Оглавление

[Введение 4](#_Toc533046684)

[Глава 1. Спецификация языка программирования 5](#_Toc533046685)

[1.1 Характеристика языка программирования 5](#_Toc533046686)

[1.2 Алфавит языка 5](#_Toc533046687)

[1.3 Применяемые сепараторы 5](#_Toc533046688)

[1.4 Применяемые кодировки 6](#_Toc533046689)

[1.5 Типы данных 6](#_Toc533046690)

[1.6 Преобразование типов данных 7](#_Toc533046691)

[1.7 Идентификаторы 7](#_Toc533046692)

[1.8 Литералы 7](#_Toc533046693)

[1.9 Объявление данных 7](#_Toc533046694)

[1.10 Инициализация данных 7](#_Toc533046695)

[1.11 Инструкции языка 8](#_Toc533046696)

[1.12 Операции языка 8](#_Toc533046697)

[1.13 Выражения и их вычисление 8](#_Toc533046698)

[1.14 Конструкции языка 8](#_Toc533046699)

[1.15 Области видимости идентификаторов. 9](#_Toc533046700)

[1.16 Семантические проверки 9](#_Toc533046701)

[1.17 Распределение оперативной памяти на этапе выполнения 9](#_Toc533046702)

[1.18 Стандартная библиотека и её состав 10](#_Toc533046703)

[1.19 Ввод и вывод данных 10](#_Toc533046704)

[1.20 Точка входа 10](#_Toc533046705)

[1.21 Препроцессор 10](#_Toc533046706)

[1.22 Соглашения о вызовах 10](#_Toc533046707)

[1.23 Объектный код 11](#_Toc533046708)

[1.24 Классификация сообщений транслятора 11](#_Toc533046709)

[1.25 Контрольный пример 11](#_Toc533046710)

[Глава 2. Структура транслятора 12](#_Toc533046711)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 12](#_Toc533046712)

[2.2 Перечень входных параметров транслятора 12](#_Toc533046713)

[2.3 Протоколы, формируемые транслятором 12](#_Toc533046714)

[Глава 3 Разработка лексического анализатора 14](#_Toc533046715)

[3.1 Структура лексического анализатора 14](#_Toc533046716)

[3.2 Контроль входных символов 14](#_Toc533046717)

[3.3 Удаление избыточных символов 14](#_Toc533046718)

[3.4 Перечень ключевых слов 15](#_Toc533046719)

[3.5 Основные структуры данных 16](#_Toc533046720)

[3.6 Структура и перечень сообщений лексического анализатора 16](#_Toc533046721)

[3.7 Принцип обработки ошибок 16](#_Toc533046722)

[3.8 Параметры лексического анализатора 16](#_Toc533046723)

[3.9 Алгоритм лексического анализа 16](#_Toc533046724)

[3.10 Контрольный пример 16](#_Toc533046725)

[Глава 4. Разработка синтаксического анализатора 17](#_Toc533046726)

[4.1 Структура синтаксического анализатора 17](#_Toc533046727)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 17](#_Toc533046728)

[4.3 Построение конечного магазинного автомата 18](#_Toc533046729)

[4.4 Основные структуры данных 19](#_Toc533046730)

[4.5 Описание алгоритма синтаксического разбора 19](#_Toc533046731)

[4.6 Структура и перечень сообщений синтаксического анализатора 19](#_Toc533046732)

[4.7 Параметры синтаксического анализатора и режимы его работы 20](#_Toc533046733)

[4.8 Принцип обработки ошибок 20](#_Toc533046734)

[4.9 Контрольный пример 20](#_Toc533046735)

[Глава 5. Разработка семантического анализатора 21](#_Toc533046736)

[5.1 Структура семантического анализатора 21](#_Toc533046737)

[5.2 Функции семантического анализатора 21](#_Toc533046738)

[5.3 Структура и перечень сообщений семантического анализатора 21](#_Toc533046739)

[5.4 Принцип обработки ошибок 22](#_Toc533046740)

[Глава 6. Преобразование выражений 23](#_Toc533046741)

[6.1 Выражения, допускаемые языком 23](#_Toc533046742)

[6.2 Польская запись и принцип ее построения 23](#_Toc533046743)

[6.3 Программная реализация обработки выражений 23](#_Toc533046744)

[6.4 Контрольный пример 23](#_Toc533046745)

[Глава 7. Генерация кода 24](#_Toc533046746)

[7.1 Структура генератора кода 24](#_Toc533046747)

[7.2 Представление типов данных в оперативной памяти 24](#_Toc533046748)

[7.3 Статическая библиотека 25](#_Toc533046749)

[7.4 Особенности алгоритма генерации кода 25](#_Toc533046750)

[7.5 Контрольный пример 25](#_Toc533046751)

[Глава 8. Тестирование транслятора 26](#_Toc533046752)

[8.1 Тестирование проверки на допустимость символов 26](#_Toc533046753)

[8.2 Тестирование лексического анализатора 26](#_Toc533046754)

[8.3 Тестирование синтаксического анализатора 26](#_Toc533046755)

[8.4 Тестирование семантического анализатора 26](#_Toc533046756)

[Приложение А 27](#_Toc533046757)

[Приложение Б 29](#_Toc533046758)

[Приложение В 33](#_Toc533046759)

[Приложение Г 34](#_Toc533046760)

[Приложение Е 35](#_Toc533046761)

[Приложение Ж 37](#_Toc533046762)

[Заключение 41](#_Toc533046763)

[Литература 42](#_Toc533046764)

# Введение

Задачей данного курсового проекта является разработка собственного языка программирования и компилятора для него. Язык называется GDV-2022. Написание компилятора будет осуществляться на языке C++.

Для выполнения курсового проекта были поставлены следующие задачи:

* Спецификация языка программирования
* Структура транслятора
* Разработка лексического анализатора
* Разработка синтаксического анализатора
* Разработка семантического анализатора
* Вычисление выражений
* Генерация кода на языке C++
* Контрольный пример
* Тестирование транслятора

# Глава 1. Спецификация языка программирования

## 1.1 Характеристика языка программирования

Язык программирования GDV-2022 – это процедурный, универсальный, строго типизированный, не С-подобный, компилируемый язык. Не является объектно-ориентированным.

## 1.2 Алфавит языка

Алфавит языка GDV-2022 основывается на таблице ASCII, представленной в таблице 1.1. Используемые в алфавите символы: **[a … z], [A … Z], [0 … 9]**, **[А … Я], [а … я]** спецсимволы: **() , ; :`'** , а также символы пробела, табуляции и перевода строки.

Таблица 1.1. Таблица кодировок ASCII



## **1.3 Применяемые сепараторы**

Применяемые сепараторы в языке программирования GDV-2022 описаны в таблице 1.2.

Таблица 1.2. Сепараторы языка GDV-2022.

|  |  |
| --- | --- |
| Сепараторы | Назначение сепаратора |
| { … } | Для блока функций |
| ( … ) | Для фактических или формальных параметров функции, а также для приоритета операций |
| ‘пробел’, ‘табуляция’ | Для разделения цепочек (не допускается в названиях идентификаторов и ключевых слов) |
| , | Для разделения параметров функции, цикла For |
| &, |, ~ | Побитовые операции |
| ; | Для разделения программных инструкций |
| = | Оператор присваивания |
| ? | Условный оператор |
| \n | Символ перехода на новую строку |
| <, >, ==, != | Операторы сравнения |

## 1.4 Применяемые кодировки

Для написания исходного кода на языке программирования GDV-2022 используется кодировка Windows-1251.

Таблица 1.3. Кодировка Windows-1251

## 1.5 Типы данных

Язык GDV-2022 позволяет использовать 3 типа данных для переменных и функций: целочисленный (num), символьный(symb) и числа с плавающей точкой(float). А также есть особый тип данных для функции — action. Описание этих типов данных приведено в таблице 1.4.

Таблица 1.4. Типы данных языка GDV-2022

|  |  |
| --- | --- |
| Тип данных | Описание |
| num | Целочисленный четырехбайтный тип данных. Значения по умолчанию нет |
| symb | Символьный однобайтовый тип данных. Относительно является целочисленным типом данных, значения колеблются от -128 до 127 |
| float | Тип данных, отвечающий за числа с плавающей точкой. Занимает 4 байта в памяти |
| action | Особый тип данных функции, который указывает, что данная функция является процедурой, которая не возвращает никакого значения |

## 1.6 Преобразование типов данных

Преобразование типов данных происходит неявно. Так как все типы данных языка GDV-2022 можно отнести к численным, их взаимодействию особо ничего не мешает, но все-таки бывают такие случаи, где без приведения типов не обойтись. При вызове функций идентификаторы и литералы будут приводится к типу параметров функций. Также при использовании побитовых операций, тип данных float будет приведен к типу данных num. При использовании цикла For по типу данных параметра, определяющему шаг, будет определен тип данных, к которому будут приведены первые два параметра цикла.

Приведение в языке GDV-2022 позволяет не ограничивать возможности типа данных float, то есть использовать побитовые операции, позволяет определить, с чем работает цикл For, и позволяет беспрепятственно использовать функцию, передавая в параметры переменные, литералы любого типа данных.

## 1.7 Идентификаторы

При создании идентификатора можно использовать буквы латинского алфавита как верхнего, так и нижнего регистра, а также можно использовать цифры и символ нижнего подчеркивания. Максимальная длина идентификатора – 50 символов. При превышении длины, идентификаторы усекаются до длины, равной 50 символов. Идентификаторы не должны совпадать с ключевыми словами и с другими идентификаторами, созданных в одной и той же области видимости.

## 1.8 Литералы

В языке GDV-2022 предусмотрены 4 вида литералов: целочисленные и строковые, символьные и чисел с плавающей точкой. Краткое описание литералов приведено в таблице 1.5.

Таблица 1.5 Литералы

|  |  |
| --- | --- |
| Литерал | Пояснение |
| Целочисленный | Максимально допустимое значение 231-1. Минимально допустимым является -231-1. При выходе за пределы допустимости выводится соответствующая ошибка. В случае отрицательного значения используется знак минус. |
| Символьный | Используются символы из кодировки Windows-1251. При присвоении целочисленного значения или значения числа с плавающей точкой будет присвоен символ с соответствующей кодировкой. Если значение выше или ниже, ошибки не будет, так как система сама решит проблему с переполнением. |
| Числа с плавающей точкой | Максимальное значение — 3.402823466e+38F. Минимальное значение — 1.175494351e-38F. При выходе за пределы вызывается ошибка. В случае отрицательного значения используется знак минус. Литерал должен иметь в записи символ точки. |
| Строковый | Используются символы кодировки ASCII. Максимальный размер строки — 255. Используется только с потоком вывода console. |

## 1.9 Объявление данных

В языке программирования GDV-2022 необходимо объявить переменную до ее использования. Есть два типа объявления: с явной типизацией и с неявной. При явной мы должны сразу указывать тип данных переменной. При неявной типизации язык сам вычислит тип данных переменной, но для этого сразу нужно использовать оператор '=>', после которого должен идти либо идентификатор, либо литерал, иначе будет ошибка.

Для определения типа переменной нужно использовать ключевое слово is после идентификатора, а после него сам тип данных (num, symb, float).

Пример объявления переменной целочисленного типа: n is num;

Пример объявления переменной символьного типа: s is symb;

Пример объявления переменной типа числа с плавающей точкой: f is float;

Пример объявления переменной целочисленного типа с неявной типизацией:

n => 2;

Пример объявления переменной символьного типа с неявной типизацией:

s => ‘s’;

Пример объявления переменной типа числа с плавающей точкой с неявной типизацией: f => .2.

