметическая операция вычитания; |
| line | Является строковым типом данных. Предназначен для работы с символами, каждый символ занимает 1 байт. Максимальное количество символов – 255, включая символ окончания строки.  Инициализация по умолчанию: нулевой символ. |

## **1.6 Преобразование типов данных**

Язык программирования RAA-2022 строго типизированный, преобразование типов данных не поддерживается.

## **1.7 Идентификаторы**

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Идентификаторы должны начинаться только с символов латинского алфавита, могут содержать цифры. Максимальная длина идентификатора равна 8 символам. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Префикс занимает 8 дополнительных символов. В случае превышения заданной длины, идентификаторы усекаются до длины, равной 16 символов (8 символов на имя идентификатора, 8 символов на префикс). Данные правила действуют для всех типов идентификаторов.

Идентификатор строится по данным правилам:

<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

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<идентификатор> ::= <буква>{ (<цифра> |<буква> ) }

## **1.8 Литералы**

В языке RAA-2022 существует 2 типа литералов: целого и символьного типов. Их краткое описание литералов представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные беззнаковые литералы, двоичное и восьмеричное представления. Литералы только rvalue. |
| Строковые литералы | Состоит из символов латинского алфавита, заключенных в "…" (двойные кавычки). Только rvalue. |

## **1.9 Объявление данных**

В языке RAA-2022 объявление данных начинается с ключевого слова var, указывается тип данных и имя идентификатора. Требуется обязательное объявление переменной перед её использованием.

Примеры: var ushort a, var line b;

Все переменные в языке RAA-2022 имеют область видимости, а именно префикс — название функции, в которой они находятся, что разрешает использование в различных функциях переменных с одинаковым именем. Параметры функции видны только внутри неё. Переменные, объявленные в одной функции, недоступны в другой.

## **1.10 Инициализация данных**

В момент объявления переменных происходит автоматическая инициализация в зависимости от типа данных. Инициализация другими значениями в момент объявления не допускается. Виды инициализации представлены в таблице 1.4.

Таблица 1.4 — Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| <идентификатор> = <значение>; | Присваивание переменной значения. |

## **1.11 Инструкции языка**

Инструкции языка RAA-2022 представлена в таблице 1.5.

Таблица 1.5 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление переменной | var <тип данных> <идентификатор> |
| Объявление внешней функции | <тип данных> function <идентификатор> ({<тип данных> <идентификатор>{,}})  {  < программный блок>  return <идентификатор>|<литерал>.  } |
| Вызов функции | <идентификатор> (<идентификатор>, …) |
| Присвоение значения | <идентификатор> = <значение>; |
| Печать данных | print <литерал>|<идентификатор> |
| Функции стандартной библиотеки  (применяются при инициализации и в выражениях) | compare(line, line) — лексикографическое сравнение строк  pow(ushort, ushort) — возводит первый операнд типа ushort в степень, равную значению второго операнда типа ushort. (перенести) |
| Возвращаемое значение | return <литерал>|<идентификатор> |
| Оператор цикла | cycle(<идентификатор>|<литерал>) {…} |

## **1.12 Операции языка**

Операции сравнения, используются в условной конструкции, которые можно использовать в языке RAA-2022, представлены в таблице 1.6.

Таблица 1.6 — Операции языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Операции сравнения языка | > — бинарное больше  < — бинарное меньше  >= — бинарное больше либо равно |

|  |  |
| --- | --- |
| Продолжение таблицы 1.6 | |
| Инструкция | Форма записи |
| Операции сравнения языка | <= — бинарное меньше либо равно  == — бинарное равно  != — бинарное не равно |
| Арифметические операции | + — бинарный плюс  - — бинарный минус |

## **1.13 Выражения и их вычисление**

В языке присутствуют выражения сравнения, использующиеся в условной конструкции, и арифметические операции. В выражении должны участвовать операторы и операнды одного типа. Не допускается запись двух подряд арифметических операций.

## **1.14 Конструкции языка**

Основные программные конструкции языка программирования RAA-2022 представлены в таблице 1.7.

Таблица 1.7 — Основные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | main  {  <программный блок>  } |
| Функции | <тип данных> function <идентификатор>  (<тип данных> <идентификатор>, …)  {  <программный блок>  return <идентификатор>|<литерал>.  } |
| Условный оператор | if(<литерал>|<идентификатор><логический оператор><литерал>|<идентификатор>)  {  <программный блок>  }  else  {  <программный блок>  } |

## **1.15 Область видимости идентификаторов**

Область видимости реализована по принципу «сверху вниз». В RAA-2022 требуется объявление переменной перед её использованием. Все переменные должны находиться внутри программного блока языка. Имеется возможность объявления одинаковых переменных в разных функциях. Каждая переменная получает префикс – название функции, в которой она объявлена.

## **1.16 Семантические проверки**

В языке программирования RAA-2022 выполняются следующие семантические проверки:

1. Наличие функции main – точки входа в программу;

2. Единственность точки входа;

3. Переопределение идентификаторов;

4. Использование идентификаторов без их объявления;

5. Проверка соответствия типа функции и возвращаемого параметра;

6. Правильность передаваемых в функцию параметров: количество, типы;

7. Правильность строковых выражений;

8. Превышение размера строковых и числовых литералов;

9. Правильность составленного условия цикла/условного оператора.

10.Недопустимость некоторых операций для определённых типов

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

Транслированный 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 определяется за счет использования префиксов.

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

В языке программирования RAA-2022 предусмотрена стандартная библиотека. Функции, входящие в состав библиотеки, описаны в табл. 1.8. Стандартная библиотека подключается автоматически на этапе генерации кода.

Таблица 1.8 - Функции стандартной библиотеки языка RAA-2022

|  |  |
| --- | --- |
| Функция | Описание |
| compare(line a, line b) | Данная функция целочисленного типа, принимает два строковых параметра. Сравнивает строки и возвращает 1, если равны, 0, если a меньше b, 2, если a больше b |
| pow(ushort a, ushort b) | Данная функция целочисленного типа принимает два целочисленных параметра. Функция возводит число a в степень b и возвращает результат. |

## **1.19 Ввод и вывод данных**

Ввод данных языком программирования RAA-2022 не поддерживается.

Для вывода данных используется функция print(<имя идентификатора>|<литерал>). Пример: print(a);

Пользовательская команда print в транслированном коде будут заменена вызовом нужных библиотечных функций.

## **1.20 Точка входа**

Функция точки входа в языке программирования RAA-2022 представлена в таблице 1.9.

