МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора MMV-2020»

Выполнил студент Малиновский Максим Витальевич

(Ф.И.О.)

Руководитель проекта преп.-стаж. Котович Дмитрий Витальевич

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

Заведующий кафедрой к.т.н., доц. Пацей Н.В.

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

Консультанты преп.-стаж. Котович Дмитрий Витальевич

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

Курсовой проект защищен с оценкой

Оглавление

[Введение 5](#_Toc532650587)

[1 Спецификация языка программирования 6](#_Toc532650588)

[1.1 Характеристика языка программирования 6](#_Toc532650589)

[1.2 Определение алфавита языка программирования 6](#_Toc532650590)

[1.3 Применяемые сепараторы 6](#_Toc532650591)

[1.4 Применяемые кодировки 7](#_Toc532650592)

[1.5 Типы данных 7](#_Toc532650593)

[1.6 Преобразование типов данных 8](#_Toc532650594)

[1.7 Идентификаторы 9](#_Toc532650595)

[1.8 Литералы 9](#_Toc532650596)

[1.9 Объявление данных 9](#_Toc532650597)

[1.10 Инициализация данных 1](#_Toc532650598)0

[1.11 Инструкции языка 1](#_Toc532650599)0

[1.12 Операции языка 1](#_Toc532650600)1

[1.13 Выражения и их вычисление 1](#_Toc532650601)2

[1.14 Конструкции языка 1](#_Toc532650602)2

[1.15 Области видимости идентификаторов 1](#_Toc532650603)3

[1.16 Семантические проверки 1](#_Toc532650604)3

[1.17 Стандартная библиотека и её состав 1](#_Toc532650606)4

[1.18 Ввод и вывод данных 1](#_Toc532650607)4

[1.19 Точка входа 1](#_Toc532650608)5

[1.20 Препроцессор 1](#_Toc532650609)5

[1.21 Соглашения о вызовах 1](#_Toc532650610)5

[1.22 Объектный код 1](#_Toc532650611)5

[1.23 Классификация сообщений транслятора 1](#_Toc532650612)5

[1.24 Контрольный пример 1](#_Toc532650613)6

[2 Структура транслятора 14](#_Toc532650614)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 14](#_Toc532650615)

[2.2 Перечень входных параметров транслятора 15](#_Toc532650616)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 15](#_Toc532650617)

[3 Разработка лексического анализатора 17](#_Toc532650618)

[3.1 Структура лексического анализатора 17](#_Toc532650619)

[3.2. Контроль входных символов 18](#_Toc532650620)

[3.3 Удаление избыточных символов 18](#_Toc532650621)

[3.4 Перечень ключевых слов 19](#_Toc532650622)

[3.5 Основные структуры данных 20](#_Toc532650623)

[3.6 Принцип обработки ошибок 22](#_Toc532650624)

[3.7 Структура и перечень сообщений лексического анализатора 22](#_Toc532650625)

[3.8 Параметры лексического анализатора 22](#_Toc532650626)

[3.9 Алгоритм лексического анализа 22](#_Toc532650627)

[3.10 Контрольный пример 23](#_Toc532650628)

[4. Разработка синтаксического анализатора 24](#_Toc532650629)

[4.1 Структура синтаксического анализатора 24](#_Toc532650630)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 24](#_Toc532650631)

[4.3 Построение конечного магазинного автомата 26](#_Toc532650632)

[4.4 Основные структуры данных 27](#_Toc532650633)

[4.5 Описание алгоритма синтаксического разбора 27](#_Toc532650634)

[4.6 Структура и перечень сообщений синтаксического анализатора 28](#_Toc532650635)

[4.7. Параметры синтаксического анализатора и режимы его работы 28](#_Toc532650636)

[4.8. Принцип обработки ошибок 28](#_Toc532650637)

[4.9. Контрольный пример 28](#_Toc532650638)

[5 Разработка семантического анализатора 29](#_Toc532650639)

[5.1 Структура семантического анализатора 29](#_Toc532650640)

[5.2 Функции семантического анализатора 29](#_Toc532650641)

[5.3 Структура и перечень сообщений семантического анализатора 29](#_Toc532650642)

[5.4 Принцип обработки ошибок 30](#_Toc532650643)

[5.5 Контрольный пример 30](#_Toc532650644)

[6. Вычисление выражений 31](#_Toc532650645)

[6.1 Выражения, допускаемые языком 31](#_Toc532650646)

[6.2 Польская запись и принцип её построения 31](#_Toc532650647)

[6.3 Программная реализация обработки выражений 31](#_Toc532650648)

[6.4 Контрольный пример 32](#_Toc532650649)

[7. Генерация кода 33](#_Toc532650650)

[7.1 Структура генератора кода 33](#_Toc532650651)

[7.2 Представление типов данных в оперативной памяти 33](#_Toc532650652)

[7.3 Статическая библиотека 34](#_Toc532650653)

[7.4 Особенности алгоритма генерации кода 34](#_Toc532650654)

[7.5 Входные параметры генератора кода 34](#_Toc532650655)

[7.6 Контрольный пример 34](#_Toc532650656)

[8. Тестирование транслятора 36](#_Toc532650657)

[8.1 Тестирование проверки на допустимость символов 36](#_Toc532650658)

[8.2 Тестирование лексического анализатора 36](#_Toc532650659)

[8.3 Тестирование синтаксического анализатора 36](#_Toc532650660)

[8.4 Тестирование семантического анализатора 37](#_Toc532650661)

[Заключение 39](#_Toc532650662)

# 

# Введение

В данном курсовом проекте поставлена задача разработки собственного языка программирования и транслятора для него. Название языка – MMV-2020. Написание транслятора будет осуществляться на языке C++, при этом код на языке MMV-2020 будет транслироваться в язык С#.

Задание на курсовой проект можно разделить на следующие задачи:

1. Разработка спецификации языка MMV-2020;
2. Разработка лексического анализатора;
3. Разработка синтаксического анализатора;
4. Разработка семантического анализатора;
5. Разбор арифметических выражений;
6. Разработка генератора кода;
7. Тестирование транслятора.

# 1 Спецификация языка программирования

## 1.1 Характеристика языка программирования

Язык MMV-2020 – это универсальный, строго типизированный, интерпритируемый язык, не имеющий элементов ООП.

## 1.2 Определение алфавита языка программирования

В алфавите языка MMV-2020 используется кодировка ASCII, таблица которой представлена на рисунке 1.1.



Рисунок 1.1 Алфавит входных символов

Символы, используемые на этапе выполнения: [a…z], [A…Z], [0…9], символы пробела, табуляции и перевода строки, спецсимволы: [ ] ( ) , ; {} + - / \* > <, &, ., `,~, %, ?, #.

## 1.3 Применяемые сепараторы

Символы-сепараторы служат в качестве разделителей цепочек языка во время обработки исходного текста программы с целью разделения на токены. Они представлены в таблице 1.1.

Таблица 1.1 Символы-сепараторы

|  |  |
| --- | --- |
| Символ(ы) | Назначение |
| ‘пробел’ | Разделитель цепочек. Допускается везде кроме названий идентификаторов и ключевых слов |
| **{...}** | Блок функции |
| **(...)** | Приоритет арифметических операций |
| **&** | Разделитель параметров функций |
| **+ - \*/** | Арифметические операции |
| **> < ` ~ # ?** | Логические операции (операции сравнения: больше, меньше, меньше либо равно, больше либо равно, проверка на равенство, на неравенство), используемые в условной конструкции. |
| **.** | Разделитель программных конструкций |
| **=** | Оператор присваивания |
| **[...]** | Блок фактических или формальных параметров функции |

## 1.4 Применяемые кодировки

Для написания программ язык MMV-2020 использует кодировку ASCII, содержащую английский алфавит, а также некоторые специальные символы, такие как [ ] ( ) , ; {} + - / \* > <, &, ., `,~, %., ?, #.

## 1.5 Типы данных

В языке MMV-2020 реализованы два фундаментальных типа данных: целочисленный и строковый. Описание типов приведено в таблице 1.2.