Объявление функции схоже с объявлением обычной переменной, также используется оператор is после идентификатора, после него уже следует ключевое слово foo, что обозначает, что это функция, далее идет открывающая круглая скобочка, после которой идет инициализация параметров. Инициализация параметров проходит также, как инициализация обычных переменных. Чтобы инициализировать несколько параметров, нужно сделать несколько инициализаций и разделить их запятыми. После инициализации будет следовать закрывающая кругла скобочка. Далее будет определен тип данных возвращаемого значения этой функции. Для определения используется, как правило, оператор is после скобочки. Типы данных функции совпадают с типами данных переменных, только тут добавляется дополнительный тип данных action. Тип данных указывается после оператора is. Далее будет идти тело функции, заключенное в фигурные скобки.

Пример объявления функции целочисленного типа:

Sum is foo(a is num, b is num) is num

{

return sum(a, b);

}

Также язык программирования GDV-2022 позволяет объявлять шаблонные функции. Шаблонная функция в языке GDV-2022 — функция, объявленная не в глобальной области видимости, а в другой, допустим в другой функции или в main. Их объявление такое же как и объявление обычной функции, только типа данных action нет и после закрывающей фигурной скобки будет идти символ точки с запятой.

## 1.10 Инициализация данных

Для инициализации переменных используются два оператора: = и =>. Их механика инициализации схожа, а именно слева от операторов должен быть идентификатор, а справа — литерал, либо значение, либо выражение. Только если оператор =, то справа может быть любое выражение, а, если оператор =>, справа могут быть выражения, начинающиеся с идентификатора или же с литерала.

## 1.11 Инструкции языка

Инструкции языка программирования GDV-2022 представлены в таблице 1.6

Таблица 1.6. Инструкции языка GDV-2022.

|  |  |
| --- | --- |
| Инструкция | Запись на языке GDV-2022 |
| Объявление переменной с явной типизацией | <идентификатор> is <тип данных>; |
| Объявление и инициализация переменной с неявной типизацией | <идентификатор> => <выражение>; |
| Вывод данных: | console([<список параметров>]); |
| Вызов подпрограммы | <идентификатор функции> ([<список параметров>]); |
| Перевод строки | console(); |
| Присваивание | <идентификатор> = <выражение>; |

## 1.12 Операции языка

В языке программирования GDV-2022 присутствуют операции, описанные в таблице 1.7. Операции, заключенные в (..) (круглые скобки) имеют наивысший приоритет, равный 4. Операция ~ (логическое не) имеет приоритет 3. Операция & (логического умножения) имеет приоритет, равный 2. Операция | (логическое сложение) имеет низший приоритет, равный 1.

Операции логического умножения и логического сложения являются бинарными, то есть требуют два операнда. Операция логического не является унарной, то есть требует только один операнд.

Таблица 1.7. Операции языка GDV-2022.

|  |  |
| --- | --- |
| Операции | Операторы |
| Побитовые | | (логическая сложение)  & (логическое умножение)  ~ (логическое не) |

## 1.13 Выражения и их вычисление

Предусмотрены следующие правила составления выражений:

* Рассматриваются слева направо.
* Для изменения приоритета операции используются круглые скобки ()
* Каждое выражение должно заканчиваться сепаратором

В выражения можно использовать комбинацию всех типов данных, кроме типа

данных процедуры action. В выражениях можно использовать все виды операций языка GDV-2022. Также есть специальные функции из статической библиотеки sum(), mult(), division(), minus() для выполнения арифметических операций.

Выражения можно использовать после операции присваивания и после оператора возвращения значения return.

## 1.14 Конструкции языка

Программные конструкции языка программирование GDV-2022 приведены в таблице 1.8.

Таблица 1.8. Программные конструкции языка

|  |  |
| --- | --- |
| Конструкции | Представление в языке |
| Точка входа в программу | main {…} |
| Пользовательская функция | <идентификатор> is foo(<идентификатор> is <тип данных>, …) is <тип данных>  {  …  return <выражение>;  } |
| Процедура | <идентификатор> is foo(<идентификатор> is <тип данных>, …) is action  {  …  } |
| Шаблонная функция | <идентификатор> is foo(<идентификатор> is <тип данных>, …) is <тип данных>  {  …  return <выражение>;  }; |
| Лямбда-функция в цикле For | <идентификатор> => {  …  } |

## 1.15 Области видимости идентификаторов.

Область видимости построена по принципу С++. Все идентификаторы должны быть вызваны из текущей области видимости или областей видимости, в которых они находятся. Дополнительных указателей не требуется. Для создания искусственной области видимости используются фигурные скобки {…}.

## 1.16 Семантические проверки

Перечень семантических проверок, предусмотренных языком программирования GDV-2022, приведен в таблице 1.9.

Таблица 1.9. Семантические проверки

|  |  |
| --- | --- |
| № | Проверка |
| 1 | Единственность точки входа |
| 2 | Превышение размера целочисленных, строковых и плавающих литералов |
| 3 | Правильность выражений |
| 4 | Правильность передаваемых в функцию параметров: количество, типы |
| 5 | Переопределение идентификаторов |
| 6 | Использование идентификаторов без их объявления |

## 1.17 Распределение оперативной памяти на этапе выполнения

Все переменные размещаются в стеке. Шаблонная функция в куче.

## 1.18 Стандартная библиотека и её состав

В языке GDV-2022 присутствует стандартная библиотека, которая подключает некоторые функции автоматически при трансляции исходного кода в язык С++, а некоторые функции можно подключить с помощью ключевого слова @import. Содержимое библиотеки и описание функций представлено в таблице 1.10.

Таблица 1.10. Стандартная библиотека языка GDV-2022

|  |  |
| --- | --- |
| Функция | Описание |
| sum is foo(a is num, b is num) is num; | Математическая функция. Заменяет арифметическое суммирование. Подключается автоматически. Тип параметров и возвращаемого значения целочисленный |
| sum is foo(a is float, b is float) is float; | Математическая функция. Заменяет арифметическое суммирование. Подключается автоматически. Тип параметров и возвращаемого значения плавающее значение |
| sum is foo(a is symb, b is symb) is symb; | Математическая функция. Заменяет арифметическое суммирование. Подключается автоматически. Тип параметров и возвращаемого значения символьный |
| minus is foo(a is num, b is num) is num; | Математическая функция. Заменяет арифметическую разницу. Подключается автоматически. Тип параметров и возвращаемого значения целочисленный |
| minus is foo(a is float, b is float) is float; | Математическая функция. Заменяет арифметическую разницу. Подключается автоматически. Тип параметров и возвращаемого значения плавающее значение |
| minus is foo(a is symb, b is symb) is symb; | Математическая функция. Заменяет арифметическую разницу. Подключается автоматически. Тип параметров и возвращаемого значения символьный |
| mult is foo(a is num, b is num) is num; | Математическая функция. Заменяет арифметическое произведение. Подключается автоматически. Тип параметров и возвращаемого значения целочисленный |
| mult is foo(a is float, b is float) is float; | Математическая функция. Заменяет арифметическое произведение. Подключается автоматически. Тип параметров и возвращаемого значения плавающее значение |
| mult is foo(a is symb, b is symb) is symb; | Математическая функция. Заменяет арифметическое произведение. Подключается автоматически. Тип параметров и возвращаемого значения символьный |
| division is foo(a is num, b is num) is num; | Математическая функция. Заменяет арифметическое деление. Подключается автоматически. Тип параметров и возвращаемого значения целочисленный |
| division is foo(a is float, b is float) is float; | Математическая функция. Заменяет арифметическое деление. Подключается автоматически. Тип параметров и возвращаемого значения плавающее значение |
| division is foo(a is symb, b is symb) is symb; | Математическая функция. Заменяет арифметическое деление. Подключается автоматически. Тип параметров и возвращаемого значения символьный |
| pow is foo(number is num, power is num) is num; | Математическое функция. Выполняет роль возведения числа в степень. Подключается с помощью ключевого слова @import. Тип параметров и возвращаемого значения целочисленный |
| symb\_to\_num is foo(s is symb) is num; | Символьная функция. Переводит из символьного литерала в число. Если значение литерала число, то вернет это же число, только делая его целочисленным литералом. Подключается с помощью ключевого слова @import. Тип параметра символьный. Тип возвращаемого значения целочисленный |
| abs is foo(number is num) is num; | Математическая функция. Возвращает модуль от числа, переданного в параметр. Подключается при помощи ключевого слова @import. Типы данных параметра и возвращаемого значения совпадают и раны num |

## 1.19 Ввод и вывод данных

В языке GDV-2022 вывод данных осуществляется с помощью оператора print. В качестве аргумента могут выступать литералы и идентификаторы. Ввод в данном языке программирования не предусмотрен.

## 1.20 Точка входа

Точкой входа в программе является ключевое слово “main”. Точка входа не может отсутствовать или быть переопределена.

## 1.21 Препроцессор

В языке GDV-2022 препроцессор не предусмотрен.

## 1.22 Соглашения о вызовах

Используется соглашение \_stdcall, то есть все параметры передаются в стек справа налево, память высвобождает вызываемый код.

## 1.23 Объектный код

Объектный код реализован на основе языка программирования Assembler.

## 1.24 Классификация сообщений транслятора

Классификация сообщений транслятора приведена в таблице 1.12.

Таблица 1.12. Сообщения транслятора

|  |  |
| --- | --- |
| Кодысообщений | Принадлежность |
| 0 – 200 | Системные ошибки |
| 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

Исходный код контрольного примера представлен ниже.

number function s(number d, number e)[

new number res;

res = e+(d+e)\*5;

end res;

]

proc function three(string a, string b, number c, 5)[

new string r;

r = catstring(a, b);

print r;

print c;

endl;

end;

]

main[

new string str;

str="string";

new number tt;

tt = plength(str);

print tt;

new number x;

x = 7;

new number y;

y = 7;

new number t;

t = x+y+s(x, y);

three("Summa", " is ", t);

]

# Глава 2. Структура транслятора

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор языка программирования GDV-2022 состоит из следующих частей:

Лексический анализатор – часть транслятора, на котором выполняется лексический анализ. На данном этапе распознаётся правильность составления лексем и идентификаторов.

Синтаксический анализатор – часть транслятора, на которой выполняется синтаксический анализ. Проверяется правильность расположения идентификаторов и ключевых слов в исходном коде. Для того, чтобы провести данную операцию используются таблица лексем и идентификаторов.

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть исходный код проверяется на наличие ошибок. Входными данными являются таблица лексем и идентификаторов.

Генератор кода – часть транслятора, выполняющая генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. На вход генератора подаются таблица лексем и таблица идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

## 2.2 Перечень входных параметров транслятора

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

Таблица 2.1. Входные параметры транслятора языка GDV-2022

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<имя in-файла> | Файл с исходным кодом на языке программирования GDV-2022, имеющий расширение .txt | Не предусмотрено |
| -log:<имя log-файла> | Файл, содержащий вывод протокола работы программы. | Значение по умолчанию:  <имя in-файла>.log |
| -out:<имя out-файла> | Выходной файл – результат работы транслятора. Содержит исходный код на языке ассемблера. | Значение по умолчанию:  <имя in-файла>.out |

## 2.3 Протоколы, формируемые транслятором

В ходе работы программы формируются протоколы работы лексического, синтаксического и семантического анализаторов, которые содержат в себе перечень протоколов работы. В таблице 2.2 приведены протоколы, формируемые транслятором и их содержимое.

