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 DVV-2018»

Выполнил студент Дубалеко Валентин Викторович

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

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

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

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

Консультанты ст.пр. Наркевич Аделина Сергеевна

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

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

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2018

Оглавление

[**Введение** 5](#_Toc532789813)

[**Глава 1 Спецификация языка программирования** 6](#_Toc532789814)

[**1.1 Характеристика языка программирования** 6](#_Toc532789815)

[**1.2 Алфавит языка** 6](#_Toc532789816)

[**1.3 Символы сепараторы** 6](#_Toc532789817)

[**1.4 Применяемы кодировки** 6](#_Toc532789818)

[**1.5 Типы данных** 7](#_Toc532789819)

[**1.7 Идентификаторы** 8](#_Toc532789820)

[**1.8Литералы** 8](#_Toc532789821)

[**1.9 Область видимости идентификаторов** 8](#_Toc532789822)

[**1.10 Инициализация данных** 8](#_Toc532789823)

[**1.11 Инструкции языка** 9](#_Toc532789824)

[**1.13 Выражения и их вычисления** 9](#_Toc532789825)

[**1.14 Программные конструкции языка** 10](#_Toc532789826)

[**1.15 Область видимости** 10](#_Toc532789827)

[**1.16 Семантические проверки** 10](#_Toc532789828)

[**1.17 Распределение оперативной памяти на этапе выполнения** 11](#_Toc532789829)

[**1.18 Стандартная библиотека и её состав** 11](#_Toc532789830)

[**1.19 Ввод и вывод данных** 11](#_Toc532789831)

[**1.20 Точка входа** 11](#_Toc532789832)

[**1.21 Препроцессор** 11](#_Toc532789833)

[**1.22 Соглашение о вызовах** 11](#_Toc532789834)

[**1.23 Объектный код** 11](#_Toc532789835)

[**1.24 Классификация сообщений транслятора** 12](#_Toc532789836)

[**1.25 Контрольный пример** 12](#_Toc532789837)

[**Глава 2 Структура транслятора** 12](#_Toc532789838)

[**2.1 Компоненты транслятора их назначение и принципы взаимодействия** 12](#_Toc532789839)

[**2.2 Перечень входных параметров транслятора** 13](#_Toc532789840)

[**2.3 Протоколы формируемые транслятором** 16](#_Toc532789841)

[**Глава 3 Разработка лексического анализатора** 18](#_Toc532789842)

[**3.1 Структура лексического анализатора** 18](#_Toc532789843)

[**3.2 Контроль входных символов** 18](#_Toc532789844)

[**3.3 Удаление избыточных символов** 19](#_Toc532789845)

[**3.4 Перечень ключевых слов** 19](#_Toc532789846)

[**3.5 Основные структуры данных** 19](#_Toc532789847)

[**3.6. Структура и перечень сообщений лексического анализатора** 20](#_Toc532789848)

[**3.7. Принцип обработки ошибок** 23](#_Toc532789849)

[**3.8 Параметры лексического анализатора** 23](#_Toc532789850)

[**3.9. Алгоритм лексического анализа** 23](#_Toc532789851)

[**3.10 Контрольный пример** 24](#_Toc532789852)

[**Глава 4 Разработка синтаксического анализатора** 26](#_Toc532789853)

[**4.1 Структура синтаксического анализатора** 26](#_Toc532789854)

[**4.2. Контекстно-свободная грамматика, описывающая синтаксис языка** 26](#_Toc532789855)

[**4.3 Построение конечного магазинного автомата** 26](#_Toc532789856)

[**4.4Основные структуры данных** 27](#_Toc532789857)

[**4.5 Описание алгоритма синтаксического разбора.** 28](#_Toc532789858)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 28](#_Toc532789859)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 30](#_Toc532789860)

[**4.8. Принцип обработки ошибок** 30](#_Toc532789861)

[**4.9. Контрольный пример** 30](#_Toc532789862)

[**Глава 5 Разработка семантического анализатора** 31](#_Toc532789863)

[**5.1 Структура семантического анализатора** 31](#_Toc532789864)