Таблица 1.9 — Точка входа

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

## **1.21 Препроцессор**

В языке программирования RAA-2022 препроцессор отсутствует.

## **1.22 Соглашение 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**

RAA-2022 транслируется в язык ассемблера.

## **1.24 Классификация сообщений транслятора**

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы. Сообщения транслятора приведены в таблице 1.10, а также в приложении Б.

Таблица 1.10 Классификация ошибок(диапазон)

|  |  |
| --- | --- |
| Префикс ошибки | Описание ошибки |
| Ошибка ###: [Системная]: | ### - код ошибки. Сообщение, генерируемое при критической ошибке. Диапазон: 0-99 |
| Ошибка ###: [Лексическая]: | ### - код ошибки. Сообщение, генерируемое на этапе лексического анализа. Диапазон: 200-299 |
| Ошибка ###: [Семантическая]: | ### - код ошибки. Сообщение, генерируемое на этапе семантического анализа. Диапазон: 500-599 |
| Ошибка ###: [Синтаксическая]: | ### - код ошибки. Сообщение, генерируемое на этапе синтаксического анализа. Диапазон: 600-699 |

**1.25 Контрольный пример**

Контрольный пример представлен в приложении А.

# **2. Структура транслятора**

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

Задачей транслятора является преобразование программы, написанной на языке RAA-2022, в программу на языке ассемблера. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.



Рисунок 2.1 – Структура транслятора

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). Назначением лексического анализатора является нахождение ошибок лексики языка и формирование таблицы лексем и таблицы идентификаторов. Подробнее описан в 3 главе.

Синтаксический анализ – часть компилятора, выполняющая синтаксический анализ, то есть проверку исходного кода на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией является дерево разбора.

Семантический анализ в свою очередь является проверкой исходной программы на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики. Подробное описание представлено в 5 главе.

Генератор кода – часть транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке RAA-2022, прошедший все предыдущие этапы, в код на языке Ассемблера. Более полно описан в главе 7.

## **2.2 Перечень входных параметров транслятора**

Для формирования файлов с результатами работы лексического, синтаксического и семантического анализаторов используются входные параметры транслятора, представленные в таблице 2.1.

Таблица 2.1 Входные параметры транслятора языка RAA-2022

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке RAA-2022, имеющий расширение .txt | Не предусмотрено |
| -log:<путь к log-файлу> | Файл журнала для вывода протоколов работы программы. | Значение по умолчанию:  <имя in-файла>.log |
| -out:<путь к out-файлу> | Выходной файл – результат работы транслятора. Содержит исходный код на языке асемблера. | Значение по умолчанию:  <имя in-файла>.asm |

## **2.3 Протоколы, формируемые транслятором**

Таблица с перечнем протоколов, формируемых транслятором RAA-2022 и их назначением представлена в таблице 2.2

Таблица 2.2 Протоколы, формируемые транслятором языка RAA-2022

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования RAA-2022. Содержит таблицу лексем и таблицу идентификаторов |
| Выходной файл, заданный параметром "-out:" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

# **3. Разработка лексического анализатор****а**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация.

Функции лексического анализатора:

− удаление «пустых» символов.

− распознавание идентификаторов и ключевых слов;

− распознавание констант;

− распознавание разделителей и знаков операций.

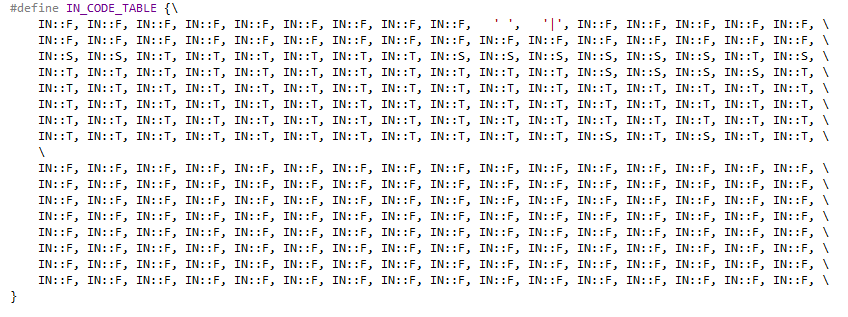
На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1



Рисунок 3.1 Структура лексического анализатора RAA-2022

## **3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2

Рисунок 3.2. Таблица контроля входных символов

Принцип работы таблицы заключается в сопоставлении кода ASCII каждого входного символа значению в таблице.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, S – сепаратор.

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы. Табуляция заменяется на пробел. Пробелы игнорируются на этапе разбиения исходного кода на слова.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом программы.

2. Пробелы выступаю в роли сепаратора и пропускаются.

## **3.4 Перечень ключевых слов**

Лексемы – это символы, соответствующие ключевым словам, символам операций и сепараторам, необходимые для упрощения дальнейшей обработки исходного кода программы. Данное соответствие описано в таблице 3.1.

Таблица 3.1 Соответствие ключевых слов, символов операций и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Тип | Слова | Лексема |
| Ключевые слова | var | d |
| ushort | t |
| line | t |
| function | f |
| print | p |
| pow | b |
| compare | a |
| return | r |
| main | m |
| Продолжение таблицы 3.1 | | |
| Тип | Слова | Лексема |
| Ключевые слова | if | c |
| else | e |
| cycle | y |
| uselib | u |
| Иное | Идентификатор | i |
| Целочисленный литерал | l |
| Строковый литерал | L |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| Операторы | = | = |
| + | + |
| - | - |
| Логические операторы | o |

Пример реализации таблицы лексем представлен в приложении Б.

Также в приложении Б находятся некоторые конечные автоматы, соответствующие лексемам языка RAA-2022.

## **3.5 Основные структуры данных**

Структуры таблиц лексем и идентификаторов данных языка RAA-2022, используемых для хранения, представлены в приложении В. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, смысловой тип идентификатора и его значение.

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

Перечень сообщений лексического анализатора представлен на рисунке 3.3.

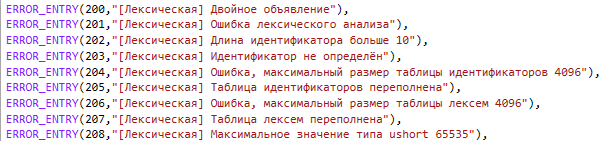


Рисунок 3.3 – Перечень ошибок лексического анализатора

## **3.7 Принцип обработки ошибок**

При возникновении ошибки транслятор завершает свою работу. 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.

## **3.8 Параметры лексического анализатора**

Входным параметром лексического анализа является структура, полученная после чтения входного файла на этапе проверки исходного кода на допустимость символов.