Таблица 2.2 Протоколы, формируемые транслятором языка GDV-2022

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" log.txt | Файл с протоколом работы транслятора языка программирования GDV-2022. Содержит таблицу лексем с польской нотацией. |
| Выходной файл, заданный параметром "-out:" out.asm | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |
| LT.txt | Сформированная таблица лексем, все выражения представлены в польской записи |
| IT.txt | Сформированная таблица идентификаторов |
| SNT.txt | Содержит протокол работы синтаксического анализатора и дерево разбора, полученные на этапе синтаксического анализа. |

# Глава 3 Разработка лексического анализатора

## 3.1 Структура лексического анализатора

Лексический анализатор – часть компилятора, которая выполняет лексический анализ. На данном этапе распознаётся правильность составления лексем и идентификаторов языка. Для работы лексический анализатор использует исходный код на языке GDV-2022. В итоге будут сформированы таблица лексем и таблица идентификаторов.

Структура лексического анализатора представлена на рисунке 3.1.

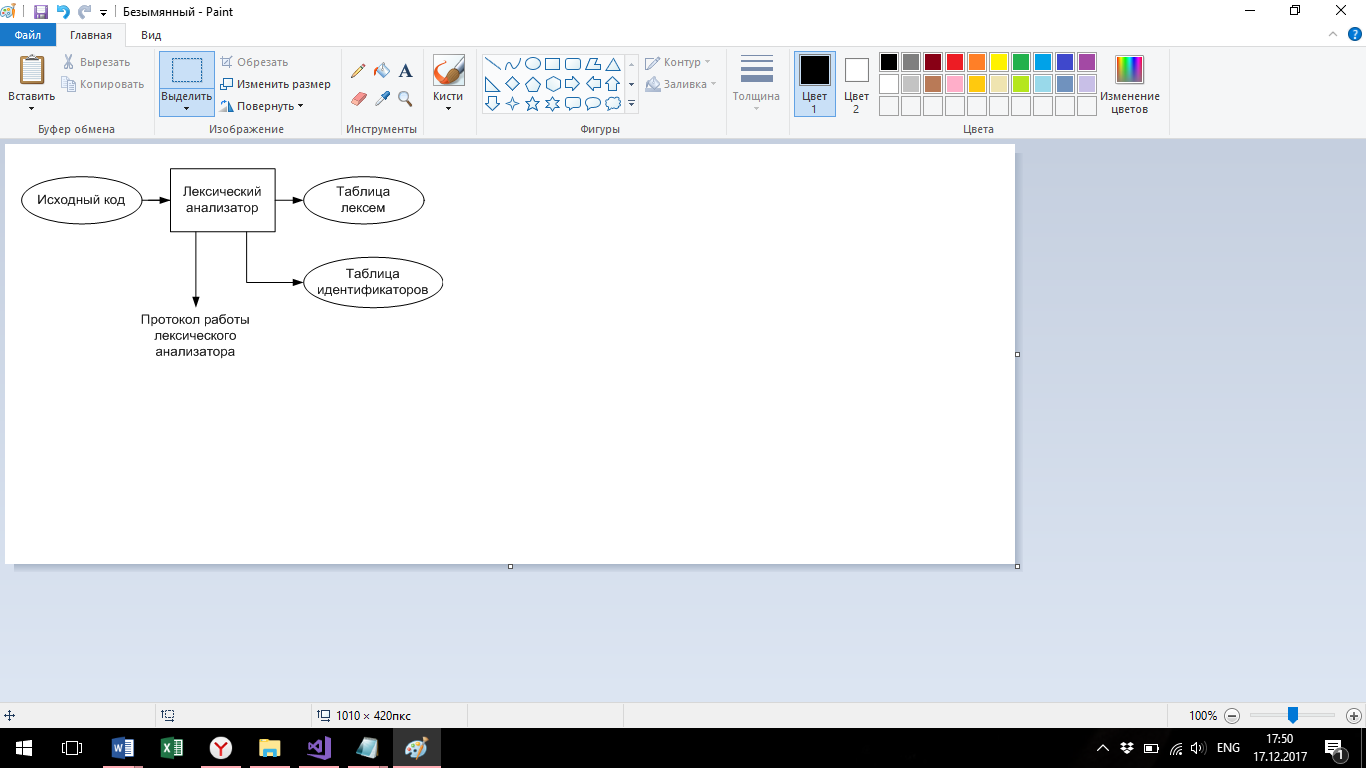


Рис. 3.1 Структура лексического анализатора

## 3.2 Контроль входных символов

При передаче исходного кода в лексический анализатор, все символы разделяются по определённым категориям, для дальнейшего использования. Категории входных символов представлены в таблице 3.1.

Таблица 3.1 Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенный | T |
| Запрещенный | F |
| Игнорируемый | I |
| Литерал | Q |
| Сепаратор | S |
| Перевод строки | N |
| Пробел, табуляция | P |

## 3.3 Удаление избыточных символов

В языке программирования GDV-2022 предусмотрено удаление избыточных символов. Удаление избыточных символов происходит на этапе формирования слов, которые поступят на вход лексического анализатора. Пробелы и символы табуляции не участвуют в формировании слов, если только они не внутри строкового литерала.

## 3.4 Перечень ключевых слов

Соответствие ключевых слов, сепараторов, символов операций с лексемами приведено в таблице 3.2.

Таблица 3.2 Соответствие ключевых слов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Конструкция | Лексема | Примечание |
| number  string | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 9 символов. |
| Литерал | l | Литерал любого доступного типа. |
| function | f | Объявление функции. |
| proc | p | Ключевое слово для процедур – функций, не возвращающих значения. Указывается перед ключевым словом function. |
| end | e | Выход из функции/процедуры. |
| main | m | Главная функция. |
| new | n | Объявление переменной. |
| print | o | Вывод данных. |
| endl | ^ | Оператор вывода в выходной поток символа перевода строки. |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| [ | [ | Начало блока/тела функции. |
| ] | ] | Закрытие блока/тела функции. |
| ( | ( | Передача параметров в функцию, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | = | Знак присваивания. |
| +, -, \* | +, -, \* | Знаки операций. |

Реализации графов переходов находятся в приложении А.

Пример реализованного конечного автомата ключевого слова main языка GDV-2022 представлен на рисунке 3.2.

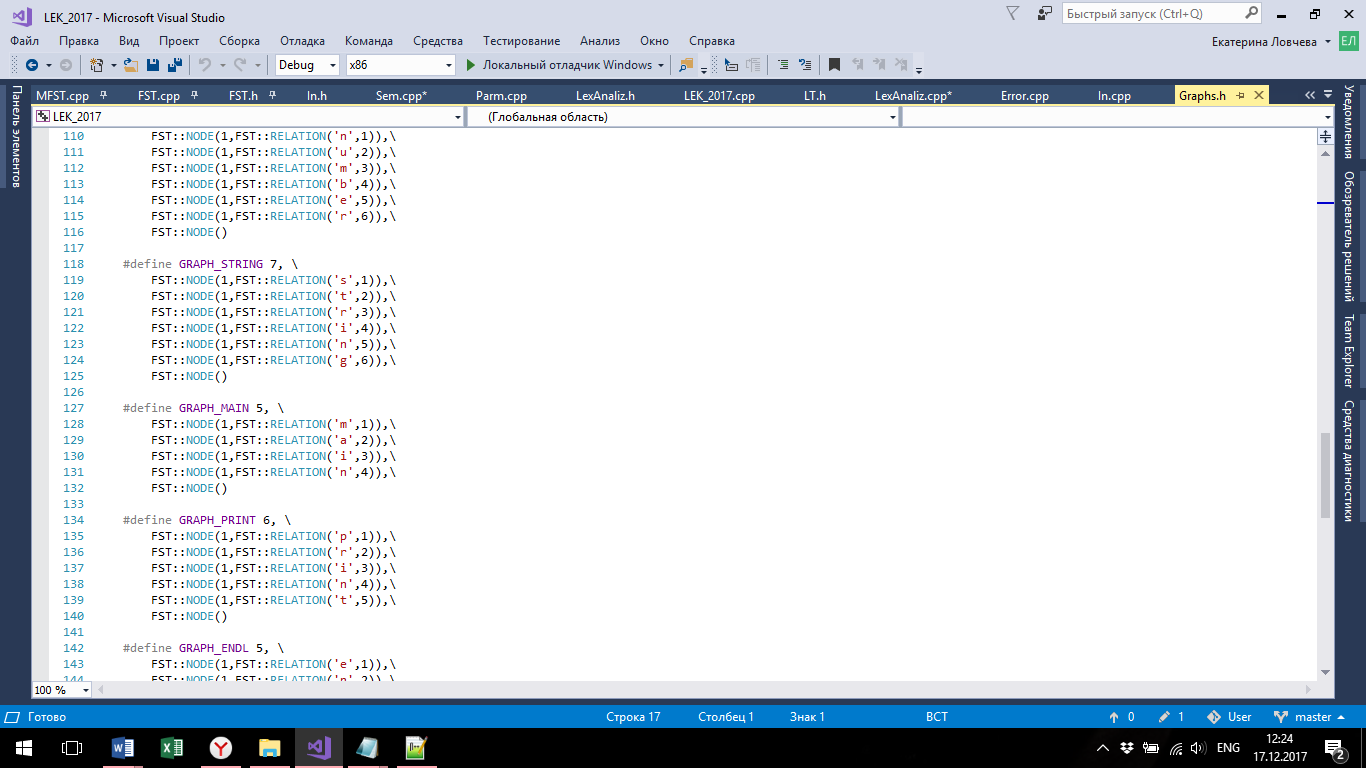


Рис. 3.2 Реализация конечного автомата для ключевого слова main

## 3.5 Основные структуры данных

Основные структуры данных приведены в приложении А.

## 3.6 Структура и перечень сообщений лексического анализатора

Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. Перечень сообщений представлены в таблице 3.3.

Таблица 3.3. Перечень ошибок лексического анализатора

|  |  |
| --- | --- |
| Код сообщения | Содержание сообщения |
| 200 | \*\*\*ЛА\*\*\* Недопустимый символ в исходном файле(-in) |
| 201 | \*\*\*ЛА\*\*\* Неизвестная последовательность символов |
| 202 | \*\*\*ЛА\*\*\* Превышен размер таблицы лексем |
| 203 | \*\*\*ЛА\*\*\* Превышен размер таблицы идентификаторов |

## 3.7 Принцип обработки ошибок

В случае возникновения ошибок происходит их протоколирование с номером ошибки и сообщением. Лексический анализатор продолжает работу с исходным кодом, проверяет его до конца и выводит оставшиеся сообщения об ошибках в файл протокола “\_log.txt", если они существуют.

## 3.8 Параметры лексического анализатора

Параметром лексического анализатора является очередь из структур, полями которых являются лексемы в исходном коде, полученные на этапе проверки кода на допустимость символов.

## 3.9 Алгоритм лексического анализа

Алгоритм работы лексического анализа заключается в распознавании и разборе цепочек исходного кода на основе конечных автоматов, а также заполнение таблиц идентификаторов и лексем. Работу конечного автомата можно показать с помощью графа переходов. Пример графа для цепочки «main» приведен на рисунке 3.3.

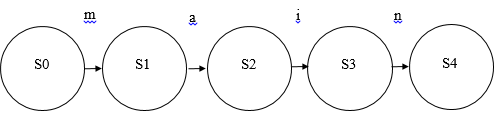


Рис 3.3 – Пример графа для цепочки main

## 3.10 Контрольный пример

Контрольный пример в виде таблиц лексем и идентификаторов представлен в приложении Б.

# Глава 4. Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора

Синтаксический анализ – фаза компилятора, которая выполняется после лексического анализа. В этой фазе будут распознаваться синтаксические конструкции и формироваться промежуточный код. На вход синтаксического анализатора будет подаваться таблица лексем и таблица идентификаторов, а результатом работы будет дерево разбора.