Таблица 1.2 Типы данных языка MMV-2020

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| 1 | 2 |
| Целочисленный тип данных **int** | Фундаментальный тип данных. Используется для работы с числовыми значениями. В памяти занимает 1 байта. Максимальное значение: 127. |
| 1 | 2 |
|  | Минимальное значение: -128.  Инициализация по умолчанию: значение 0.  Поддерживаемые операции:  **+** (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  **\*** (бинарный) – оператор умножения;  **/** (бинарный) – оператор деления;  **=** (бинарный) – оператор присваивания;  Библиотечные функции.  В качестве условия условного оператора поддерживаются следующие логические операции:  **>** (бинарный) – оператор «больше»;  **<** (бинарный) – оператор «меньше»;  ? (бинарный) – оператор проверки на равенство;  ` (бинарный) – оператор «меньше либо равно»;  ~ (бинарный) – оператор «больше либо равно»;  # (бинарный) – оператор проверки на неравенство. |
| Строковый тип данных **str** | Фундаментальный тип данных. Используется для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 255.  Инициализация по умолчанию: строка нулевой длины “”. |

## 1.6 Преобразование типов данных

Преобразование типов данных в языке MMV-2020 не предусмотрено. Попытка преобразования типов данных приведет к семантической ошибке.

## 1.7 Идентификаторы

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Идентификаторы должны содержать только символы нижнего регистра латинского алфавита. Максимальная длина идентификатора равна пяти символам. Идентификаторы не должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, имя функции, параметр функции.

## 1.8 Литералы

С помощью литералов осуществляется инициализация переменных. Типы литералов языка MMV-2020 представлены в таблице 1.3.

Таблица 1.3 Литералы

|  |  |
| --- | --- |
| Литералы | Пояснение |
| Целочисленные литералы | Последовательность цифр 0…9 с предшествующим знаком минус или без него. |
| Строковые литералы | Набор символов (от 1 до 255), заключённых в двойные кавычки |

Ограничения на строковые литералы языка MMV-2020: внутри литерала не допускается использование одинарных и двойных кавычек. Ограничения на целочисленные литералы: не могут начинаться с нуля, если их значение не ноль.

## 1.9 Объявление данных

Для объявления переменной используется ключевое слово **var**, после которого указывается тип данных и имя идентификатора. Допускается инициализация при объявлении.

Пример объявления числового типа с инициализацией:

**var int** num = -1

Пример объявления переменной символьного типа с инициализацией:

**var** **str** str1= “hello world”

Для объявления функций используется ключевое слово **function**, после которого указывается тип функции, а после – имя функции. Далее список параметров и тело функции.

## 1.10 Инициализация данных

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть только идентификаторы или литералы. При объявлении без инициализации предусмотрены значения по умолчанию: значение 0 для типа **int** и строка длины 0 (“”) для типа **str**.

## 1.11 Инструкции языка

Инструкции языка MMV-2020 представлены в таблице 1.4.

Таблица 1.4 ИнструкцииязыкаMMV-2020

|  |  |
| --- | --- |
| Инструкция | Реализация |
| 1 | 2 |
| Объявление переменной | **var** <тип данных> <идентификатор>. |
| Объявление переменной с явной инициализацией | **var** <тип данных> <идентификатор> = <значение>.  Значение – инициализатор конкретного типа. Может быть только литералом или идентификатором |
| Возврат из функции или процедуры | Для функций, возвращающих значение:  **return** [идентификатор/литерал]. |
| Вывод данных | **output [**идентификатор/литерал]. |
| Вызов функции | <идентификатор функции>[(<список параметров>]. |
| 1 | 2 |
| Присваивание | <идентификатор> **=** <выражение>.  Выражением может быть идентификатор, литерал, или вызов функции соответствующего типа. Для целочисленного типа выражение может быть дополнено арифметическими операциями с любым количеством операндов с использованием скобок. Для строкового типа выражение может быть только идентификатором, литералом или вызовом функции, возвращающей значение строкового типа. |

## 1.12 Операции языка

В языке MMV-2020 предусмотрены следующие операции с данными. Приоритетность операции умножения и деления выше приоритета операций сложения и вычитания. Для установки наивысшего приоритета используются круглые скобки. Операции языка представлены в таблице 1.5.

Таблица 1.5 Операции языка MMV-2020

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические | 1. **+ –** сложение 2. - – вычитание 3. \* – умножение 4. **/** – деление 5. % - остаток от деления 6. = – присваивание |
| Строковые | 1. **=** – присваивание 2. + - конкатенация |
| Логические | 1. **>** – больше  2. **<** – меньше  3. ? – проверка на равенство  4. # – проверка на неравенство  5. ` меньше либо равно  6. ~ больше либо равно |

## 1.13 Выражения и их вычисление

Вычисление выражений – одна из важнейших задач языков программирования. Всякое выражение составляется согласно следующим правилам:

1. Допускается использовать скобки для смены приоритета операций;
2. Выражение записывается в строку без переносов;
3. Использование двух подряд идущих операторов не допускается;
4. Допускается использовать в выражении вызов функции, вычисляющей и возвращающей целочисленное значение.

## 1.14 Конструкции языка

Программа на языке MMV-2020 оформляется в виде функций пользователя и главной функции.

Программные конструкции языка представлены в таблице 1.6.

Таблица 1.6 Программные конструкции языка MMV-2020

|  |  |
| --- | --- |
| Конструкция | Реализация |
| 1 | 2 |
| Главная функция | **head**  **{**  …  } |
| Внешняя функция | **function** <тип данных> <идентификатор>  [<тип> <идентификатор>, …]  **{**…  **return** [идентификатор/литерал];  **}** |
| 1 | 2 |
| Условная конструкция | **check[** <идентификатор1> <оператор> <идентификатор2>]  { … }  **otherwise**  { … }  <идентификатор1>, <идентификатор2> - идентификаторы или литералы целочисленного типа (но не два литерала одновременно). <оператор> - один из операторов сравнения, устанавливающий отношение между двумя операндами и организующий условие данной конструкции. |

## 1.15 Области видимости идентификаторов

Область видимости: сверху вниз (как и в С++). Переменные, объявленные в одной функции, недоступны в другой. Все объявления и операции с переменными происходят внутри какого-либо блока. Внутри условной конструкции объявление переменных запрещено.

Все идентификаторы являются локальными и обязаны быть объявленными внутри какой-либо функции. Глобальных переменных нет. Параметры видны только внутри функции, в которой объявлены.

## 1.16 Семантические проверки

В языке программирования MMV-2020 выполняются следующие семантические проверки:

1. Закрытие строкового литерал

2)Отсутствует точка входа Head

3)Несколько точек входа Head

4)Превышение размера строкового литерала

5)Объявление переменной без ключевого слова var

6)Использование необъявленного идентификатора

7)Объявление переменной без указания типа

8)Попытка реализовать уже существующую функцию

9)Попытка переопределить параметр или ошибка области видимости

10)Попытка переопределить переменную или ошибка области видимости

11)Не указан тип функции

12)Использование пустого строкового литерала

13)Тип функции и тип возвращаемого значения должны совпадать

14)Превышение максимально допустимого (2) количество параметров функции

15)Несовпадение типов передаваемых параметров

16)Слишком много аргументов в вызове функции

17)Слишком мало аргументов в вызове функции

18)Несовместимые типы при присваивании

19)Недопустимый целочисленный литерал

20)Типы данных в выражении не совпадают

21)Проверка употребления арифметических операторов

22)Проверка употребления логических операторов

В языке MMV-2020 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержимое библиотеки и описание функций представлено в таблице 1.8.

Таблица 1.8 Стандартная библиотека языка MMV-2020

|  |  |
| --- | --- |
| Функция | Описание |
| MaxNum[num1& num2]. | Целочисленная функция. Возвращает максимальное из двух значение. |
| Pow[num1 & num2] | Целочисленная функция. Возводит num1 в степень num2. |

Стандартная библиотека написана на языке С#, подключается к транслированному коду на этапе генерации кода. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций.

## 1.18 Ввод и вывод данных

Вывод данных осуществляется с помощью оператора **output**. Допускается использование оператора **output** с литералами и идентификаторами.

## 1.19 Точка входа

В языке MMV-2020 каждая программа должна содержать главную функцию (точку входа) **head**, с первой инструкции которой начнётся последовательное выполнение команд программы.

## 1.20 Препроцессор

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке MMV-2020 отсутствует.

## 1.21 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.

Таблица 1.10 Классификация ошибок

|  |  |
| --- | --- |
| Номера ошибок | Характеристика |
| 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 |

## 1.24 Контрольный пример

Контрольный пример демонстрирует главные особенности языка MMV-2020: его фундаментальные типы, основные структуры, функции, процедуры, использование функций статической библиотеки. Исходный код контрольного примера представлен в example.txt.