## **3.9 Алгоритм лексического анализа**

Алгоритм работы лексического анализа:

* проверяет входной поток символов программы на исходном языке на допустимость, удаляет лишние пробелы и добавляет сепаратор для вычисления номера строки для каждой лексемы;
* для выделенной части входного потока выполняется функция распознавания лексемы;
* при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;
* формирует протокол работы;
* при неуспешном распознавании выдается сообщение об ошибке.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов.

Пример. Регулярное выражение для ключевого слова «main».

Граф конечного автомата для этой лексемы представлен на рисунке 3.4. S0 – начальное состояние, S4 – конечное состояние автомата.

n

m

a

i

Рисунок 3.4 – Граф переходов для цепочки ‘main’

## **3.10 Контрольный пример**

Результат работы лексического анализатора – таблицы лексем и идентификаторов – представлен в приложении Б.

# **4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ – это фаза компилятора, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций. Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа. Выходом – 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 RAA-2022 используется контекстно-свободная грамматика , где

T – множество терминальных символов (алфавит языка RAA-2022),

N – множество нетерминальных символов,

P – множество правил языка,

S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика приведена к нормальной форме Грейбах, 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.

Грамматика языка RAA-2022 представлена в приложении Г.

TS – терминальные символы, которыми являются сепараторы, знаки арифметических операций и некоторые строчные буквы, которые приведены в разделе 1.2

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

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

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | S→m{NrE;};| tfi(F){NrE;};S| tfi( ){NrE;};S| ul | ulS | Порождает правила, описывающее общую структуру программы |
| N | N→dti;|i=E;|dti;N| i=E;N|p(l);N| p(i);N|p(l);| p(i);|  c(C){N}e{N}N| c(C){N}e{N}|  c(C){N}| c(C){N}N | Порождает правила, описывающие инструкции языка |
| E | E→i|l|i(W)|i( )| b(W)|a(W)| i+l|i+i|l+l|i-i|i-l|l-l | Порождает правила, описывающие выражения |
| F | F→ti|ti,F | Порождает правила, описывающие параметры локальной функции при её объявлении |
| W | W→i|l|i,W|l,W | Порождает правила, описывающие параметры вызываемой функции |
| C | С→ioi|iol|loi|lol | Порождает правила, описывающие условия оператора if |

## **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

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | 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 S) |
|  | 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 представлены в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила и цепочки правил. Данные структуры представлены в приложении Г.

## **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 на рисунке 4.1.

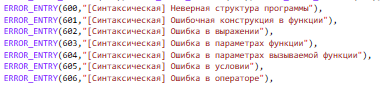


Рисунок 4.1 – Перечень сообщений синтаксического анализатора

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

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

Выходными параметрами являются правила разбора, которые выводятся на консоль.

## **4.8 Принцип обработки ошибок**

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку. Тогда анализ останавливается и выводится сообщение об ошибке, если она найдена. Если в процессе анализа находятся более трёх ошибок, то анализ останавливается. После всей процедуры трассировки в протокол будет выведено диагностическое сообщение.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке RAA-2022 представлен в приложении Е. Дерево разбора исходного кода также представлено в графическом материале.

# **5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ языка RAA-2022 выполняется после выполнения лексического и синтаксического анализа. Несмотря на это, некоторые семантические проверки выполняются на этапе лексического анализа. На вход семантического анализатора подаются таблица лексем и таблица идентификаторов. Схема семантического анализатора изображена на рисунке 5.1.

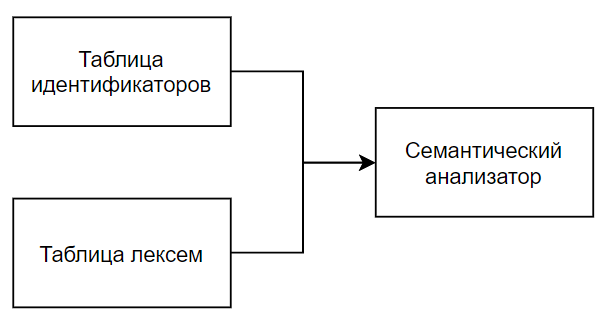


Рисунок 5.1 Схема семантического анализатора языка RAA-2022

## **5.2 Функции семантического анализатора**

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

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

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.1.

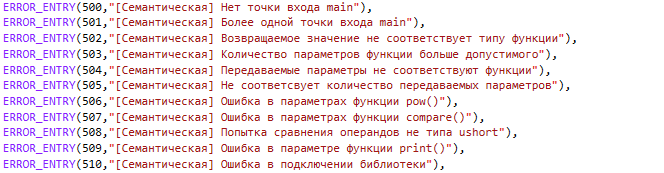


Рисунок 5.1 – Перечень сообщений семантического анализатора

## **5.4 Принцип обработки ошибок**

Принцип обработки ошибок идентичен принципу обработки ошибок на этапе лексического анализа (раздел 3.7).

## **5.5 Контрольный пример**

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

Таблица 5.1. Примеры диагностики ошибок

|  |  |
| --- | --- |
| Текст ошибки | Исходный код |
| Ошибка 501: [Cемантическая]: Более одной точки входа main, строка 0, позиция 0 | main  {  …  }  main  {  …  } |
| Ошибка 502: [Cемантическая]: Возвращаемое значение не соответствует типу функции, строка 2, позиция 0 | line function fun(ushort a){  return a;  } |
| Ошибка 504: [Cемантическая]: Передаваемые параметры не соответствуют функции, строка 7, позиция 0 | line function fun(ushort a){  return a;  }  main  {  var line str;  str = func("b");  return 0;  } |

# **6. Вычисление выражений**

## **6.1 Выражения, допускаемые языком**

В языке RAA-2022 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются логические операции, такие как < > == <= >= != и арифметические +-

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

В языке RAA-2022 польская запись не используется.

# **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.

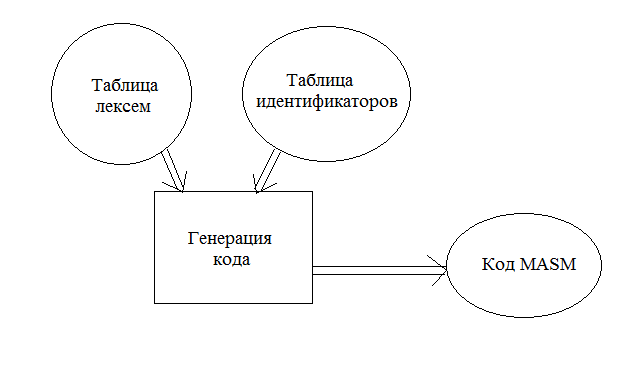


Рисунок 7.1 Структура генератора кода

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

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка RAA-2022 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке RAA-2022 и на языке ассемблера приведены в таблице 7.1.