## Контекстно-свободная грамматика, описывающая синтаксис языка

Синтаксис языка GDV-2022 описывается грамматикой типа 2 по иерархии Хомского:

G = <T, N, P, S>

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

N – множество нетерминальных символов,

P – множество правил языка,

S – начальный символ грамматики, представленный нетерминальным символом «S».

Множество терминальных символов соответствует элементам, содержащимся в таблице лексем. Правила нетерминальных символов описаны в таблице 4.1

Таблица 4.1. Правила нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Нетерминалы | Назначение | Правила |
| S | Стартовый символ | tfiPTS |
| pfiPGS |
| m[L] |
| P | Правила для параметров объявляемых функций | () |
| (E) |
| T | Правила для тела функций | [eV;] |
| [LeV;] |
| G | Правила для тела процедур | [e;] |
| [Le;] |
| E | Правила для параметров вызываемых функций | ti |
| ti,E |
| F | Правила для вывозов функций | () |
| (N) |
| N | Правила для параметров вызываемых функций | i |
| L |
| i,N  l,N |

Таблица 4.1. Правила нетерминальных символов (продолжение)

|  |  |  |
| --- | --- | --- |
| A | Правила для арифметических операторов | + |
| - |
| \* |
| V | Правила для выражений | i |
| l |
| W | Правила для арифметического выражения | i |
| l |
| (W) |
| (W)AW |
| iF |
| iAW |
| lAW |
| iFAW |
| L | Программные конструкции | nti;L |
|  | i=W;L |
|  | oV;L |
|  | ^;L |
|  | iF;L |
|  | i=W; |
|  | nti; |
|  | oV; |
|  | ^; |
|  | iF; |

## 4.3 Построение конечного магазинного автомата

Распознавателем грамматики является конечный автомат с магазинной памятью, который представляет собой семерку.Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2. Описание компонент магазинного автомата

|  |  |
| --- | --- |
| Компонента | Определение |
|  | Множество состояний автомата |
|  | Алфавит входных символов |
|  | Алфавит специальных магазинных символов |
|  | Функция переходов автомата |
|  | Начальное состояние автомата |
|  | Начальное состояние магазина автомата |
|  | Множество конечных состояний |

## 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 В.

## 4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора можно описать следующим образом:

* В магазин записывается стартовый символ.
* На основе полученной таблицы лексем формируется входная лента.
* Запускается автомат и выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке.
* Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется с ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала.
* Если в магазине встретился нетерминал, переходим к пункту 3.
* Если символ достиг символа дна стека, и лента в этот момент имеет символ дна стека, то синтаксический анализ выполнен успешно. Иначе генерируется ошибка.

## 4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен в таблице 4.3.

Таблица 4.3. Перечень ошибок синтаксического анализатора

|  |  |
| --- | --- |
| Код сообщения | Содержание сообщения |
| 600 | \*\*\*СИНТАКСИС\*\*\* Ошибка. Неверная структура программы |
| 601 | \*\*\*СИНТАКСИС\*\*\* Ошибка. Не найден список параметров функции |
| 602 | \*\*\*СИНТАКСИС\*\*\* Ошибка в теле функции |
| 603 | \*\*\*СИНТАКСИС\*\*\* Ошибка в теле процедуры |
| 604 | \*\*\*СИНТАКСИС\*\*\* Ошибка в списке параметров функции |
| 605 | \*\*\*СИНТАКСИС\*\*\* Ошибка в вызове функции/выражении |
| 606 | \*\*\*СИНТАКСИС\*\*\* Ошибка в списке фактических параметров функции |
| 607 | \*\*\*СИНТАКСИС\*\*\* Ошибка. Неверный арифметический оператор |
| 608 | \*\*\*СИНТАКСИС\*\*\* Ошибка. Неверное выражение. Ожидаются только идентификаторы/литералы |
| 609 | \*\*\*СИНТАКСИС\*\*\* Ошибка в арифметическом выражении |
| 610 | \*\*\*СИНТАКСИС\*\*\* Ошибка. Недопустимая синтаксическая конструкция" |

## 4.7 Параметры синтаксического анализатора и режимы его работы

Входными параметрами для синтаксического анализатора в языке программирования GDV-2022 являются таблица лексем и таблица идентификаторов.

## 4.8 Принцип обработки ошибок

Синтаксический анализатор перебирает все возможные правила и цепочки правила грамматики в целях поиска подходящего соответствия. Если ни одна из цепочек правила не подошла для рассматриваемой конструкции, то генерируется ошибка в соответствии с таблицей 4.3. Ошибка заносится в протокол.

## 4.9 Контрольный пример

Пример разбора исходного кода на языке программирования GDV-2022 синтаксическим анализатором представлен в приложении Г.

# Глава 5. Разработка семантического анализатора

## 5.1 Структура семантического анализатора

Семантический анализатор принимает на свой вход таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Также семантические проверки предусмотрены на этапе лексического анализа.

## 5.2 Функции семантического анализатора

Семантический анализатор выполняет проверку на основе правил языка, описанных в п. 1.16

## 5.3 Структура и перечень сообщений семантического анализатора

Перечень сообщений семантического анализатора представлен в таблице 5.1.

Таблица 5.1. Перечень ошибок семантического анализатора

|  |  |
| --- | --- |
| Код сообщения | Содержание сообщения |
| 300 | \*\*\*СЕМ\*\*\* Необъявленный идентификатор |
| 301 | \*\*\*СЕМ\*\*\* Отсутствует точка входа main |
| 302 | \*\*\*СЕМ\*\*\* Обнаружено несколько точек входа main |
| 303 | \*\*\*СЕМ\*\*\* В объявлении не указан тип идентификатора |
| 304 | \*\*\*СЕМ\*\*\* В объявлении отсутствует ключевое слово new |
| 305 | \*\*\*СЕМ\*\*\* Попытка переопределения идентификатора |
| 306 | \*\*\*СЕМ\*\*\* Превышено максимальное количество параметров функции |
| 307 | \*\*\*СЕМ\*\*\* Слишком много параметров в вызове |
| 308 | \*\*\*СЕМ\*\*\* Кол-во ожидаемых функцией и передаваемых параметров не совпадают " |
| 309 | \*\*\*СЕМ\*\*\* Несовпадение типов передаваемых параметров |
| 310 | \*\*\*СЕМ\*\*\* Использование пустого строкового литерала недопустимо |
| 311 | \*\*\*СЕМ\*\*\* Обнаружен символ \'\"\'. Возможно, не закрыт строковый литерал |
| 312 | \*\*\*СЕМ\*\*\* Превышен размер строкового литерала |
| 313 | \*\*\*СЕМ\*\*\* Недопустимый целочисленный литерал |
| 314 | \*\*\*СЕМ\*\*\* Типы данных в выражении не совпадают |
| 315 | \*\*\*СЕМ\*\*\* Тип функции и возвращаемого значения не совпадают |
| 316 | \*\*\*СЕМ\*\*\* Недопустимое строковое выражение справа от знака \'=\'" |

## 5.4 Принцип обработки ошибок

Семантический анализатор, в случае возникновения ошибки, заносит её в протокол. Следующий этап трансляции не будет запущен при возникновении ошибки. Семантический анализ начинает проверки уже на стадии лексического анализа, если на этой стадии обнаружены семантические ошибки – программа завершит свою работу.

**5.5 Контрольный пример**

Обработка ошибок семантического анализатора представлена в п. 8.4.

# Глава 6. Преобразование выражений

## 6.1 Выражения, допускаемые языком

В языке программирования GDV-2022 выражения могут содержать вычисления целочисленных типов данных, а также допускаются вызов функций (возвращающих тип) внутри выражений. Приоритет операций представлен на таблице 6.1.

Таблица 6.1 Приоритет операция языка GDV-2022

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( ) | 3 |
| \* | 2 |
| +, - | 1 |

## 6.2 Польская запись и принцип ее построения

В языке программирования GDV-2022 представлена польская запись. Результат операции заменяет в выражении последовательность её операндов и знак операции. Результатом вычисления становится результат последней вычисленной операции.

Принцип построения польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку в порядке их следования;
* операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;
* по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

## 6.3 Программная реализация обработки выражений

Программная реализация обработки выражений представлена в приложении Е.

## 6.4 Контрольный пример

Пример преобразования выражения к польской записи приведен в таблице 6.2.

Таблица 6.2. Пример выражения в польской записи

|  |  |
| --- | --- |
| Исходная строка | Результирующая строка |
| i = i+(i+i)\*l; | i=iii+l\*+##; |

# Глава 7. Генерация кода

## 7.1 Структура генератора кода

Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В соответствии с таблицей лексем строится выходной файл на языке ассемблера, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода GDV-2022 представлена на рисунке 7.1.



Рисунок 7.1 Структура генератора кода

## 7.2 Представление типов данных в оперативной памяти

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

Таблица 7.1 Соответствия типов идентификаторов

|  |  |  |
| --- | --- | --- |
| Тип идентификатора | Тип идентификатора на языке ассемблера | Пояснение |
| number | sdword | Хранит целочисленный тип данных. |
| string | dword | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |
| L(0-4) | sdword, byte | Целочисленные и строковые литералы |

## 7.3 Статическая библиотека

В языке GDV-2022 предусмотрена статическая библиотека, которая содержит функции, написанные на языке C++, приведенные в таблицах 1.10 и 1.11. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера.

## 7.4 Особенности алгоритма генерации кода

Алгоритм генерации объектного кода выглядит следующим образом:

* На основе таблицы идентификаторов идет заполнение поля .const литералами. Результат представлен ниже

.const  
endl byte 13, 10, 0

L0 sdword 5

L1 byte ‘string’, 0

L2 sdword 7

L3 byte ‘Summa’, 0

L4 byte ‘ is ‘, 0

* Далее идет обход таблицы лексем, ищется объявление переменных. В случае, если был найден литерал ‘n’, идет объявление данной переменной в поле .data. Результат представлен ниже

.data

temp sdword ?

buffer byte 256 dup(0)

sres sdword 0

sr dword ?

mainstr dword ?

maintt sdword 0

mainx sdword 0

mainy sdword 0

maint sdword 0

* Далее идет обход таблицы лексем с целью описать функции и конструкции языка GDV-2022.

## 7.5 Контрольный пример

Контрольный пример ассемблерного кода приведен в приложении Ж.

Результат генерации ассемблерного кода приведен на рисунке 7.2.