[**5.2 Функции семантического анализатора** 31](#_Toc532789865)

[**5.3 Структура и перечень сообщений семантического анализатора** 32](#_Toc532789866)

[**5.4 Принцип обработки ошибок** 32](#_Toc532789867)

[**5.5 Контрольный пример** 32](#_Toc532789868)

[**Глава 6 Вычисление выражений** 33](#_Toc532789869)

[**6.1 Выражения, допускаемы языком** 33](#_Toc532789870)

[**6.2 Польская запись и принцип ее построения** 33](#_Toc532789871)

[**6.3 Программная реализация обработки выражений** 33](#_Toc532789872)

[**6.4 Контрольный пример** 33](#_Toc532789873)

[**Глава 7 Генерация Кода** 34](#_Toc532789874)

[**7.1 Структура генератора кода** 34](#_Toc532789875)

[**7.2 Представление типов данных в оперативной памяти** 34](#_Toc532789876)

[**7.3 Статическая библиотека** 35](#_Toc532789877)

[**7.4 Особенности алгоритма генерации кода** 35](#_Toc532789878)

[**7.5 Входные параметры генератора кода** 35](#_Toc532789879)

[**7.6 Контрольный пример** 35](#_Toc532789880)

[**Глава 8 Тестирование транслятора** 36](#_Toc532789881)

[**8.1 Общие положения** 36](#_Toc532789882)

[**8.2 Результаты тестирования** 36](#_Toc532789883)

[**Заключение:** 40](#_Toc532789884)

[**Графический материал** 41](#_Toc532789885)

[**Список использованных источников:** 42](#_Toc532789886)

[**Приложение А** 43](#_Toc532789887)

[**Приложение Б** 48](#_Toc532789888)

[**Приложение В** 50](#_Toc532789889)

[**Приложение Г** 51](#_Toc532789890)

[**Приложение Д** 53](#_Toc532789891)

[**Приложение Е** 54](#_Toc532789892)

[**Приложение К** 55](#_Toc532789893)

# **Введение**

Основной целью данной курсовой работы является разработка транслятора для языка программирования DVV-2018. Главной задачей для транслятора будет сделать язык DVV-2018 понятным для вычислительной машины. В данном курсовом проекте будет происходить трансляция кода на языке DVV-2018 в код на языке ассемблер.

Для данного курсового проекта были определены следующие задачи:

* разработка спецификации языка программирования;
* разработка структуры транслятора;
* разработка лексического анализатора;
* разработка синтаксического анализатора;
* разработка семантического анализатора;
* обработка выражений;
* генерация кода в C#;
* тестирование транслятора.

Решение для каждой из поставленных задач будут отображены в следующих главах курсового проекта:

* спецификация языка программирования;
* структура транслятора;
* разработка лексического анализатора;
* разработка синтаксического анализатора;
* разработка семантического анализатора;
* вычисление выражений;
* генерация кода;
* тестирование транслятора.

Язык программирования DVV-2018 предназначается для выполнения простейших арифметический и логических действий , а также для операций над строками.

# **Глава 1 Спецификация языка программирования**

**1.1 Характеристика языка программирования**

Язык DVV-2018 – это процедурный, универсальный, строготипизированный, компилируемый язык. Так же язык не является объектно-ориентированным.

* универсальный — язык, система терминов, определенных строго и однозначно, а потому допускающих над собой чисто формальные операции.
* процедурный – язык при программировании на котором последовательно выполняемые операторы можно собрать в подпрограммы, то есть более крупные целостные единицы кода, с помощью механизмов самого языка.
* сильной(строгой) типизацией в языке является отсуствия приведения типов
* компилируемый – язык который конвертируется в машинный код.

**1.2 Алфавит языка**

Алфавит языка DVV-2018 основан на кодировке Windows-1251, представленной на рисунке 1.1. Исходный код DVV-2018 может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, символы русского алфавита можно использовать только в строковых литералах. Код, написанный на языке DVV-2018, может содержать символы : +, -, \*, /, {, }, (, ), ", >, <, =, : .