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

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языкеRAA-2022 | Тип идентификатора на языке ассемблера | Пояснение |
| ushort | WORD | Хранит целочисленный тип данных со знаком. |
| line | DWORD | Хранит указатель на начало строки. |
| L(0-99) | BYTE  WORD | Литералы: символьные,  целочисленные |

## **7.3 Статическая библиотека**

В языке RAA-2022 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера. Объявление функций статической библиотеки генерируется автоматически.

Таблица 7.3 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| \_pow(ushort a, ushort b) | Возводит число a в степень b |
| \_printN(ushort a)  \_printS(line str) | Вывод на консоль целочисленной переменной a и строки str |
| \_compare(line a, line b) | Лексикографическое сравнение строк, если строка a меньше b возвращает 0, равна 1, больше 2 |

## **7.4 Особенности алгоритма генерации кода**

В языке RAA-2022 генерация кода строится на основе таблиц лексем и идентификаторов. Преобразования происходят по мере прохождения по таблицам. Функции статической библиотеки начинаются с нижнего подчёркивания для исключения их переопределения. Перед началом основной трансляции производится запись литералов и идентификаторов в сегменты констант и данных соответственно.

## **7.5 Входные параметры генератора кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке RAA-2022. Результаты работы генератора кода выводятся в файл с расширением .asm

## **7.6 Контрольный пример**

Результат генерации ассемблерного кода на основе контрольного примера приведен в приложении Ж. Результат работы контрольного примера приведён на рисунке 7.1.

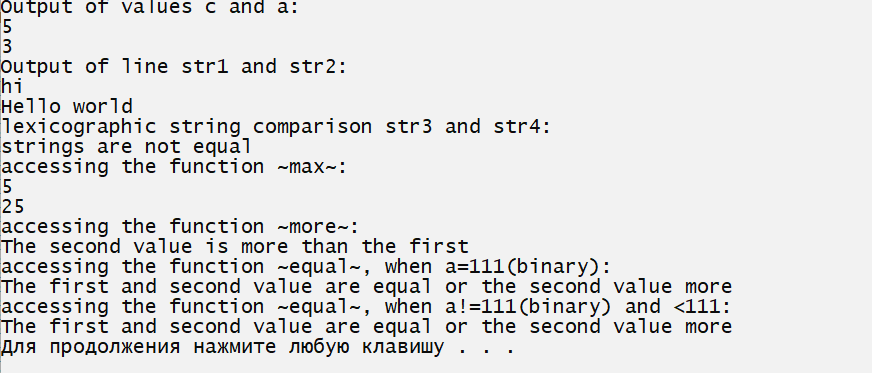


Рисунок 7.1 Результат работы программы на языке RAA-2022

# **8. Тестирование транслятора**

## **8.1 Общие положения**

При возникновении ошибки на каком-либо этапе трансляции, она обрабатывается в главном файле программы: ошибка выводится на консоль и записывается в файл логирования.

## **8.2 Результаты тестирования**

Описание тестовых наборов, демонстрирующих проверки на разных этапах трансляции, приведено в таблице 8.1

Таблица 8.1 – Результаты тестирования транслятора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| э | Ошибка 111: [Системная]: Недопустимый символ в исходном файле (-in), строка 1, позиция 0 |
| main  {  var line str1;  var line str1;  } | Ошибка 200: [Лексическая]: Двойное объявление, строка 4, позиция 0 |
| main  {  A  } | Ошибка 201: [Лексическая]: Ошибка лексического анализа, строка 3, позиция 0 |
| var line a23456789; | Ошибка 202: [Лексическая]: Длина идентификатора больше 8, строка 1, позиция 0 |
| str = ""; | Ошибка 203: [Лексическая]: Идентификатор не определён, строка 1, позиция 0 |
| main  {  var ushort a;  a = 65536;  } | Ошибка 208: [Лексическая]: Максимальное значение типа ushort 65535, строка 4, позиция 0 |
| var ushort a; | Ошибка 500: [Cемантическая]: Нет точки входа main, строка 0, позиция 0 |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  }  main  {  } | Ошибка 501: [Cемантическая]: Более одной точки входа main, строка 0, позиция 0 |
| line function func()  {  var ushort a;  return 0;  }  main  {  var ushort a;  return 0;  } | Ошибка 502: [Cемантическая]: Возвращаемое значение не соответствует типу функции, строка 4, позиция 0 |
| ushort function func(ushort a, ushort b, ushort c, ushort d, ushort e, ushort f)  {  a = 3;  return a;  }  main  {  var ushort a;  a = func(1, 2, 3, 4, 5, 6);  return 0;  } | Ошибка 503: [Cемантическая]: Количество параметров функции больше допустимого, строка 9, позиция 0 |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| ushort function func(ushort a)  {  a = 3;  return a;  }  main  {  var ushort a;  a = func("1");  return 0;  } | Ошибка 504: [Cемантическая]: Передаваемые параметры не соответствуют функции, строка 9, позиция 0 |
| ushort function func(ushort a)  {  a = 3;  return a;  }  main  {  var ushort a;  a = func(1, 2);  return 0;  } | Ошибка 505: [Cемантическая]: Не соответсвует количество передаваемых параметров, строка 9, позиция 0 |
| uselib "../Debug/StaticLib.lib"  main  {  var ushort a;  a = pow(1);  return 0;  } | Ошибка 506: [Cемантическая]: Ошибка в параметрах функции pow(), строка 5, позиция 0 |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| uselib "../../RAAlib/Debug/RAAlib"  main  {  var ushort a;  a = compare(1);  return 0;  } | Ошибка 507: [Cемантическая]: Ошибка в параметрах функции compare(), строка 5, позиция 0 |
| main  {  var ushort a;  var line b;  if(a > b){  print("some");  }  return 0;  } | Ошибка 508: [Cемантическая]: Попытка сравнения операндов не типа ushort, строка 5, позиция 0 |
| uselib 5  main  {  print("a");  return 0;  } | Ошибка 510: [Cемантическая]: Ошибка в подключении библиотеки, строка 1, позиция 0 |
| ushort function func(ushort a)  {  var line;  return 0;  }  main  {  var line a;  return 0;  } | 601: строка 1, [Cинтаксическая] : Ошибочная конструкция в функции |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  var line a;  a = 1, 2;  return 0;  } | 602: строка 4, [Cинтаксическая]: Ошибка в выражении |
| ushort function func(ushort a)  {  a = 0;  return a;  }  main  {  var ushort a;  a = func(,);  return 0;  } | 604: строка 9, [Cинтаксическая]: Ошибка в параметрах вызываемой функции |
| ushort function func(,)  {  a = 0;  return a;  }  main  {  var ushort a;  a = func(1);  return 0;  } | 603: строка 1, [Cинтаксическая]: Ошибка в параметрах функции |
| main  {  if("a")  {  print("a");  }  return 0;  } | 605: строка 3, [Cинтаксическая]: Ошибка в условии |

**Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования RAA-2022. Таким образом, были выполнены основные задачи данной курсовой работы:

* Представлена спецификация языка RAA-2022;
* Разработан лексический анализатор;
* Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика;
* Разработан синтаксический анализатор;
* Разработан семантический анализатор;
* Разработан транслятор кода на язык ассемблера;
* Проведено тестирование транслятора.