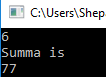


Рис.7.2 Результат генерации ассемблерного кода

# Глава 8. Тестирование транслятора

## 8.1 Тестирование проверки на допустимость символов

В языке GDV-2022 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 Тестирование проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| three("д", " is ", t); | Ошибка N200: \*\*\*ЛА\*\*\* Недопустимый символ в исходном файле(-in) Строка: 22 Позиция в строке: 8 |

## 8.2 Тестирование лексического анализатора

Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  [ new number x11; ] | Ошибка N201: \*\*\*ЛА\*\*\* Неизвестная последовательность символов Строка: 19 |

## 8.3 Тестирование синтаксического анализатора

Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main new number x; ] | 600: строка 1, \*\*\*СИНТАКСИС\*\*\* Ошибка. Неверная структура программы |

## 8.4 Тестирование семантического анализатора

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main [ a = 1 ] | Ошибка N300: \*\*\*СЕМ\*\*\* Необъявленный идентификатор Строка: 1 |

# Приложение А

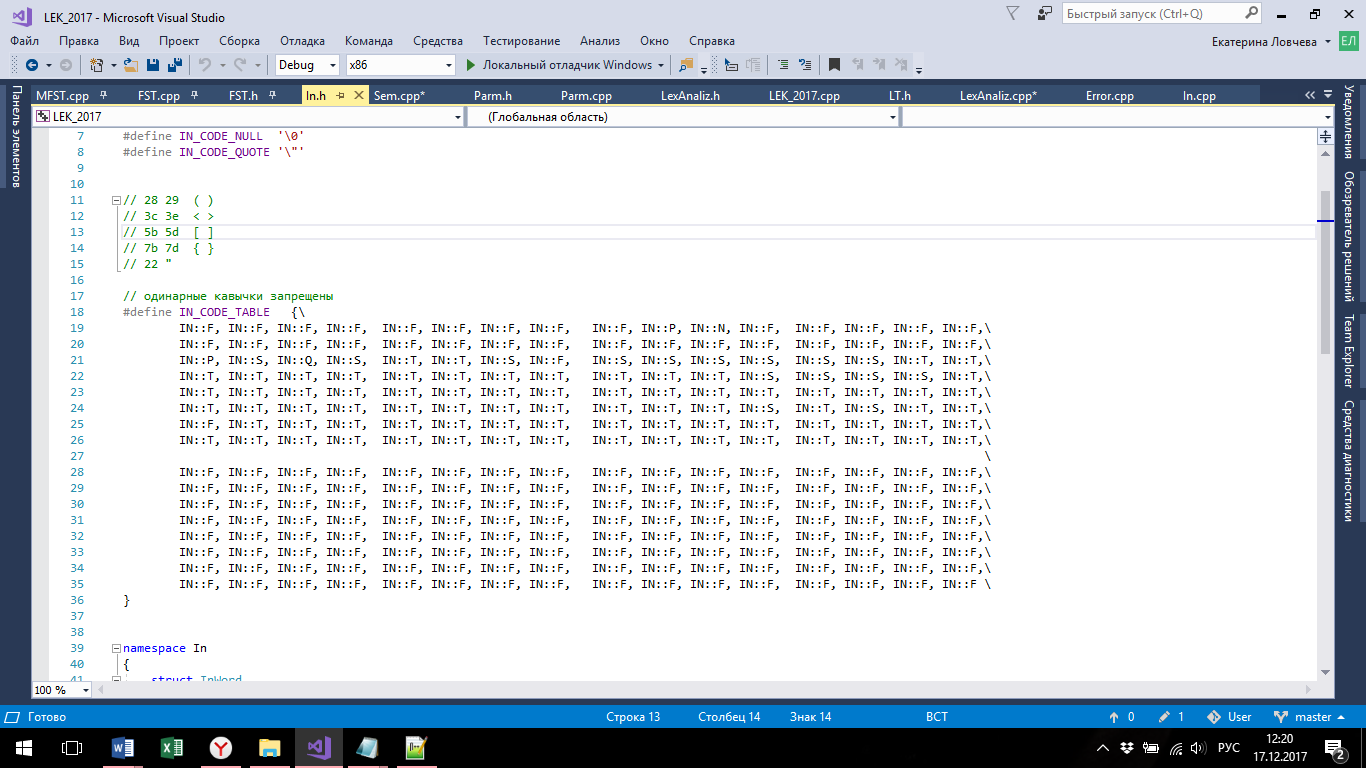
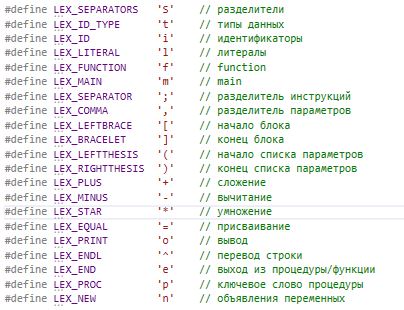


Рис. 1 Контроль входных символов

  
Рис. 2 Лексемы языка GDV-2022

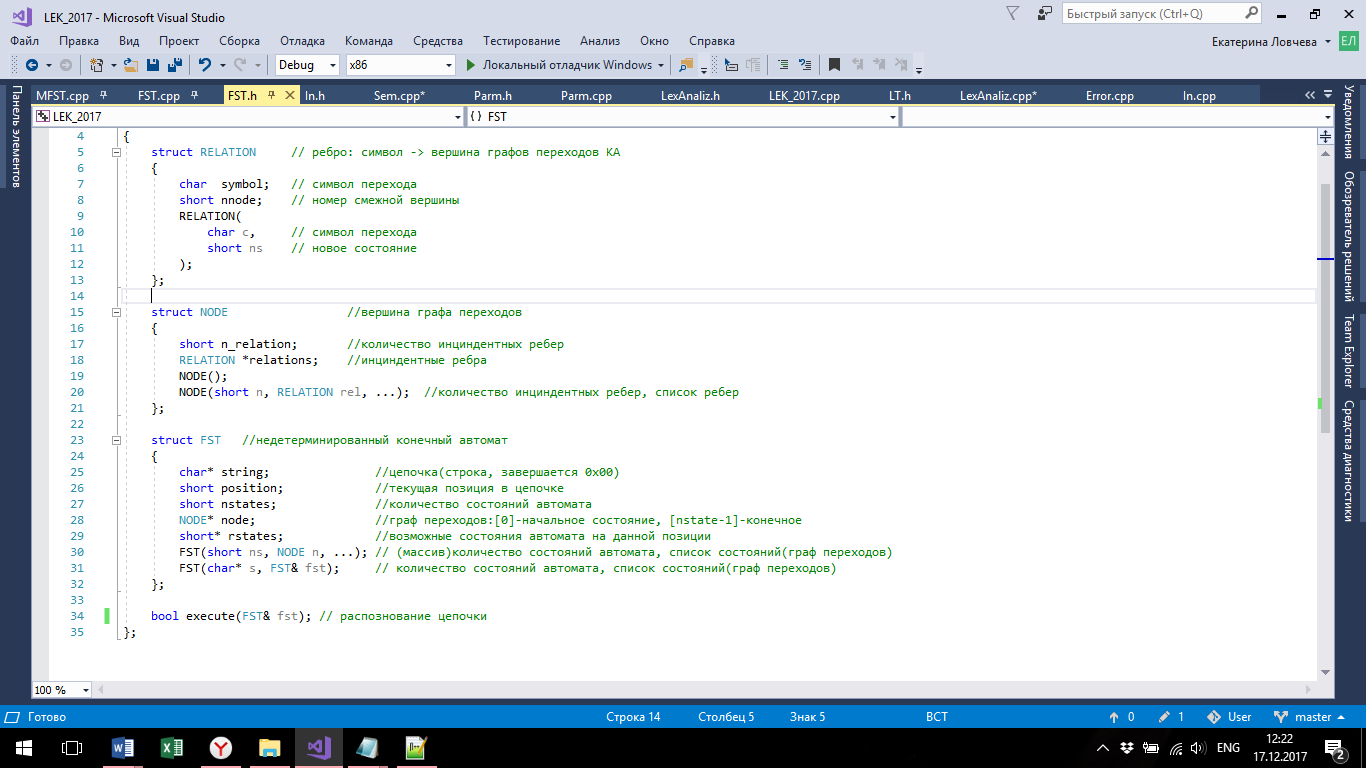
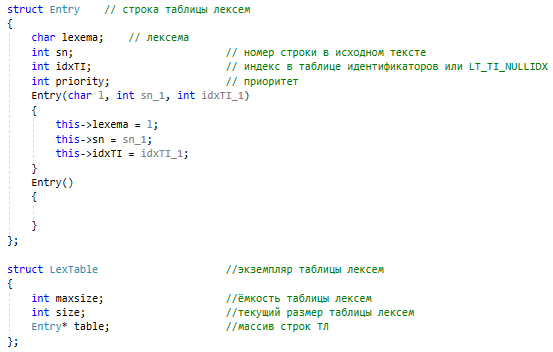
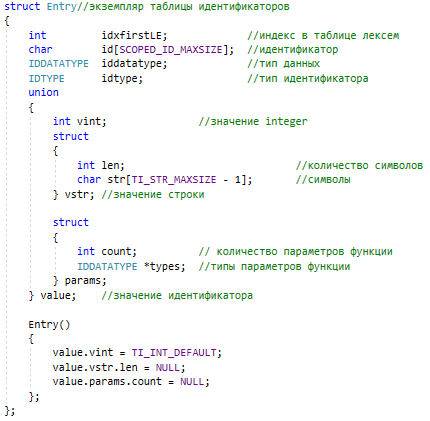


Рис.3 Реализация конечных автоматов





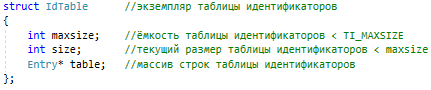
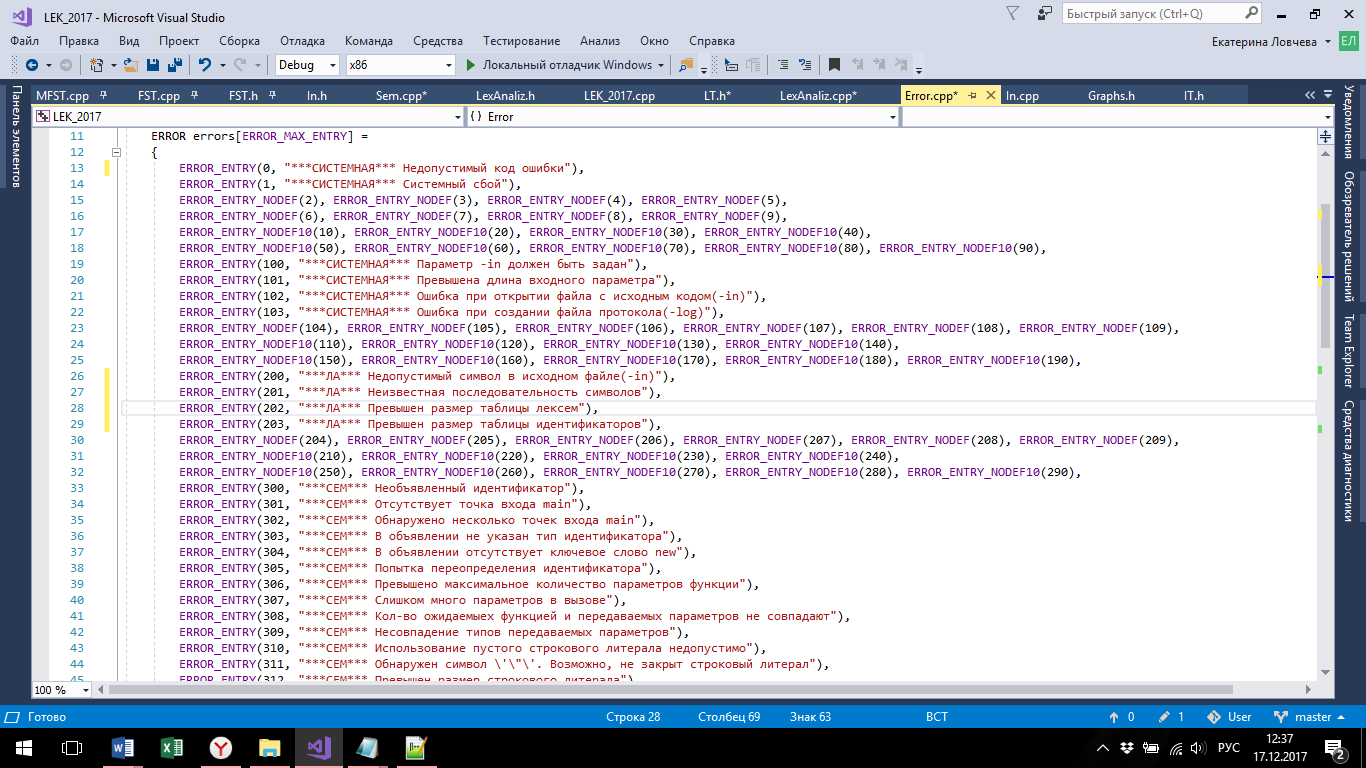