# 2 Структура транслятора

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Исходный код, написанный на языке программирования MMV-2020, является для транслятора входными данными.

Как выходные данные используется объектный код и протоколы работы транслятора, описанные в пункте 2.3.

Компоненты транслятора приведены на рисунке 1.



Рис 1 - Структура транслятора MMV-2020

Первоначально на вход лексического анализатора передается исходный код. Анализатором проверяется исходный текст на недопустимые символы, выделяет литералы, идентификаторы и ключевые слова, а также формирует таблицы лексем и идентификаторов.

Далее наступает черед синтаксического анализатора, к нему на вход поступа-ет таблица лексем, полученная на этапе лексического анализа. Если программа по-строена синтаксически правильно, то осуществляется переход к этапу трансляции стоящему далее, при ином раскладе работа транслятора останавливается.

Наборы функций, проверяющие правила на разных этапах работы транслято-ра представлены в семантическом анализаторе. Продолжение или остановка работы транслятора всецело зависит от критичности возникающих ошибок.

Генерация кода реализуется посредством чистой интерпретации, без создания промежуточного представления кода. В финале происходит генерация кода, во время исполнения которого формируется объектный код.

## 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 MMV-2020

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке MMV-2020 , 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 С#. | Значение по умолчанию:  <имя in-файла>.cs |

## 2.3 Перечень протоколов, формируемых транслятором и их содержимое

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 MMV-2020

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования MMV-2020 . Содержит таблицу лексем и таблицу идентификаторов, протокол работы синтаксического анализатора и дерево разбора, полученные на этапе лексического и синтаксического анализа, а также результат работы алгоритма преобразования выражений к польской записи. |
| Выходной файл, заданный параметром "-out:" | Результат работы программы – файл, содержащий исходный код на языке C#. |

# 3 Разработка лексического анализатора

## 3.1 Структура лексического анализатора

Исходный код на языке MMV-2020 является входными данными;

Таблицы лексем и идентификаторов являются выходными данными;



Рисунок 3.1 Структура лексического анализатора

## 3.2. Контроль входных символов

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

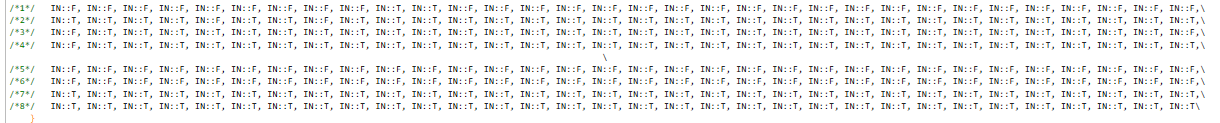


Рисунок 3.2 Таблица контроля входных символов

Таблица 3.1 Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенный | T |
| Запрещенный | F |

## 3.3 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на лексемы перед лексическим анализом.

## 3.4 Перечень ключевых слов

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

Таблица 3.2 Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| var, str | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 8 символов. |
| Литерал | l | Литерал любого доступного типа. |
| function | f | Объявление функции. |
| return | r | Выход из функции. |
| head | h | Главная функция. |
| var | v | Объявление переменной. |
| output | o | Вывод данных. |
| check | с | Указывает начало условного оператора. |
| otherwise | u | Ложная ветвь условного оператора |
| NewLine | n | Оператор вывода символа перевода строки. |
| . | . | Разделение выражений. |
| & | & | Разделение параметров функций. |
| { | { | Начало блока/тела функции. |
| } | } | Закрытие блока/тела функции. |
| [ | [ | Передача параметров в функцию |
| ] | ] | Закрытие блока для передачи параметров |
| = | = | Знак присваивания. |
| +  -  \*  /  % | +  -  \*  /  % | Знаки операций. |
| >  <  ?  `  ~  # | >  <  &  !  ~  # | Знаки логических операторов |

## 3.5 Основные структуры данных

В приложении Б представлены основные структуры данных на этапе синтаксического анализа.

## 3.6 Принцип обработки ошибок

При возникновении ошибки происходит запись этой ошибки в log-файл и прекращения работы транслятора.

## 3.7 Структура и перечень сообщений лексического анализатора

Перечень сообщений, формируемых лексическим анализатором в ходе своей работы, представлен в таблице 3.2.

Таблица 3.2 – Перечень сообщений лексического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 201 | Превышен размер таблицы лексем |
| 202 | Переполнение таблицы лексем |
| 203 | Превышен размер таблицы идентификаторов |
| 204 | Переполнение таблицы идентификаторов |
| 205 | Неизвестная последовательность символов |

## 3.8 Параметры лексического анализатора

## Текст кода на языке MMV-2020 подается на вход. Параметры не определяют режим работы лексического анализатора.

## 3.9 Алгоритм лексического анализа

Лексический анализ – является первой и наиболее простой фазой трансляции. Алгоритм лексического анализатора заключается в распознавании и разборе цепочек исходного кода. Распознавание цепочек основывается на работе конечных автоматов.

Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «output» представлен на рисунке 3.2, где S0 – начальное, а S6 – конечное состояние автомата.

o u t p u t

S6

S5

S4

S3

S1

S0

S2

Рисунок 3.2 Пример графа переходов для цепочки string

## 3.10 Контрольный пример

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

# 4. Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора

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

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

Рисунок 4.1 Структура синтаксического анализатора.

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

Грамматика, описывающая язык MMV-2020 представлена в таблице 4.1.

Таблица 4.1 - грамматика языка MMV-2020

|  |  |
| --- | --- |
| Нетереминалы | Описание |
| S | Правила, описывающие общую структуру программы |
| F | Порождает правила, описывающие формальные параметры функции |
| P | Порождает правила, описывающие формальные параметры функции при её объявлении |
| B | Порождает правила, описывающие тело функции |
| I | Порождает правила, описывающие литералы и идентификаторы |
| N | Порождает правила, описывающие инструкции языка |
| R | Порождает правила, описывающие логические действия |
| K | Порождает правила, описывающие вызов функции |
| E | Порождает правила, описывающие выражения |
| M | Порождает правила, описывающие арифметические действия |
| X | Порождает правила, описывающие конструкция в теле условного выражения |

## 4.3 Построение конечного магазинного автомата

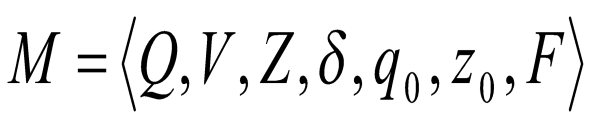
Конечный автомат с магазинной памятью представляет собой семерку. Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит представляет из себя множества терминальных и нетерминальных символов, описание которых содержится в таблица 3.1 и 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $) |
|  | 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 |
|  | 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 представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка MMV-2020 . Данные структуры в приложении В.

## 4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата следующий:

1. В магазин записывается стартовый символ;
2. На основе полученных ранее таблиц формируется входная лента;
3. Запускается автомат;
4. Выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
5. Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и 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, формируемых синтаксическим анализатором в ходе своей работы, представлен в таблице 4.3.

Таблица 4.3 – Перечень сообщений лексического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 600 | Неверная структура программы |
| 601 | Отсутствует список параметров функции при её объявлении |
| 602 | Ошибка в параметрах функции при её объявлении |
| 603 | Возможно отсутствует тело функции |
| 604 | Недопустимое выражение. Ожидаются только литералы и идентификаторы |
| 605 | Неверная конструкция в теле функции |
| 606 | Ошибка в условном выражении |
| 607 | Ошибка в вызове функции |
| 608 | Ошибка в арифметическом выражении |
| 609 | Ошибка в списке параметров при вызове функции |
| 610 | Неверная конструкция в условном выражении |
| 611 | Требуется закрывающаяся фигурная скобка |
| 612 | Требуется открывающаяся фигурная скобка |

## 4.7. Параметры синтаксического анализатора и режимы его работы

Входной информацией для синтаксического анализатора является таблица лексем и идентификаторов. Кроме того используется описание грамматики в форме Грейбах. Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью выводятся в журнал работы программы.

## 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.

## 4.9. Контрольный пример

Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью приведены в приложении В.

# 5 Разработка семантического анализатора

## 5.1 Структура семантического анализатора

Семантический анализатор состоит из набора функций для проверки правильности исходной программы. Функции анализатора выполняются на различных этапах работы транслятора. Структура семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1. Структура семантического анализатора

## 5.2 Функции семантического анализатора

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

## 5.3 Структура и перечень сообщений семантического анализатора

Перечень сообщений, формируемых семантического анализатором в ходе своей работы, представлен в таблице 5.1.