Окончательная версия языка RAA-2022 включает:

* 2 типа данных;
* Возможность вызова функций стандартной библиотеки;
* Наличие 6 логических операторов и 2 арифметических операций;
* Поддержку функций, оператора условия и цикла;

# **Список использованных источников**

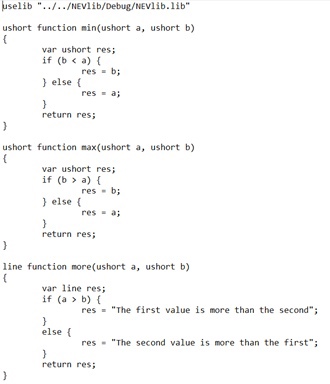
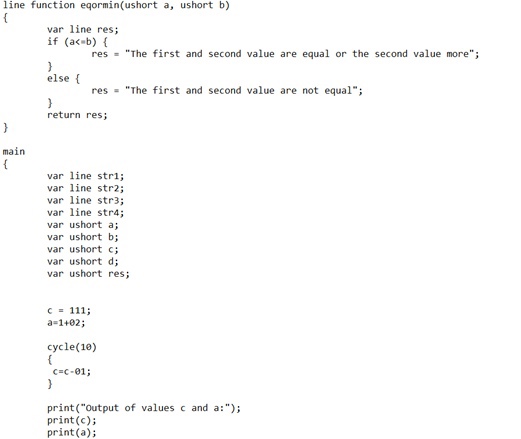
1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.

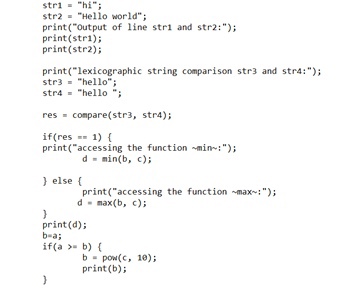
2. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.

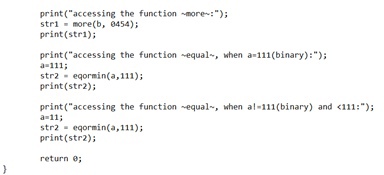
3. Ирвин, К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.

# **Приложение А**

Контрольный пример

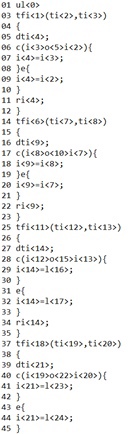
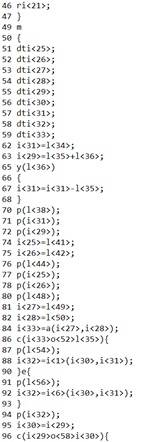
  


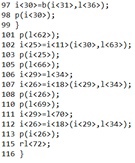


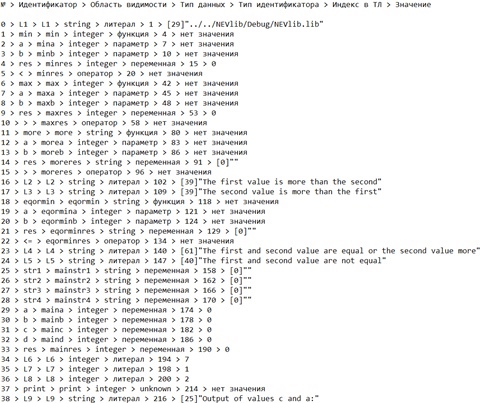
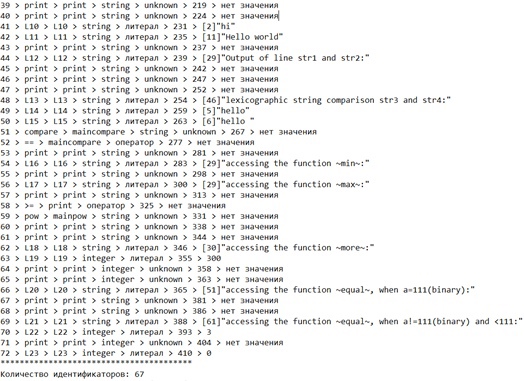


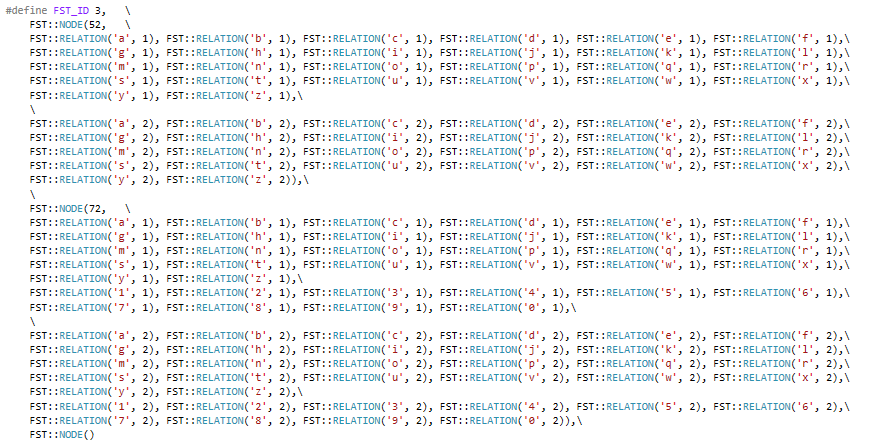
# **Приложение Б**

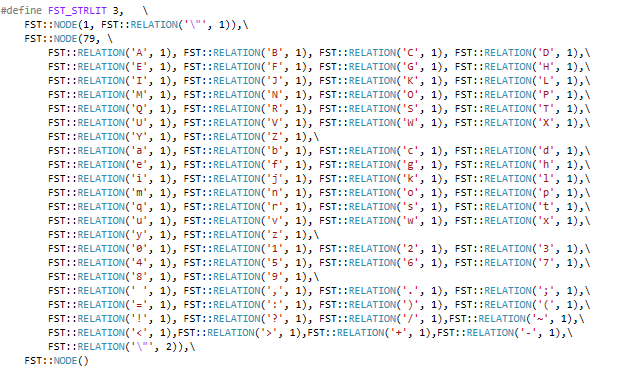
Правила разбора для лексического анализатора и результат его работы