Рис. 4 Основные структуры данных лексиечского анализатора

# Приложение Б



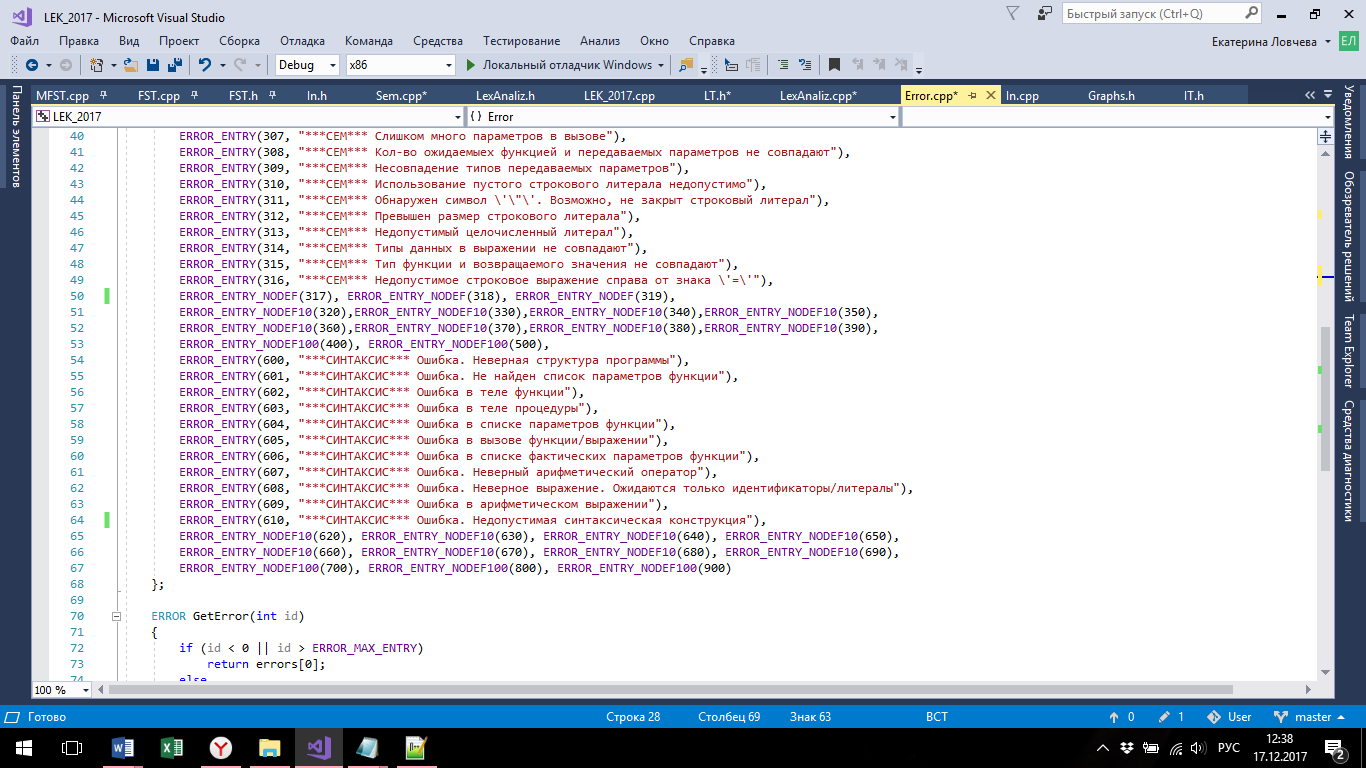
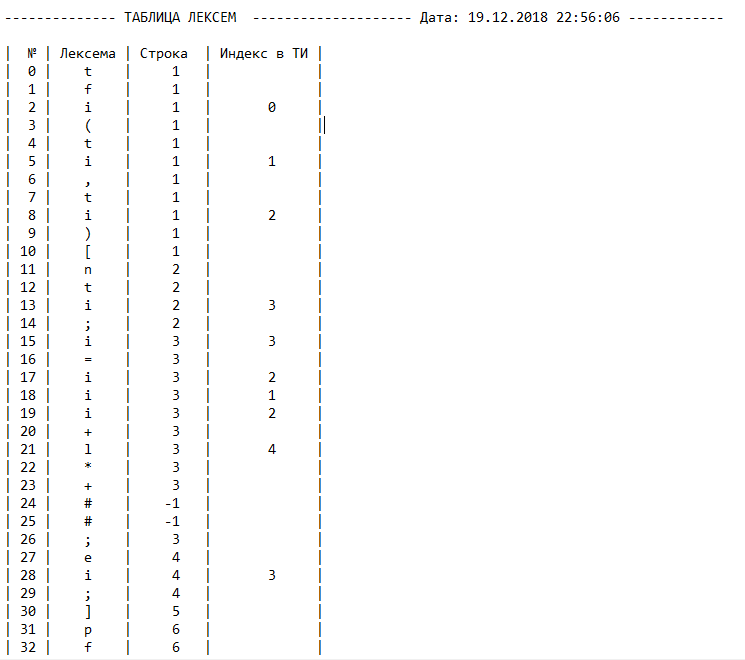


Рис.4 Таблица ошибок языка GDV-2022



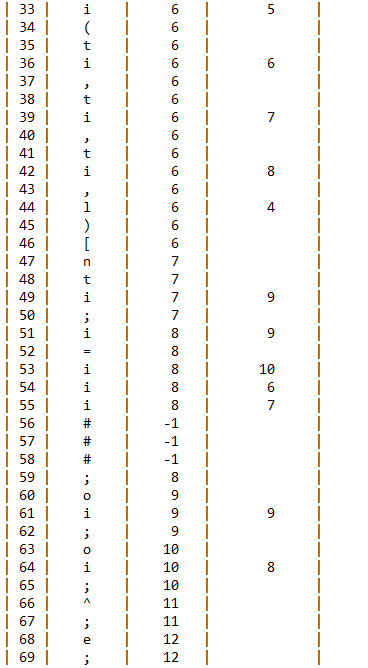
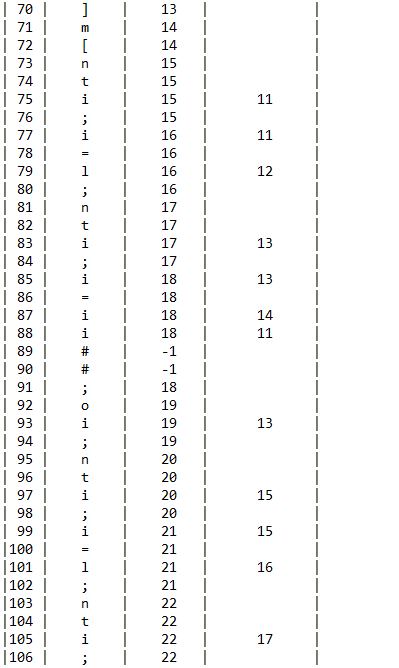


Рис.5 Таблица лексем языка GDV-2022



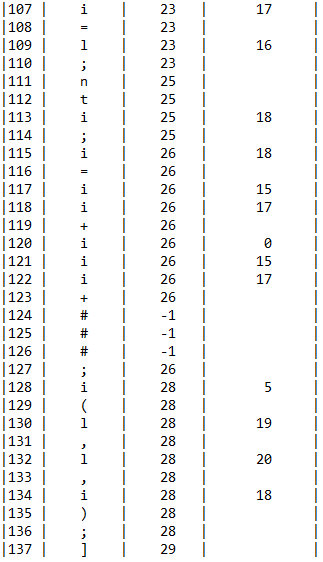


Рис.6 Таблица лексем языка GDV-2022(продолжение)