Таблица 5.1 – Перечень сообщений сементического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 300 | Неверная структура программы |
| 301 | Отсутствует список параметров функции при её объявлении |
| 302 | Ошибка в параметрах функции при её объявлении |
| 303 | Возможно отсутствует тело функции |
| 304 | Недопустимое выражение. Ожидаются только литералы и идентификаторы |
| 305 | Неверная конструкция в теле функции |
| 306 | Ошибка в условном выражении |
| 307 | Ошибка в вызове функции |
| 308 | Ошибка в арифметическом выражении |
| 309 | Ошибка в списке параметров при вызове функции |
| 310 | Неверная конструкция в условном выражении |
| 311 | Требуется закрывающаяся фигурная скобка |
| 312 | Требуется открывающаяся фигурная скобка |
| 313 |  |
| 314 |  |
| 315 |  |

## 5.4 Принцип обработки ошибок

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

## 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

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| main[  number x = 9;  Write x;  ] | Ошибка N304: Семантическая ошибка: В объявлении отсутствует ключевое слово new Строка: 2 |
| main[  new number x = 9;  new string y =x;  ] | Ошибка N314: Семантическая ошибка: Типы данных в выражении не совпадают Строка: 3 |
| main[  new number x = 9;  ]  main[  new string y = "qwerty";  ] | Ошибка N302: Семантическая ошибка: Обнаружено несколько точек входа main Строка: 0 |

# 6. Вычисление выражений

## 6.1 Выражения, допускаемые языком

В языке PAA-2018 допускаются вычисления выражений целочисленного типа данных с поддержкой вызова функций внутри выражений. Приоритет операций представлен на таблице 6.1.

Таблица 6.1. Приоритеты операций

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

## 6.2 Польская запись и принцип её построения

Все выражения языка PAA-2018 преобразовываются к обратной польской записи.

Польская запись - это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: прямая и обратная, также известные как префиксная и постфиксная. Отличие их от классического, инфиксного способа заключается в том, что знаки операций пишутся не между, а, соответственно, до или после аргументов. Алгоритм построения польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* 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.

## 6.3 Программная реализация обработки выражений

Программная реализация алгоритма преобразования выражений к польской записи представлена в приложении Г.

## 6.4 Контрольный пример

Пример преобразования выражений из контрольных примеров к обратной польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления и преобразования к ассемблерному коду. В приложении Г приведены изменённые таблицы лексем и идентификаторов, отображающие результаты преобразования выражений в польский формат.

Таблица 6.2. Преобразование выражений к ПОЛИЗ

|  |  |
| --- | --- |
| Выражение | Обратная польская запись для выражения |
| i[2]=(((l[3]+l[4])-i[0])\*l[5])/l[6]; | i[2]=l[3]l[4]+i[0]-l[5]\*l[6]/ |
| i[23]=(i[23]+l[26])\*l[26] | i[23]=i[23]l[26]+l[26]\* |
| i[3]=(((l[4]+l[5])-i[0])\*l[6]) | i[3]=l[4]l[5]+i[0]-l[6]\* |

# 7. Генерация кода

## 7.1 Структура генератора кода

В языке PAA-2018 генерация кода является заключительным этапом трансляции. 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 PAA-2018 представлена на рисунке 7.1.



Рисунок 7.1 – Структура генератора кода

## 7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке PAA-2018 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка PAA-2018 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке PAA-2018 | Тип идентификатора на языке ассемблера | Пояснение |
| number | sdword | Хранит целочисленный тип данных. |
| string | dword | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |

## 7.3 Статическая библиотека

В языке PAA-2018 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 |
| void outlstr(char\* str) | Вывод на консоль строки str |
| void outnum(int num) | Вывод на консоль целочисленной переменной num |
| int lenght(char\* buffer, char\* str) | Вычисление длины строки |
| char\* concat(char\* buffer, char\* str1, char\* str2) | Объединение строк str1 и str2 |

## 7.4 Особенности алгоритма генерации кода

В языке PAA-2018 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке

Рисунок 7.2 – Структура генератора кода

## 7.5 Входные параметры генератора кода

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке PAA-2018. Результаты работы генератора кода выводятся в файл с расширением .asm.

## 7.6 Контрольный пример

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Д. Результат работы контрольного примера приведён на рисунке 7.2.

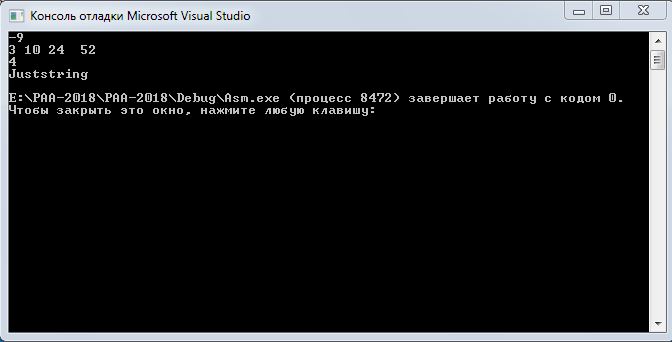


Рисунок 7.2 Результат работы программы на языке PAA-2018

# 8. Тестирование транслятора

## 8.1 Тестирование проверки на допустимость символов

В языке PAA-2018 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 |
| main [ё] | Ошибка N200: Лексическая ошибка: Недопустимый символ в исходном файле(-in) Строка: 2 Позиция в строке: 2 |

## 8.2 Тестирование лексического анализатора

На этапе лексического анализа в языке PAA-2018 могут возникнуть ошибки, описанные в пункте 3.7. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 - Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  [new number x11;] | Ошибка N201: Лексическая ошибка: Неизвестная последовательность символов Строка: 3 |

## 8.3 Тестирование синтаксического анализатора

На этапе синтаксического анализа в языке PAA-2018 могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 - Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main new number x; ] | Ошибка 600: строка 1, Синтаксическая ошибка: Неверная структура программы |
| string function fi([]  main[] | Ошибка 601: строка 1, Синтаксическая ошибка: Не найден список параметров функции |
| string function fi()  [newline; write]  main[] | Ошибка 602: строка 1, Синтаксическая ошибка: Ошибка в теле функции |
| procedure function fi()  [newline write;]  main[] | Ошибка 603: строка 1, Синтаксическая ошибка: Ошибка в теле процедуры |

Продолжение таблицы 8.3

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| procedure function fi(number number)[]  main[] | Ошибка 604: строка 1, Синтаксическая ошибка: Ошибка в списке параметров функции |
| string function fi(number x)[return 3;]  main [newline;fi(5,5; newline;] | Ошибка 605: строка 2, Синтаксическая ошибка: Ошибка в вызове функции/выражении |
| string function fi(number x)[return 3;]  main [newline;fi(5,5,5 5);] | Ошибка 606: строка 2, Синтаксическая ошибка: Ошибка в списке фактических параметров функции |
| main [new number x; condition: x > 2 # cycle #] | Ошибка 607: строка 1, Синтаксическая ошибка: Ошибка при конструировании цикла/условного выражения |
| main [new number x; condition: x > 2 # cycle #] | Ошибка 608: строка 1, Синтаксическая ошибка: Ошибка в теле цикла/условного выражения |
| main [condition: 1 = 2 #] | Ошибка 609: строка 1, Синтаксическая ошибка: Ошибка в условии цикла/условного выражения |
| main [condition: 1 = 2 #] | Ошибка 610: строка 1, Синтаксическая ошибка: Неверный условный оператор |
| main [new number x; x = x ! x;] | Ошибка 611: строка 1, Синтаксическая ошибка: Неверный арифметический оператор |
| main [new number x; write new;] | Ошибка 612: строка 1, Синтаксическая ошибка: Неверное выражение. Ожидаются только идентификаторы/литералы |
| main [new number x; x = 1 ++;] | Ошибка 613: строка 1, Синтаксическая ошибка: Ошибка в арифметическом выражении |
| main [newline; 4;] | Ошибка 614: строка 1, Синтаксическая ошибка: Недопустимая синтаксическая конструкция |
| main [new number a; condition: a < 3 # istrue [newline; 3;] #] | Ошибка 615: строка 1, Синтаксическая ошибка: Недопустимая синтаксическая конструкция в теле цикла/условного выражения |

## 8.4 Тестирование семантического анализатора

Семантический анализ в языке PAA-2018 содержит множество проверок по семантическим правилам, описанным в пункте 1.16. Итоги тестирования семантического анализатора на корректное обнаружение семантических ошибок приведены в таблице 8.4.