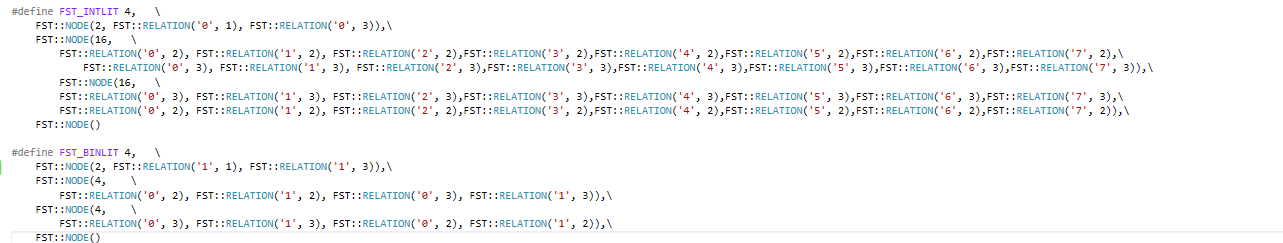
  


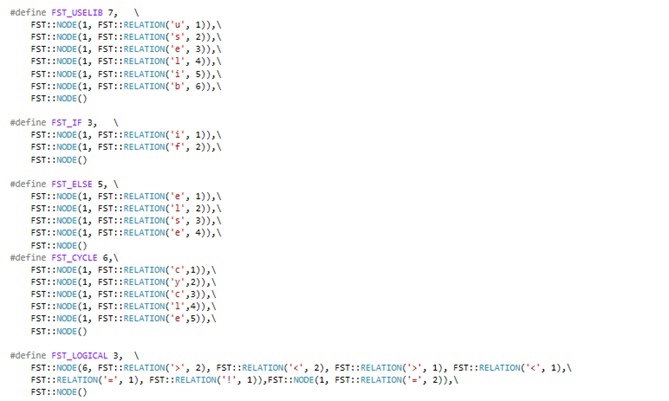


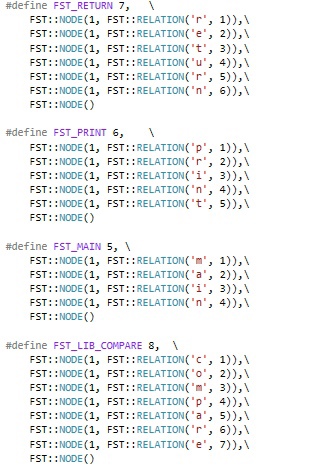
  


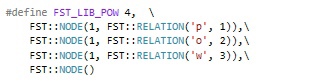


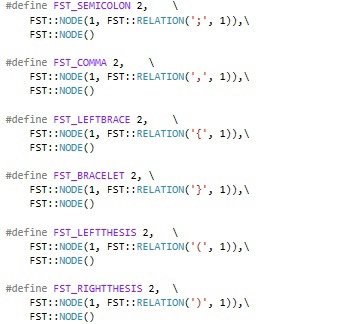


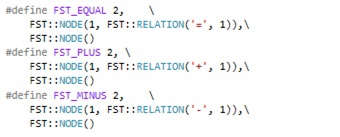






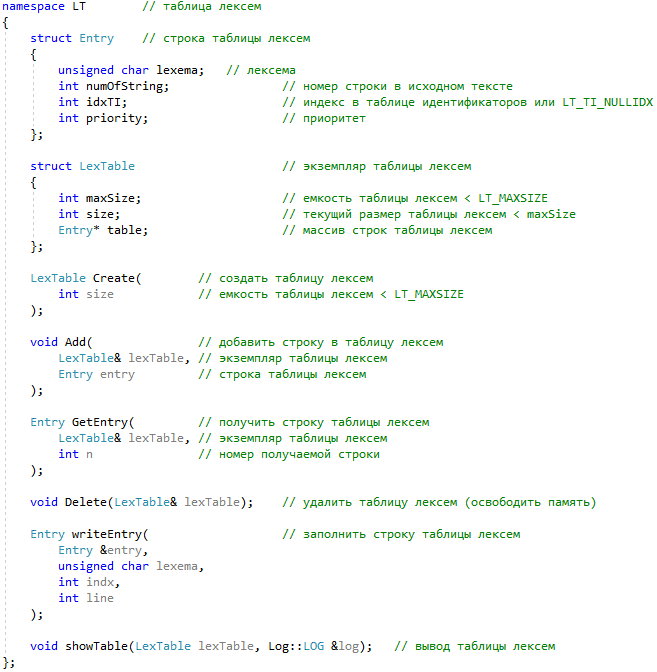


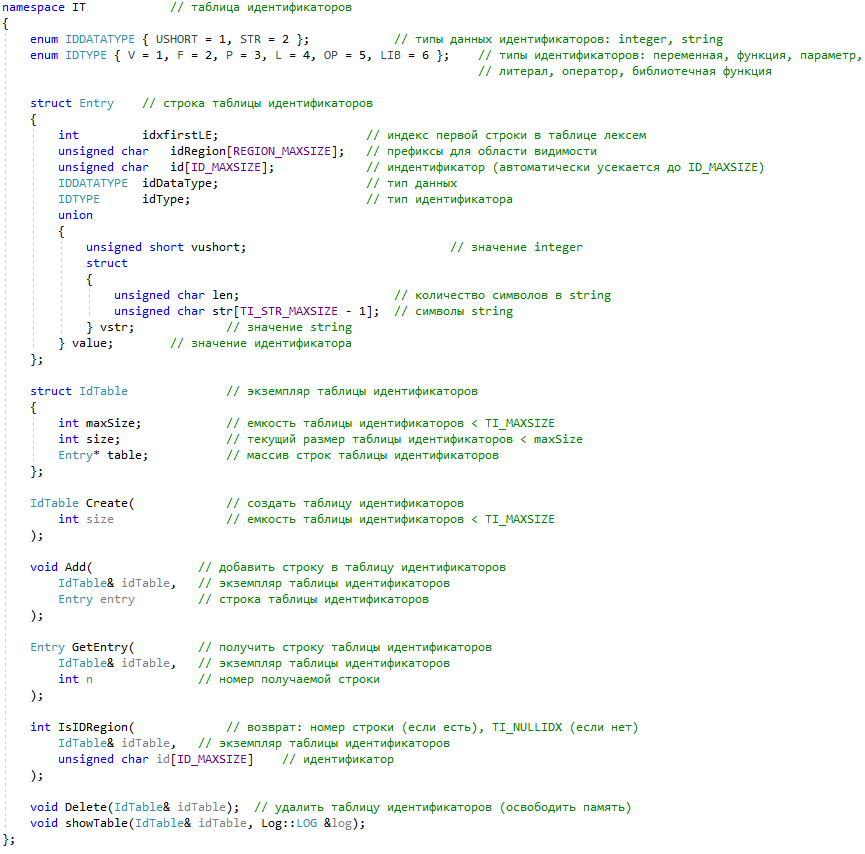




# **Приложение В**

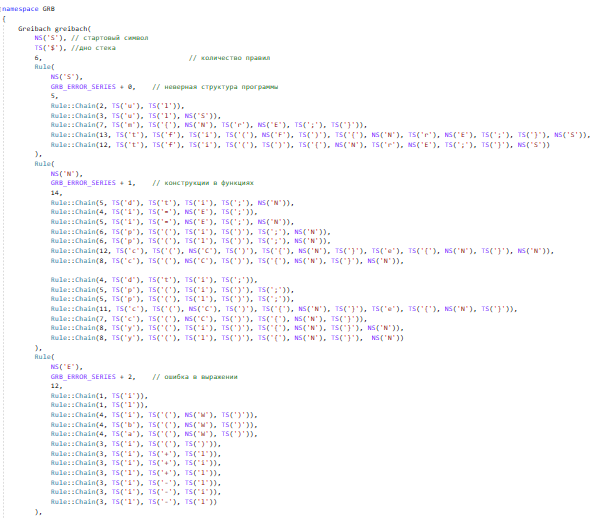
Основные структуры таблиц лексем и идентификаторов данных

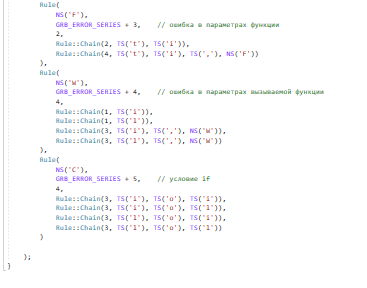