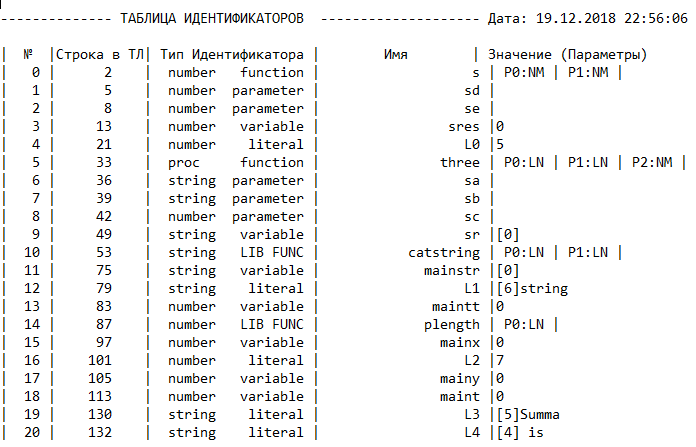
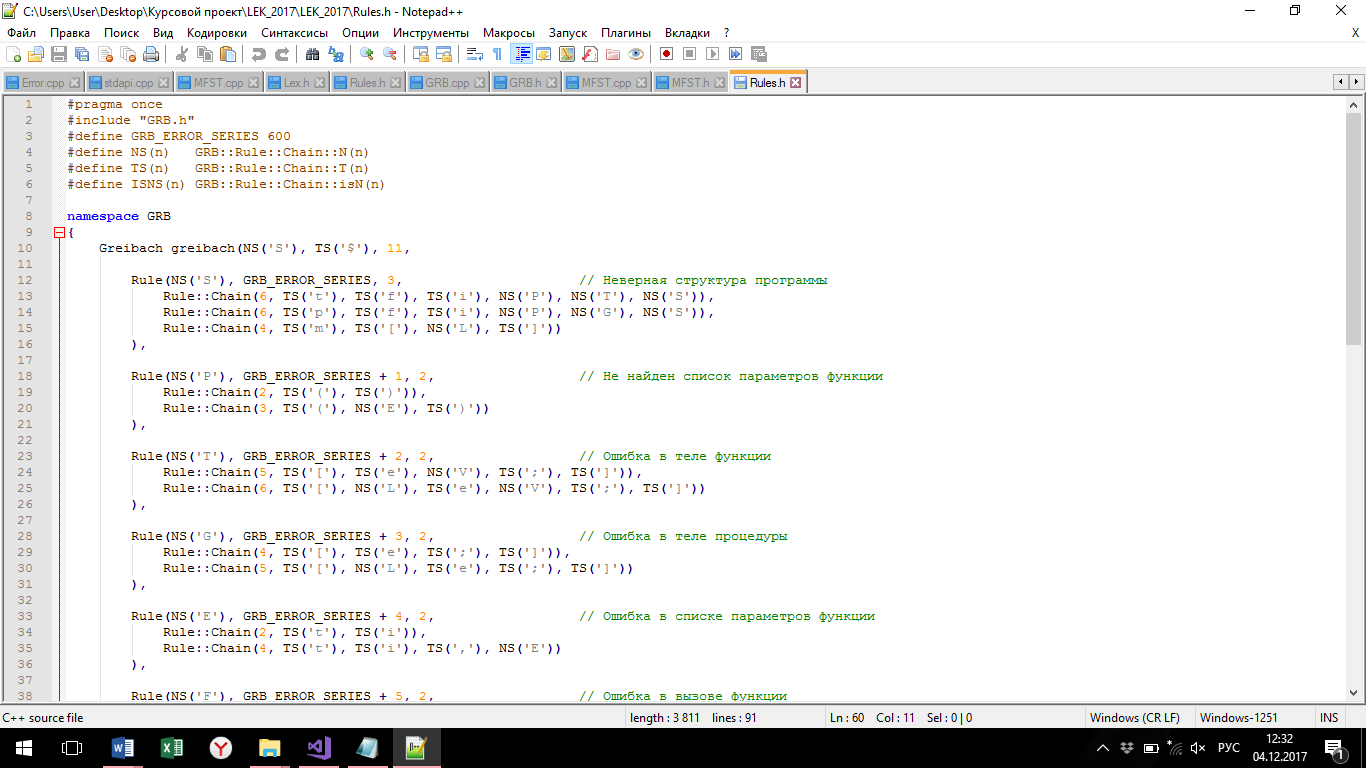
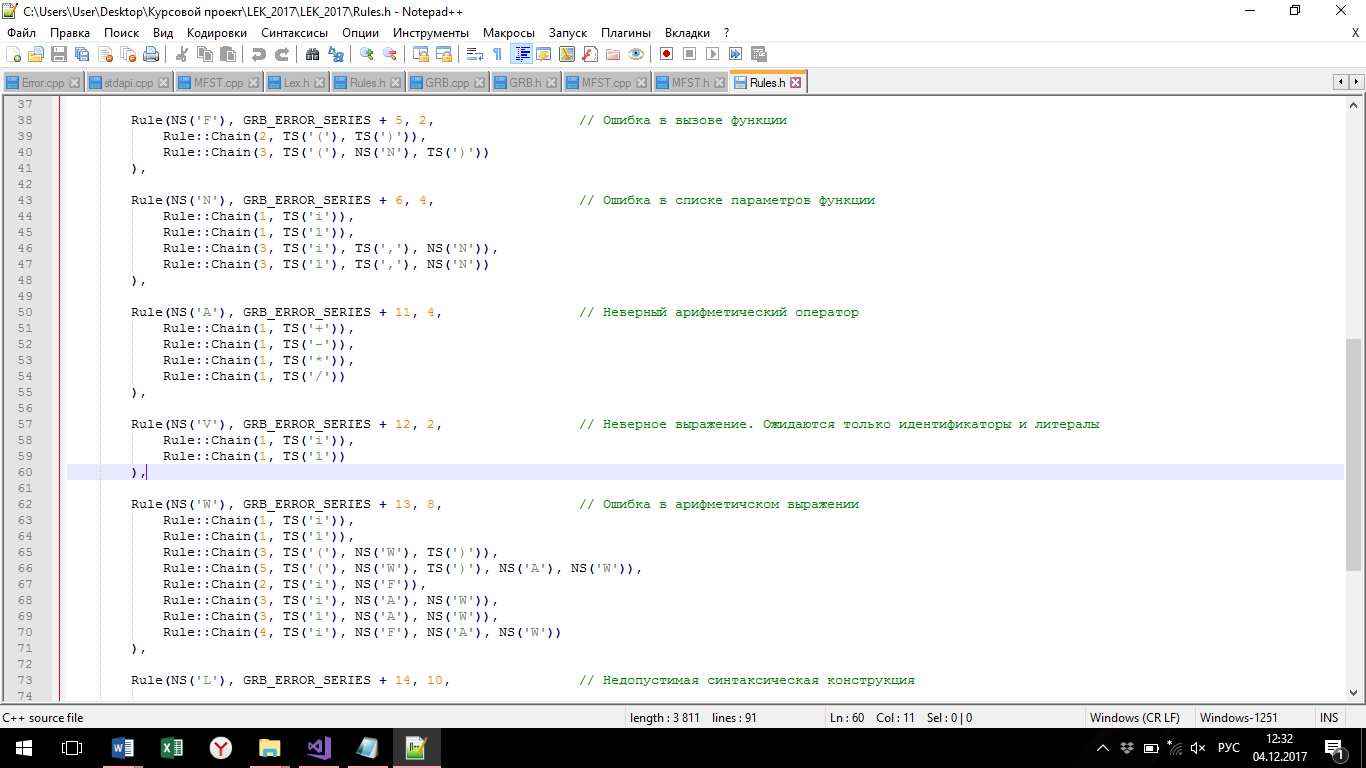