Таблица 8.4 - Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main [ a = 1 ] | Ошибка N300: Семантическая ошибка: Необъявленный идентификатор Строка: 1 |
| string function fi()[] | Ошибка N301: Семантическая ошибка: Отсутствует точка входа main Строка: 0 |
| main[]  main[] | Ошибка N302: Семантическая ошибка: Обнаружено несколько точек входа main Строка: 0 |
| main[a = 1;] | Ошибка N304: Семантическая ошибка: В объявлении отсутствует ключевое слово new Строка: 1 |
| main[new number t;  new string t;] | Ошибка N305: Семантическая ошибка: Попытка переопределения идентификатора Строка: 3 |
| procedure function fi()[]  main[fi(“a”,”b”,”c”,”d”] | Ошибка N307: Семантическая ошибка: Слишком много параметров в вызове Строка: 1 |
| string function fi(string x, string y, string z, string s)  main[] | Ошибка N306: Семантическая ошибка: Превышено максимальное количество параметров функции Строка: 1 |
| string function fi(string x)[return "a";]  main[fi("a", "b");] | Ошибка N308: Семантическая ошибка: Кол-во ожидаемых функцией и передаваемых параметров не совпадают Строка: 2 |
| string function fi(string x)[return "a";]  main[fi("a", "b");] | Ошибка N309: Семантическая ошибка: Несовпадение типов передаваемых параметров Строка: 2 |
| main[new string x="";] | Ошибка N310: Семантическая ошибка: Использование пустого строкового литерала недопустимо Строка: 1 |
| main[new string x=";] | Ошибка N311: Семантическая ошибка: Обнаружен символ '"'. Возможно, не закрыт строковый литерал Строка: 1 |
| main[new number x=99999999999999999;] | Ошибка N313: Семантическая ошибка: Недопустимый целочисленный литерал Строка: 1 |
| main[new number x; x = 5 + "abc";] | Ошибка N314: Семантическая ошибка: Типы данных в выражении не совпадают Строка: 1 |
| string function fi()[return 5;]  main[newline;] | Ошибка N315: Семантическая ошибка: Тип функции и возвращаемого значения не совпадают Строка: 1 |
| main[new string x; x = "abc" + "d";] | Ошибка N316: Семантическая ошибка: Недопустимое строковое выражение справа от знака '=' Строка: 1 |
| main  [condition: "string"& 6#  istrue[write "string";]] | Ошибка N317: Семантическая ошибка: Неверное условное выражение Строка: 1 |
| main[new number a =5;  a = a/0; write a;] | Ошибка N318: Семантическая ошибка: Деление на ноль Строка: 4 |

# Заключение

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования PAA-2018 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка PAA-2018;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
5. Осуществлена программная реализация синтаксического анализатора;
6. Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
7. Разработан транслятор кода на язык ассемблера;
8. Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка PAA-2018 включает:

1. 2 типа данных;
2. Поддержка операторов вывода и перевода строки;
3. Возможность вызова функций стандартной библиотеки;
4. Наличие 4 арифметических операторов для вычисления выражений;
5. Поддержка функций, процедур, операторов цикла и условия;
6. Структурированная и классифицированная система для обработки ошибок пользователя.

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении трансляторов, а также основные различия и преимущества тех или иных средств трансляции.

Приложение А

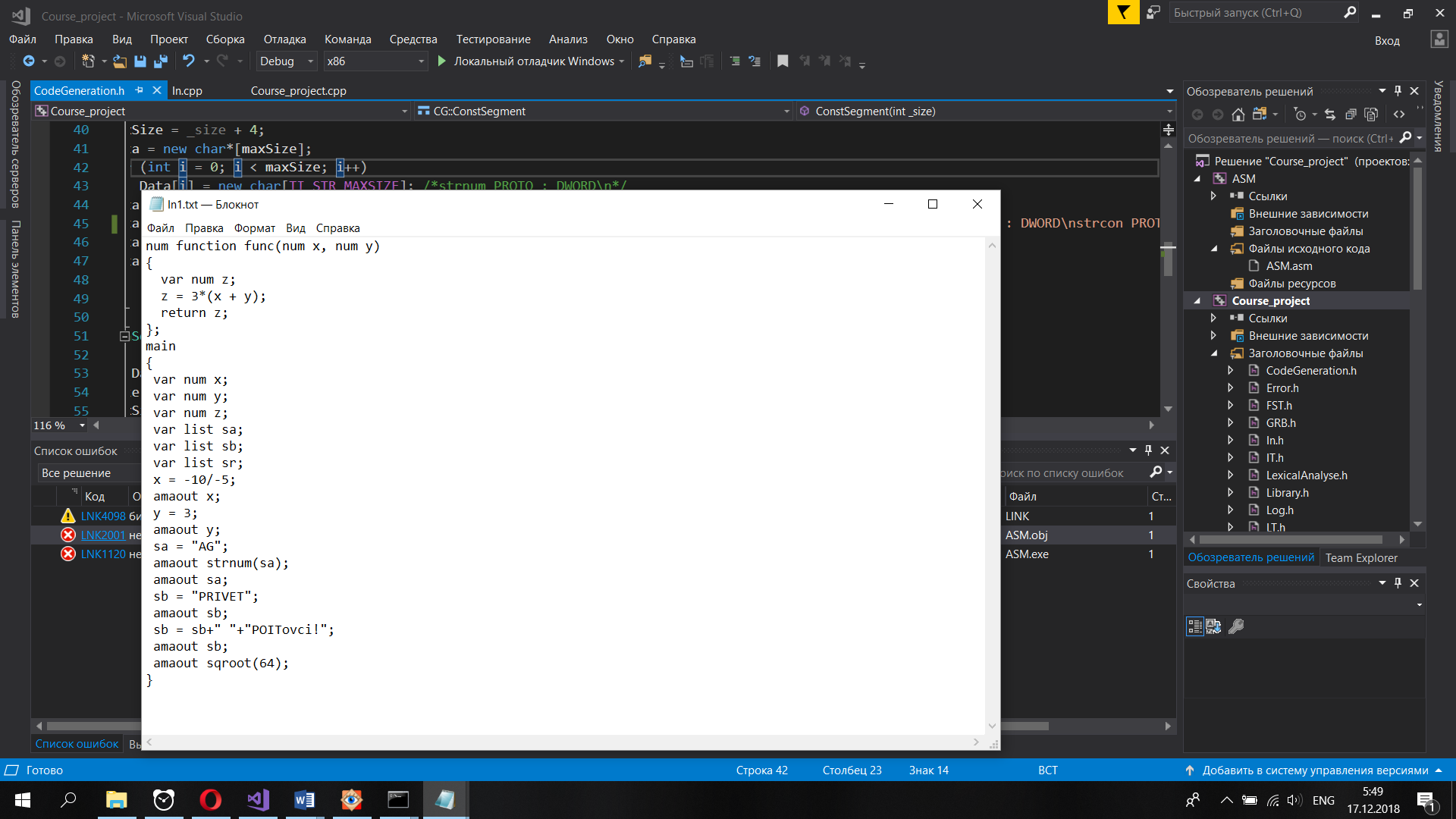


Рис. 8 - Исходный код

Приложение Б

enum IDDATATYPE {

NDEFD = 0, //неопределен

INT = 1, //ingeter

STR = 2 }; //string

enum IDTYPE

{

NDEFT = 0, //неопределен

V = 1, //переменная

F = 2, //функция

L = 3, //литерал

P = 4 //параметр

};

struct Entry // строка таблицы идентефикаторов

{

int idxfirstLE; // индекс первой строки в таблице лексем

char id[30] = "";// идентефикатор (автоматически усекается до ID\_MAXSIZE)

IDDATATYPE iddatatype; // тип данных

IDTYPE idtype;

char\* scope;

union

{

int vint;

struct

{

char len;

char str[TI\_STR\_MAXSIZE - 1];

} vstr[TI\_STR\_MAXSIZE];

}value;

void reset()

{

memset(id, NULL, ID\_MAXSIZE);

idxfirstLE = TI\_NULLIDX;

iddatatype = IT::NDEFD;

idtype = IT::NDEFT;