# **Приложение Г**

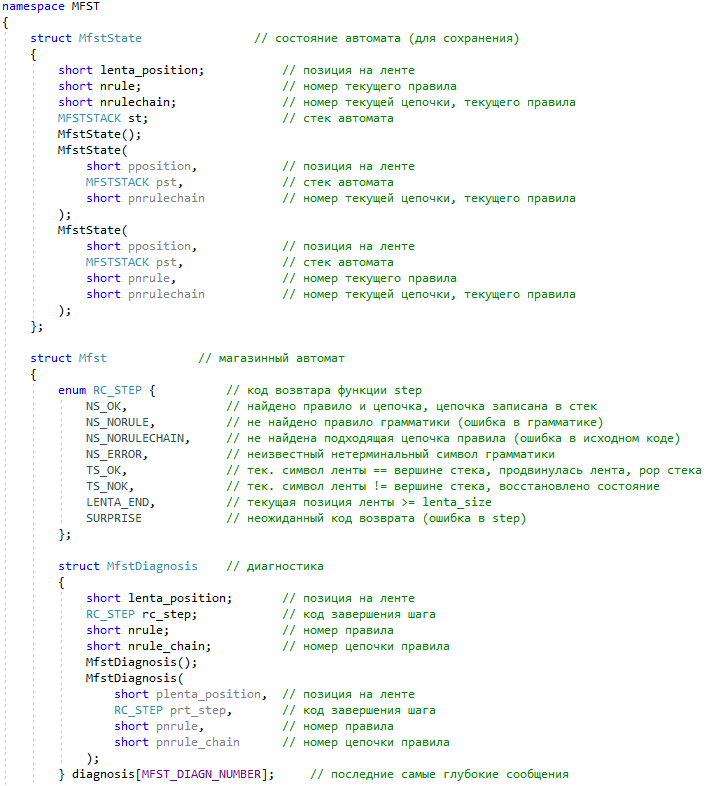
Грамматика языка

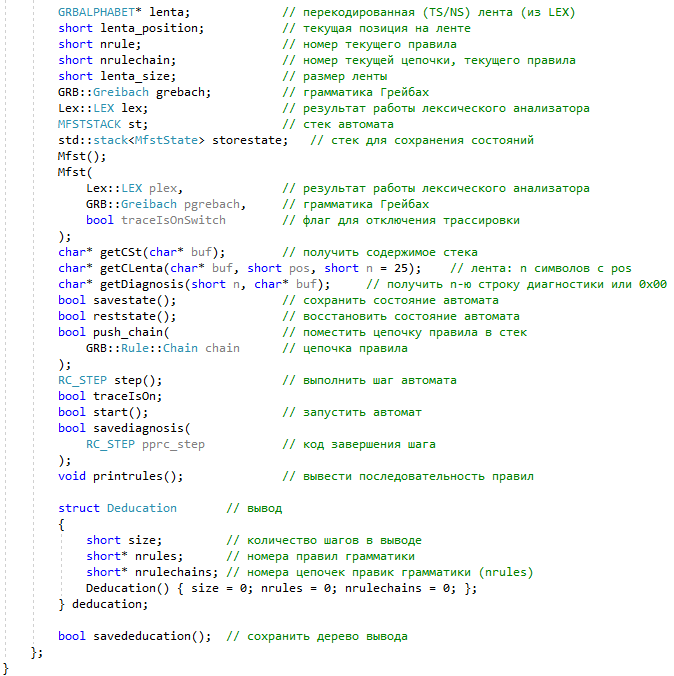




# **Приложение Д**

Структура магазинного автомата и грамматики Грейбах

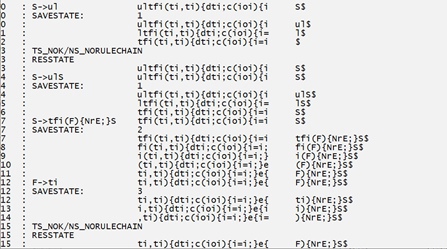




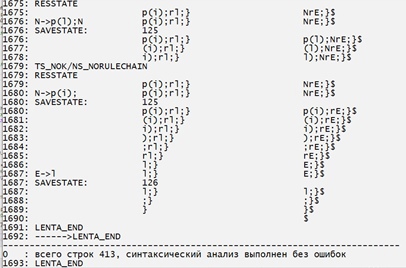
# **Приложение Е**

Результат синтаксического анализа

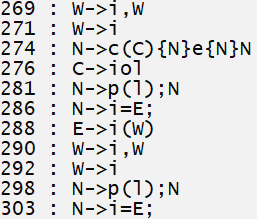
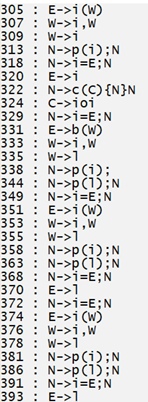
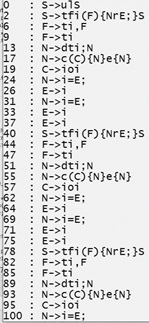
Начало разбора:

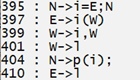
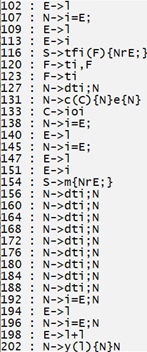
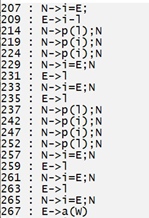


Конец разбора:



Правила, использованные при разборе:

****



# **Приложение Ж**

Сгенерированный код ассемблера

.586

.model flat, stdcall

includelib libucrt.lib

includelib kernel32.lib

includelib C:/Users/rubas/Desktop/RAA-2022-main/RAAlib/Debug/RAAlib

EXTERN \_printS :PROC

EXTERN \_printN :PROC

EXTERN \_pow :PROC

EXTERN \_compare :PROC

EXTERN \_pause :PROC

ExitProcess PROTO :DWORD

.stack 4096

.const

L1 BYTE 'C:/Users/rubas/Desktop/RAA-2022-main/RAAlib/Debug/RAAlib', 0

L2 BYTE 'The first value is more than the second', 0

L3 BYTE 'The second value is more than the first', 0

L4 BYTE 'The first and second value are equal or the second value more', 0

L5 BYTE 'The first and second value are not equal', 0

L6 WORD 1

L7 WORD 2

L8 BYTE 'Output of values c and a:', 0

L9 BYTE 'hi', 0

L10 BYTE 'Hello world', 0

L11 BYTE 'Output of line str1 and str2:', 0

L12 BYTE 'lexicographic string comparison str3 and str4:', 0

L13 BYTE 'hello', 0

L14 BYTE 'hello ', 0

L15 BYTE 'accessing the function ~min~:', 0

L16 BYTE 'accessing the function ~max~:', 0

L17 BYTE 'accessing the function ~more~:', 0

L18 WORD 300

L19 BYTE 'accessing the function ~equal~, when a=111(binary):', 0

L20 WORD 7

L21 BYTE 'accessing the function ~equal~, when a!=111(binary) and <111:', 0

L22 WORD 3

L23 WORD 0

.data

minres WORD 0

maxres WORD 0

moreres DWORD 0

eqorminres DWORD 0

mainstr1 DWORD 0

mainstr2 DWORD 0

mainstr3 DWORD 0

mainstr4 DWORD 0

maina WORD 0

mainb WORD 0

mainc WORD 0

maind WORD 0

mainres WORD 0

.code

min PROC mina : WORD, minb : WORD

mov ax, minb

cmp ax, mina

jl p0

jg p1

je p1

p0:

push minb

pop minres

jmp ife0

p1:

push mina

pop minres

ife0:

push minres

pop eax

ret

min ENDP

max PROC maxa : WORD, maxb : WORD

mov ax, maxb

cmp ax, maxa

jg p2

jl p3

je p3

p2:

push maxb

pop maxres

jmp ife1

p3:

push maxa

pop maxres

ife1:

push maxres

pop eax

ret

max ENDP

more PROC morea : WORD, moreb : WORD

mov ax, morea

cmp ax, moreb

jg p4

jl p5

je p5

p4:

push offset L2

pop moreres

jmp ife2

p5:

push offset L3

pop moreres

ife2:

push moreres

pop eax

ret

more ENDP

eqormin PROC eqormina : WORD, eqorminb : WORD

mov ax, eqormina

cmp ax, eqorminb

je p6

jl p6

jg p7

p6:

push offset L4

pop eqorminres

jmp ife3

p7:

push offset L5

pop eqorminres

ife3:

push eqorminres

pop eax

ret

eqormin ENDP

main PROC

push L6

pop mainc

push L6

pop eax

push L7

pop ebx

add eax, ebx

push eax

pop maina

movzx ecx, L7

p8:

push mainc

pop ebx

push L6

pop eax

sub ebx, eax

push ebx

pop mainc

loop p8

push offset L8

call \_printS

push mainc

call \_printN

push maina

call \_printN

push offset L9

pop mainstr1

push offset L10

pop mainstr2

push offset L11

call \_printS

push mainstr1

call \_printS

push mainstr2

call \_printS

push offset L12

call \_printS

push offset L13

pop mainstr3

push offset L14

pop mainstr4

push mainstr4

push mainstr3

call \_compare

push eax

pop mainres

mov ax, mainres

cmp ax, L6

je p9

jg p10

jl p10

p9:

push offset L15

call \_printS

movzx eax, mainc

push eax

movzx eax, mainb

push eax

call min

push eax

pop maind

jmp ife4

p10:

push offset L16

call \_printS

movzx eax, mainc

push eax

movzx eax, mainb

push eax

call max

push eax

pop maind

ife4:

push maind

call \_printN

push maina

pop mainb

mov ax, maina

cmp ax, mainb

je p11

jg p11

jl p12

p11:

movzx eax, L7

push eax

movzx eax, mainc

push eax

call \_pow

push eax

pop mainb

push mainb

call \_printN

p12:

push offset L17

call \_printS

movzx eax, L18

push eax

movzx eax, mainb

push eax

call more

push eax

pop mainstr1

push mainstr1

call \_printS

push offset L19

call \_printS

push L20

pop maina

movzx eax, L20

push eax

movzx eax, maina

push eax

call eqormin

push eax

pop mainstr2

push mainstr2

call \_printS

push offset L21

call \_printS

push L22

pop maina

movzx eax, L20

push eax

movzx eax, maina

push eax

call eqormin

push eax

pop mainstr2

push mainstr2

call \_printS

push 0

call \_pause

call ExitProcess

main ENDP

end main