Рис.7 Таблица идентификаторов языка GDV-2022

# Приложение В





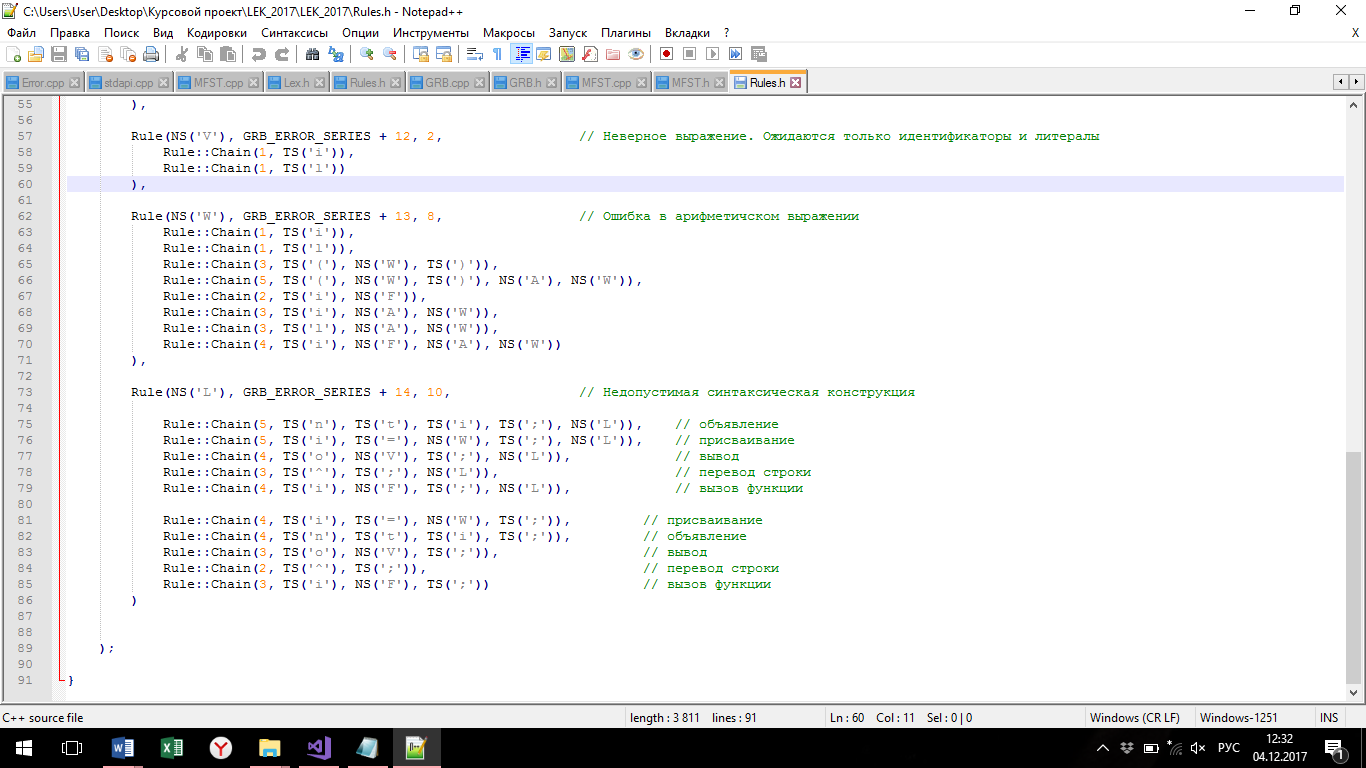


Рис. 8 Структура Грамматики Грейбах

# Приложение Г

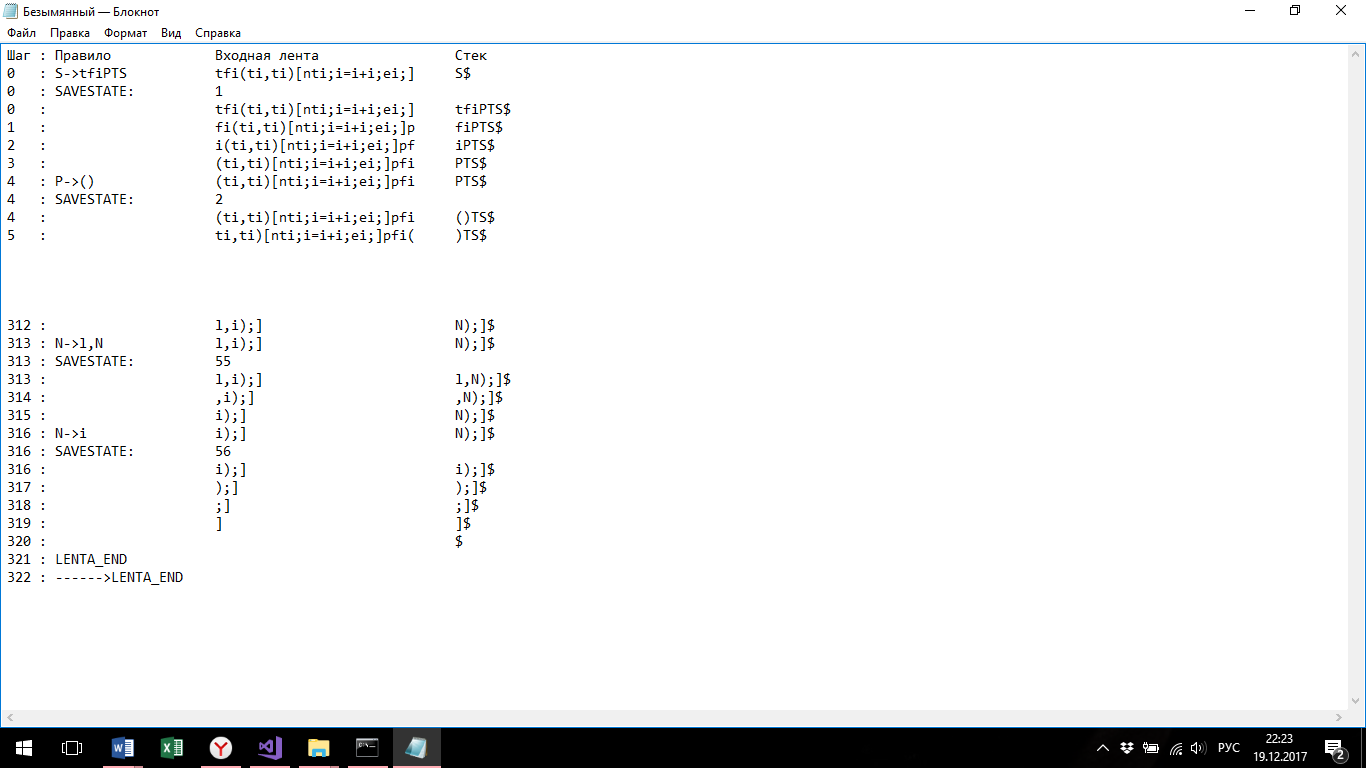


Рис. 125 Разбор исходного кода синтаксическим анализатором

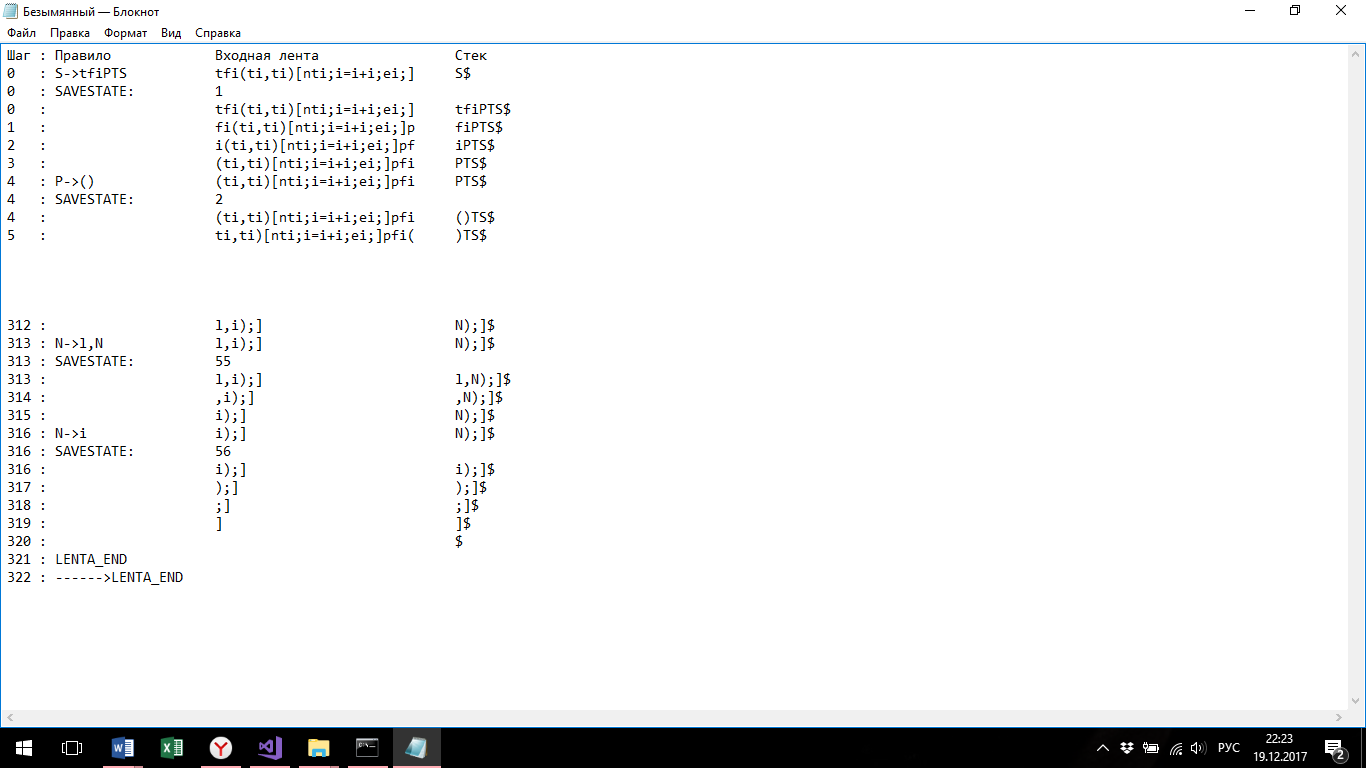
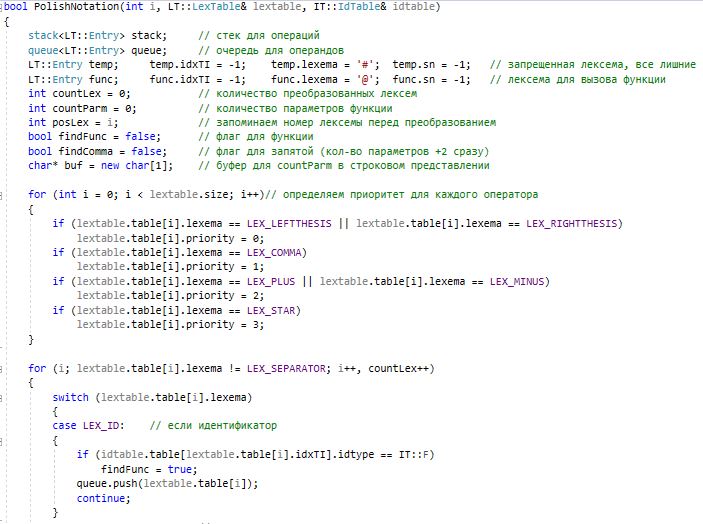


Рис. 125 (продолжение) Разбор исходного кода синтаксическим анализатором(конец)

# Приложение Е



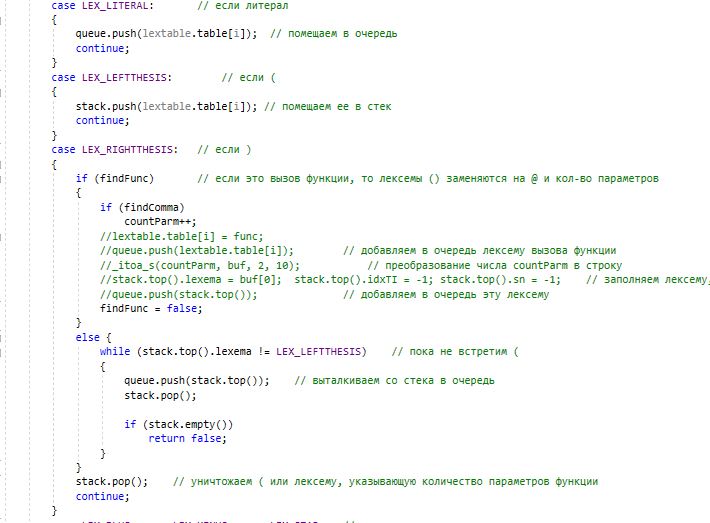
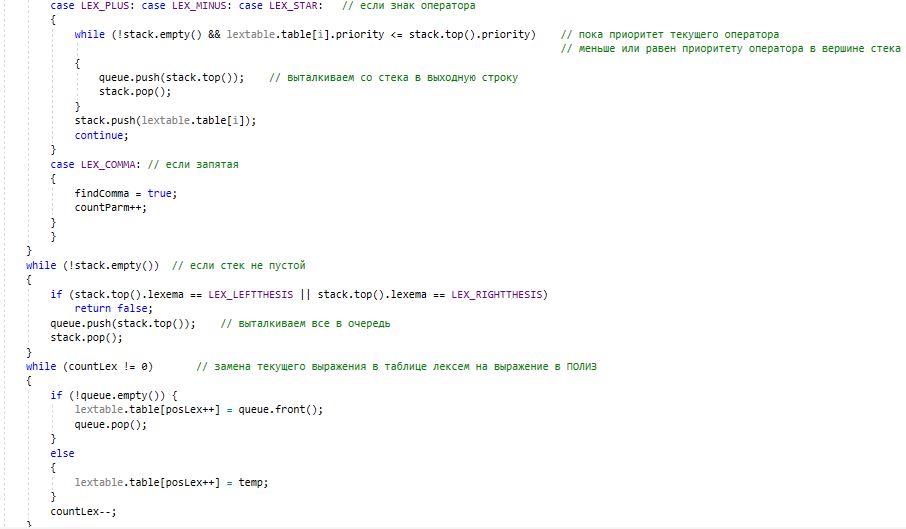


Рис. 1 Преобразование выражений к польской записи



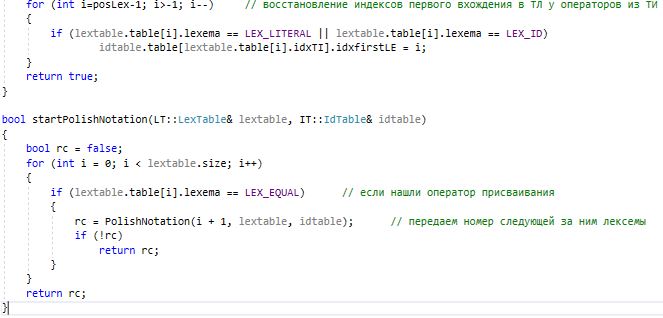


Рис. 1 (Продолжение) Преобразование выражений к польской записи

# Приложение Ж

Таблица 1. Результат генерации кода

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib "..\..\CJA\Debug\LIB.lib"  ExitProcess PROTO :DWORD  .stack 4096  printnum PROTO: DWORD  printstr PROTO : DWORD  catstring PROTO : DWORD, : DWORD, : DWORD  plength PROTO : DWORD, : DWORD  .const  endl byte 13, 10, 0  L0 sdword 5  L1 byte 'string', 0  L2 sdword 7  L3 byte 'Summa', 0  L4 byte ' is ', 0  .data  temp sdword ?  buffer byte 256 dup(0)  sres sdword 0  sr dword ?  mainstr dword ?  maintt sdword 0  mainx sdword 0  mainy sdword 0  maint sdword 0  .code  ;----------- s ------------  s PROC,  sd : sdword, se : sdword  ; --- save registers ---  push ebx  push edx  ; ----------------------  push se  push sd  push se  pop ebx  pop eax  add eax, ebx  push eax  push L0  pop ebx  pop eax  imul eax, ebx  push eax  pop ebx |

Таблица 1. Результат генерации кода (продолжение)

|  |
| --- |
| pop eax  add eax, ebx  push eax  pop ebx  mov sres, ebx  ; --- restore registers ---  pop edx  pop ebx  ; -------------------------  mov eax, sres  ret 8  s ENDP  ;------------------------------  ;----------- three ------------  three PROC,  sa : dword, sb : dword, sc : sdword  ; --- save registers ---  push ebx  push edx  ; ----------------------  push maint  push offset L4  push offset L3  push three  push mainy  push mainx  push s  push mainy  push mainx  push maint  push maint  push L2  push mainy  push mainy  push L2  push mainx  push mainx  push maintt  push mainstr  push plength  push maintt  push maintt  push offset L1  push mainstr  push mainstr  push sc  push sr  push sb  push sa  push offset buffer  call catstring |

Таблица 1. Результат генерации кода (продолжение)

|  |
| --- |
| add esp, 12  mov sr, eax  push sr  call printstr  add esp, 4  push sc  call printnum  add esp, 4  push offset endl  call printstr  add esp, 4  ; --- restore registers ---  pop edx  pop ebx  ; -------------------------  ret 12  three ENDP  ;------------------------------  ;----------- MAIN ------------  main PROC  mov mainstr, offset L1  push maint  push offset L4  push offset L3  push three  push mainy  push mainx  push s  push mainy  push mainx  push maint  push maint  push L2  push mainy  push mainy  push L2  push mainx  push mainx  push maintt  push mainstr  push offset buffer  call plength  add esp, 8  push eax  pop ebx  mov maintt, ebx |

Таблица 1. Результат генерации кода (продолжение)

|  |
| --- |
| push maintt  call printnum  add esp, 4  push L2  pop ebx  mov mainx, ebx  push L2  pop ebx  mov mainy, ebx  push mainx  push mainy  pop ebx  pop eax  add eax, ebx  push eax  push maint  push offset L4  push offset L3  push three  push mainy  push mainx  call s  add esp, 12  push eax  pop ebx  mov maint, ebx  push maint  push offset L4  push offset L3  call three  add esp, 16  push 0  call ExitProcess  main ENDP  end main |

# Заключение

В ходе выполнения курсовой работы был разработан компилятор и генератор кода для языка программирования GDV-2022. Таким образом, были выполнены основные задачи данной курсовой работы:

* Сформулирована спецификация языка GDV-2022;
* Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
* Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
* Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
* Осуществлена программная реализация синтаксического анализатора, позволяющая расширять набор синтаксических конструкций языка только за счёт внесения изменений в разработанную грамматику;
* Разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
* Разработан транслятор кода на язык ассемблера;
* Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка GDV-2022 включает:

* 2 типа данных;
* Поддержка оператора вывода;
* Возможность вызова функций стандартной библиотеки;
* Наличие 3 арифметических операторов для вычисления выражений;
* Структурированная и классифицированная система для обработки ошибок пользователя.

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении компиляторов.

# Литература

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

4. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с