}

};

struct IdTable // экземпляр таблицы идентефикаторов

{

int maxsize;// емкость таблицы идентефикаторов < TI\_MAXSIZE

int size; // текущий размер таблицы идентефикаторов < maxsize

Entry\* table;// массив строк таблицы идентефикаторов

};

Рис. 9 - Структуры данных, используемые на фазе лексического анализа

Приложение В

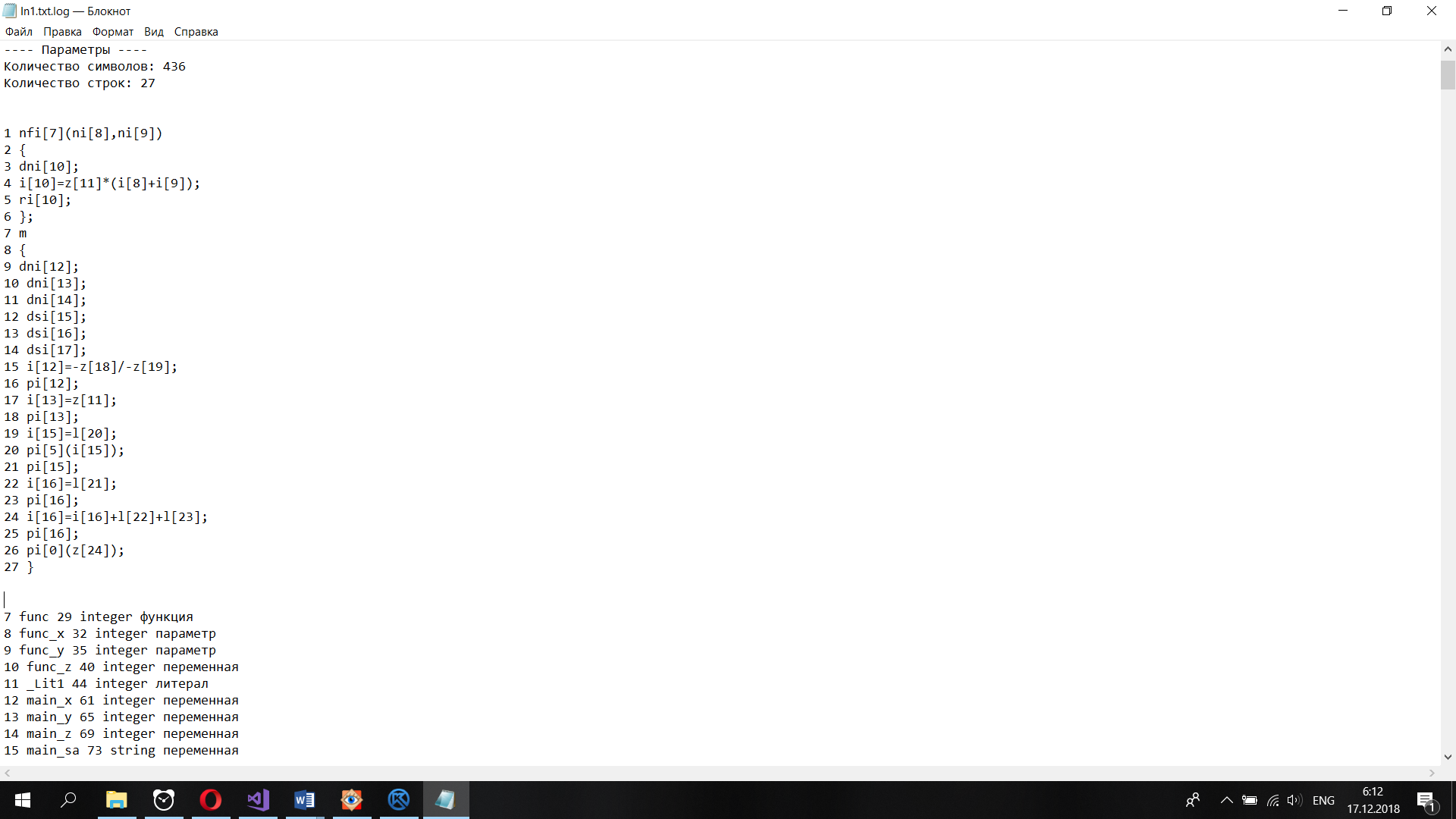


Рис. 10 - Таблица лексем

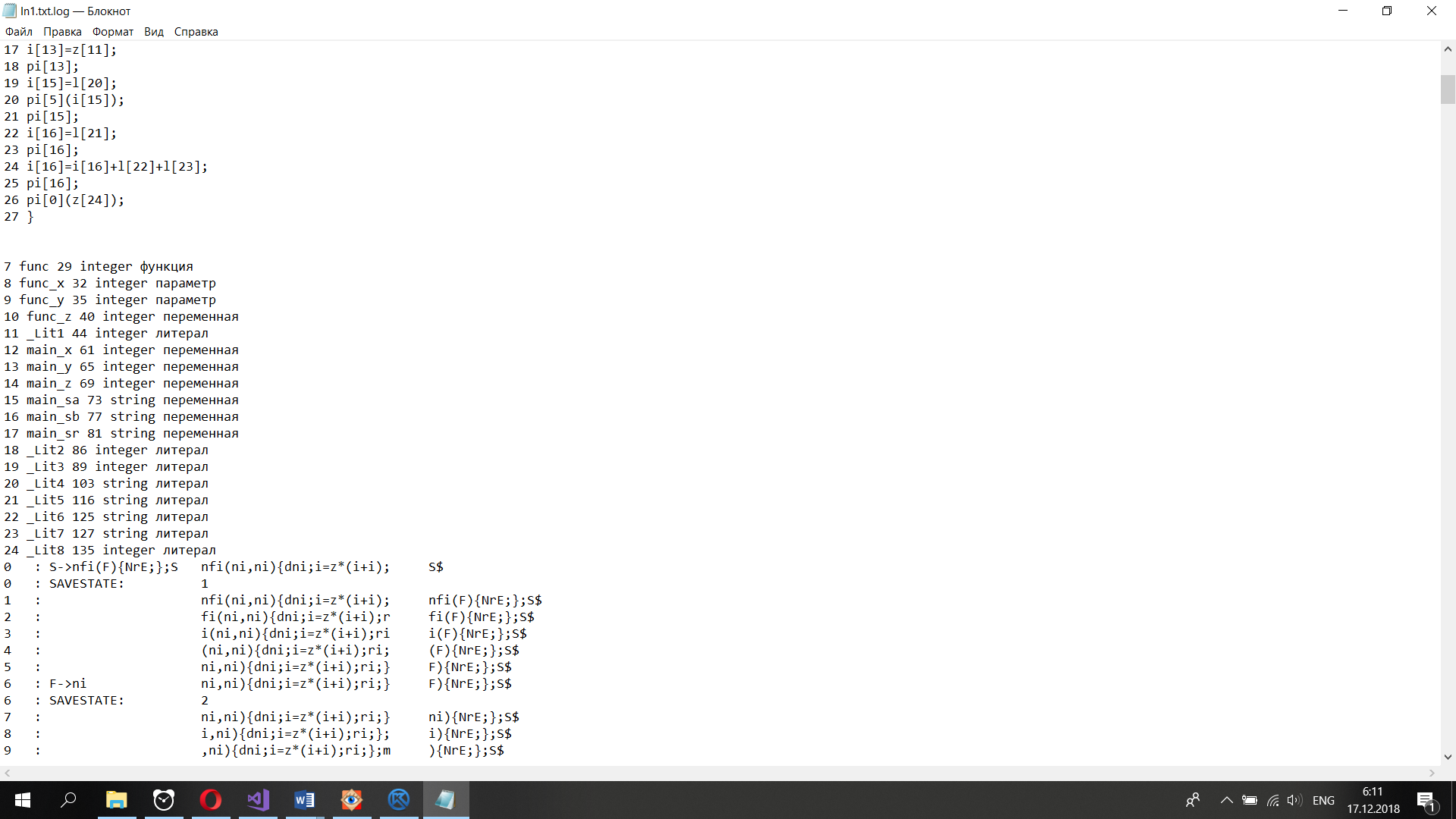


Рис. 11 - Таблица идентификаторов

Приложение Г

Greibach greibach(NS('S'), TS('$'), // стартовый символ, дно стека

6, // количество правил

Rule(NS('S'), GRB\_ERROR\_SERIES + 0, // Неверная структура программы

9, // S->m{N}; | pi(F){N}S

Rule::Chain(14, TS('n'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),

Rule::Chain(14, TS('s'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),

Rule::Chain(13, TS('n'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),

Rule::Chain(13, TS('s'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),

Rule::Chain(12, TS('n'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),

Rule::Chain(12, TS('s'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),

Rule::Chain(13, TS('n'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),

Rule::Chain(13, TS('s'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),

Rule::Chain(4, TS('m'), TS('{'), NS('N'), TS('}'))

),

Rule(NS('N'), GRB\_ERROR\_SERIES + 1, // Ошибочный оператор

25,//c(i>i)N c(i<i)N

Rule::Chain(5, TS('d'), TS('n'), TS('i'), TS(';'), NS('N')), //объявление integer

Rule::Chain(5, TS('d'), TS('s'), TS('i'), TS(';'), NS('N')), //объявление strin

Rule::Chain(5, TS('d'), TS('k'), TS('i'), TS(';'), NS('N')), //объявление bool

Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')), // инициализация идентификатора

Rule::Chain(7, TS('d'), TS('n'), TS('i'), TS('='), NS('E'), TS(';'), NS('N')), // объявл.+иниц. integer

Rule::Chain(7, TS('d'), TS('s'), TS('i'), TS('='), NS('E'), TS(';'), NS('N')), // объявл.+иниц. string

Rule::Chain(4, TS('p'), TS('i'), TS(';'), NS('N')), // вывод идентификатора

Rule::Chain(4, TS('p'), TS('l'), TS(';'), NS('N')), // вывод литерала

Rule::Chain(8, TS('i'), TS('='), TS('i'), TS('('), NS('W'), TS(')'), TS(';'), NS('N')), // присваение ид-ру рез-та вызова ф-ии

Rule::Chain(7, TS('p'), TS('i'), TS('('), NS('W'), TS(')'), TS(';'), NS('N')), // вывод результата ф-ии

Rule::Chain(6, TS('p'), TS('i'), TS('('), TS(')'), TS(';'), NS('N')),

Rule::Chain(9, TS('i'), TS('='), TS('i'), TS('('), NS('W'), TS(')'), NS('M'), TS(';'), NS('N')),

Rule::Chain(4, TS('p'), NS('E'), TS(';'), NS('N')),

Rule::Chain(4, TS('d'), TS('n'), TS('i'), TS(';')),

Rule::Chain(4, TS('d'), TS('s'), TS('i'), TS(';')),

Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),

Rule::Chain(6, TS('d'), TS('n'), TS('i'), TS('='), NS('E'), TS(';')),

Rule::Chain(6, TS('d'), TS('s'), TS('i'), TS('='), NS('E'), TS(';')),

Rule::Chain(3, TS('p'), TS('i'), TS(';')),

Rule::Chain(3, TS('p'), TS('l'), TS(';')),

Rule::Chain(7, TS('i'), TS('='), TS('i'), TS('('), NS('W'), TS(')'), TS(';')),

Rule::Chain(6, TS('p'), TS('i'), TS('('), NS('W'), TS(')'), TS(';')),

Rule::Chain(5, TS('p'), TS('i'), TS('('), TS(')'), TS(';')),

Rule::Chain(8, TS('i'), TS('='), TS('i'), TS('('), NS('W'), TS(')'), NS('M'), TS(';')),

Rule::Chain(3, TS('p'), NS('E'), TS(';'))

),

Rule(NS('E'), GRB\_ERROR\_SERIES + 2, // Ошибка в выражении

17, //E -> i | iM | (E) | (E)M | i(W) | i(W)M | nM | n

Rule::Chain(2, TS('i'), NS('M')),

Rule::Chain(2, TS('z'), NS('M')),

Rule::Chain(2, TS('l'), NS('M')),

Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),

Rule::Chain(3, TS('i'), TS('('), TS(')')),

Rule::Chain(4, TS('i'), TS('('), TS(')'), NS('M')),

Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M')),

Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),

Rule::Chain(5, TS('-'), TS('('), NS('E'), TS(')'), NS('M')),

Rule::Chain(3, TS('('), NS('E'), TS(')')),

Rule::Chain(1, TS('z')),

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('l')),

Rule::Chain(3, TS('-'), TS('i'), NS('M')),

Rule::Chain(3, TS('-'), TS('z'), NS('M')),

Rule::Chain(2, TS('-'), TS('z')),

Rule::Chain(2, TS('-'), TS('i'))

),

Rule(NS('M'), GRB\_ERROR\_SERIES + 3, // Ошибка в выражении

4, //M -> vE

Rule::Chain(2, TS('+'), NS('E')),

Rule::Chain(2, TS('-'), NS('E')),

Rule::Chain(2, TS('\*'), NS('E')),

Rule::Chain(2, TS('/'), NS('E'))

),

Rule(NS('F'), GRB\_ERROR\_SERIES + 4, // Ошибка в параметрах функции +

4, //F -> ti | ti,F

Rule::Chain(2, TS('n'), TS('i')),

Rule::Chain(4, TS('n'), TS('i'), TS(','), NS('F')),

Rule::Chain(2, TS('s'), TS('i')),

Rule::Chain(4, TS('s'), TS('i'), TS(','), NS('F'))

),

Rule(NS('W'), GRB\_ERROR\_SERIES + 5, // Ошибки в параметрах вызываемой +

6, //W -> i | l | i,W | l,W

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('z')),

Rule::Chain(1, TS('l')),

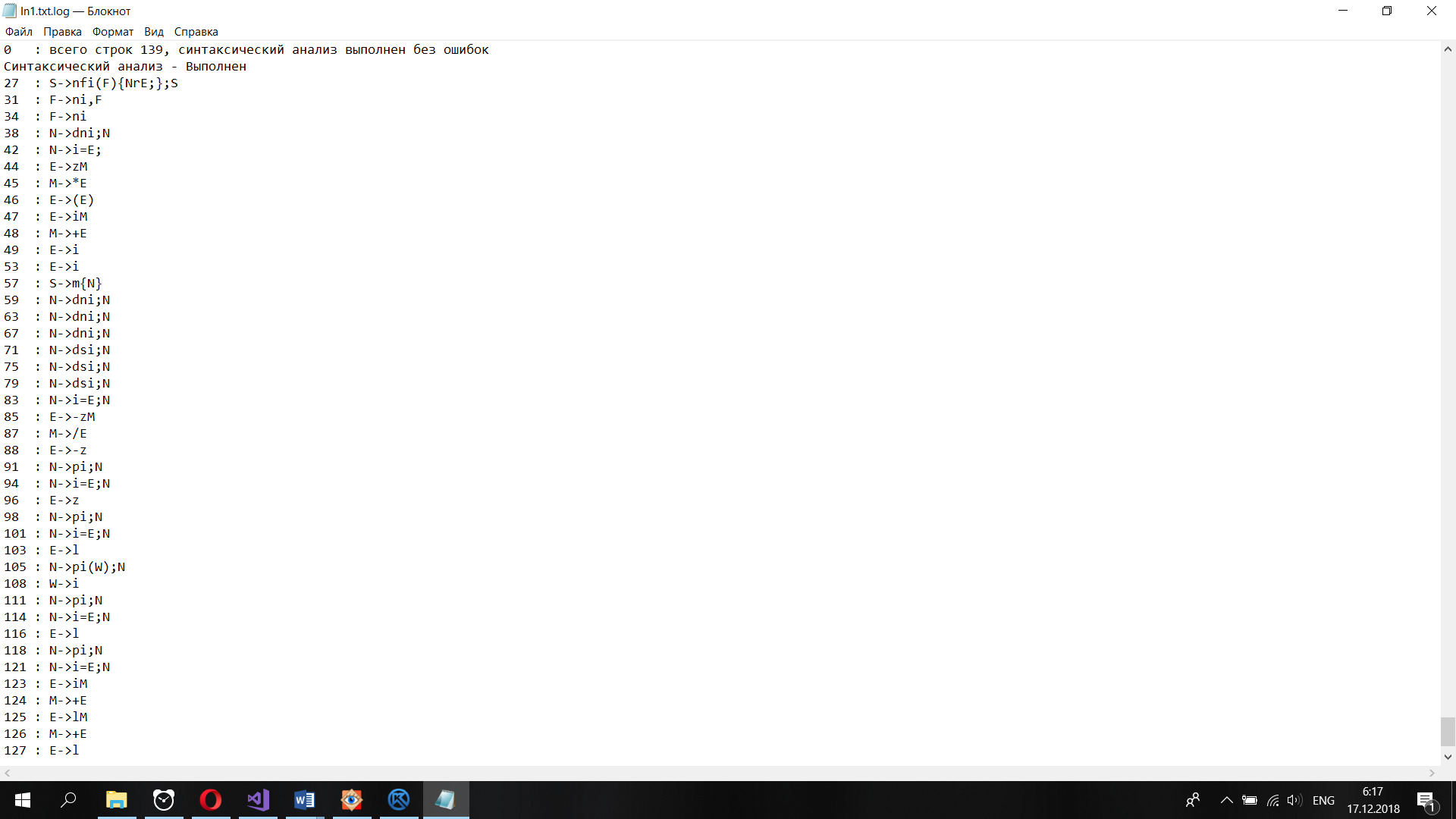
Rule::Chain(3, TS('i'), TS(','), NS('W')),

Rule::Chain(3, TS('l'), TS(','), NS('W')),

Rule::Chain(3, TS('z'), TS(','), NS('W'))));

Рис.11 - Структура данных грамматики Грейбах

Приложение Д



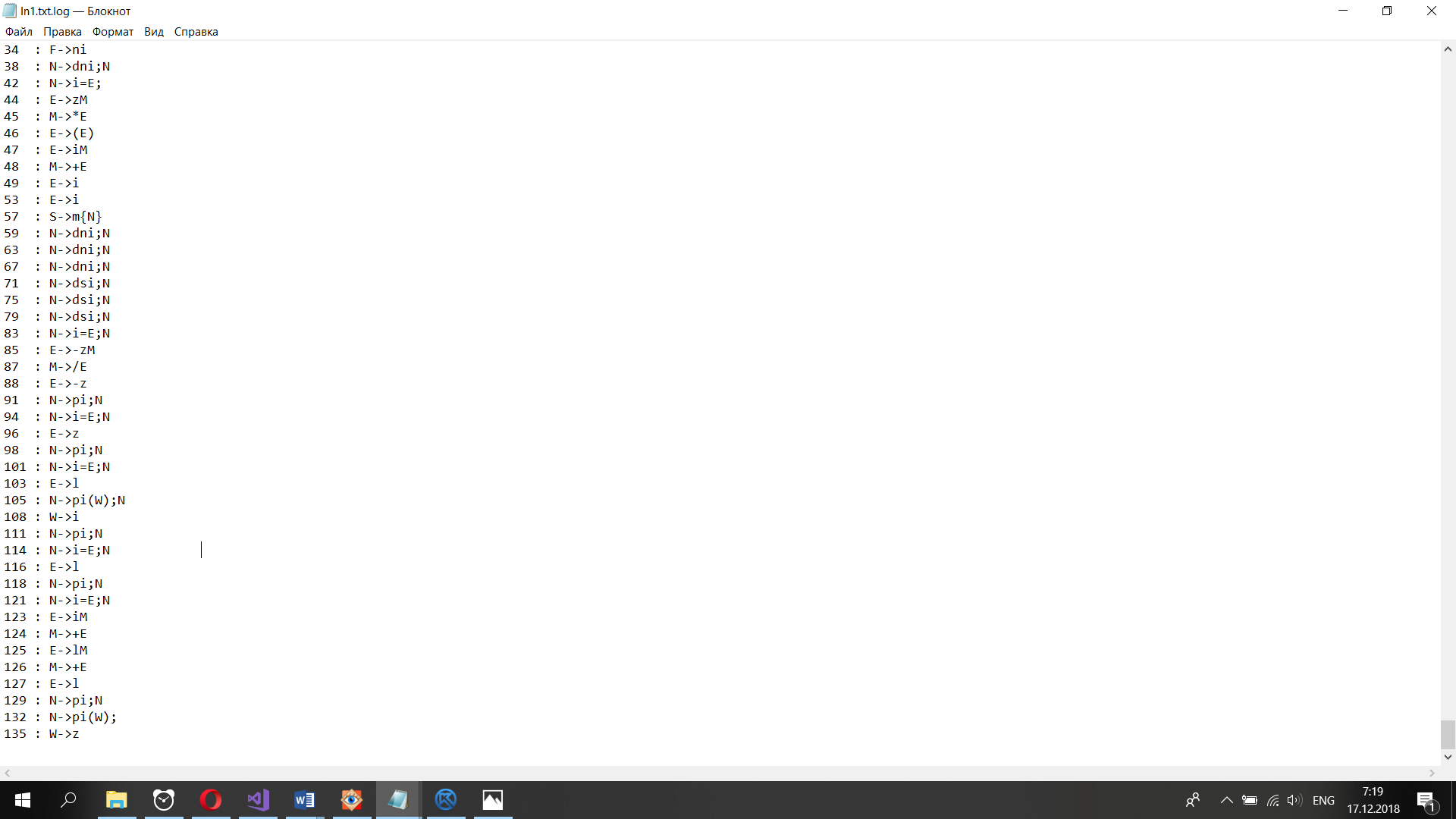


Рис. 12 - Дерево разбора

Приложение Е

Таблица 1 – Результат генерации кода

|  |
| --- |
| .586  .model flat, stdcall  includelib userlib.lib  includelib kernel32.lib  includelib libucrt.lib  ExitProcess PROTO : DWORD  strnum PROTO : DWORD  sqroot PROTO : DWORD  copystr PROTO : DWORD, : DWORD  strcon PROTO : DWORD, : DWORD  outstr PROTO : DWORD  outint PROTO : SDWORD  .stack 4096  .const  overflow db 'ERROR: VARIABLE OVERFLOW', 0  null\_division db 'ERROR: DIVISION BY ZERO', 0  \_Lit1 SDWORD 3  \_Lit2 SDWORD 10  \_Lit3 SDWORD 5  \_Lit4 BYTE "AG", 0  \_Lit5 BYTE "PRIVET", 0  \_Lit6 BYTE " ", 0  \_Lit7 BYTE "POITovci!", 0  \_Lit8 SDWORD 64  .data  func\_z SDWORD 0  main\_x SDWORD 0  main\_y SDWORD 0  main\_z SDWORD 0  main\_sa BYTE 255 DUP(0)  main\_sb BYTE 255 DUP(0)  main\_sr BYTE 255 DUP(0)  .code  func\_proc PROC, func\_x : SDWORD, func\_y : SDWORD  push \_Lit1  push func\_x  push func\_y  pop eax  pop ebx  add eax, ebx  jo EXIT\_OVERFLOW  push eax  pop eax  pop ebx  imul eax, ebx  jo EXIT\_OVERFLOW  push eax  pop func\_z  push func\_z  jmp EXIT  EXIT\_DIV\_ON\_NULL:  push offset null\_division  call outstr  push - 1  call ExitProcess  EXIT\_OVERFLOW:  push offset overflow  call outstr  push - 2  call ExitProcess  EXIT:  pop eax  ret 8  func\_proc ENDP  main PROC  push \_Lit2  pop eax  neg eax  push eax  push \_Lit3  pop eax  neg eax  push eax  pop ebx  pop eax  test ebx,ebx  jz EXIT\_DIV\_ON\_NULL  cdq  idiv ebx  push eax  pop main\_x  push main\_x  call outint  push \_Lit1  pop main\_y  push main\_y  call outint  push offset \_Lit4  push offset main\_sa  call copystr  push offset main\_sa  call strnum  push eax  call outint  push offset main\_sa  call outstr  push offset \_Lit5  push offset main\_sb  call copystr  push offset main\_sb  call outstr  push offset main\_sb  push offset \_Lit6  call strcon  jo EXIT\_OVERFLOW  push eax  push offset \_Lit7  call strcon  jo EXIT\_OVERFLOW  push eax  push offset main\_sb  call copystr  push offset main\_sb  call outstr  push \_Lit8  call sqroot  push eax  call outint  jmp EXIT  EXIT\_DIV\_ON\_NULL:  push offset null\_division  call outstr  push - 1  call ExitProcess  EXIT\_OVERFLOW:  push offset overflow  call outstr  push - 2  call ExitProcess  EXIT:  push 0  call ExitProcess  main ENDP  end main |

Литература

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

2. Смелов, В.В. Курс лекций по предмету языки программирования – 2016

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.