**1.3 Символы сепараторы**

Символы, которые являются сепараторами в языке DVV-2018 представлены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| (space)  (\n)  .  , | разделение инструкций  переход на новую строку  признак окончания строки  разделение параметров функций |
| {  } | программный блок |
| (  ) | параметры/приоритетность операций (в выражениях) |

**1.4 Применяемы кодировки**

Для написания исходного кода на языке программирования DVV-2018 используется кодировка Windows-1251. Кодировка Windows-1251 представлена на рисунке 1.1



Рисунок 1.1 – Алфавит входных символов

**1.5 Типы данных**

В языке DVV-2018 есть 3 типа данных: целочисленный integer, строковый string и логический bool. Описание типов данных, предусмотренных данным языком представлены в таблице 1.2.

Таблица 1.2 – Типы данных языка DVV-2018

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| integer | Фундаментальный тип данных. Предусмотрен для объявления целочисленных положительных данных.  Автоматически инициализируется нулевым значением.  Возможные операции:  + – суммирование;  - – вычитание;  \* – умножение;  / – деление;  : – присваивание значения; |
| String | Фундаментальный тип данных. Предусмотрен для объявления строк. Автоматическая инициализируется строкой нулевой длины. Максимальное количество символов в строке – 255. |
| Bool | Фундаментальный тип данных. Предусмотрен для использования в конструкции if() else. Может быть инициализирован 2 значениями true или false. Автоматическая инициализация не предусмотрена.  Возможные операции:  == , < , > – логические операторы сравнения. |

**1.6 Преобразование типов данных**

Так как язык DVV-2018 является строготипизированным , то по определению

сильной(строгой) типизации преобразование(приведение) типов в языке отсутствует.

**1.7 Идентификаторы**

В языке DVV-2018 для идентификатора можно использовать строчные буквы латинского алфавита. Максимальная длина идентификатора 20 символов.

**1.8Литералы**

Для языка DVV-2018 характерно присутствие литералов 3 типов: литералы целого типа, строковые литералы, а также литералы типа bool. Описание данных литералов можно прочитать ниже в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные неотрицательные литералы, инициализируются 0. Литералы только могут быть только rvalue. Диапазон изменения от 0 до 2147483647. |
| Строковые литералы | Символы, заключённые в " " (двойные кавычки), инициализируются пустой строкой, строковые переменные. Литералы могут быть только rvalue. |
| Логические литералы | Логические литералы могут быть инициализированы только двумя значениями truth или false. |

**1.9 Область видимости идентификаторов**

Область видимости «сверху вниз». В языке DVV-2018 требуется обязательное объявление переменной перед её использованием. Все переменные должны располагаться внутри функций. Возможность объявления переменных c одинаковым идентификаторами и глобальных переменных отсутствует.

**1.10 Инициализация данных**

Таблица 1.4 – Инициализация данных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| <тип данных><идентификатор>. | Автоматическая инициализация: переменные типа integer инициализируются нулём, переменные типа string – пустой строкой, переменные типа bool не имеют автоматической инициализации. |
| <идентификатор> : <значение>. | Присваивание переменной значения. |

**1.11 Инструкции языка**

Все возможные инструкции языка программирования DVV-2018 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования DVV-2018

|  |  |
| --- | --- |
| Инструкция | Запись на языке DVV-2018 |
| Объявление переменной | <тип данных> <идентификатор>. |
| Объявление функции | <тип данных> func <идентификатор> (<тип данных> <идентификатор>, …) […] |
| Присваивание | <идентификатор> : <значение>/<идентификатор>. |
| Возвращение значения | return <идентификатор> / <литерал>). |
| Вывод данных | print (<идентификатор> / <литерал>). |

**1.12 Операции языка**

Язык программирования DVV-2018 может выполнять арифметические и логические операции, представленные в таблице 1.5.

Таблица 1.6 – Приоритетности операций языка программирования DVV-2018

|  |  |
| --- | --- |
| Операция | Приоритетность операции |
| (  ) | 0 |
| , | 1 |
| \*  / | 3 |
| +  - | 2 |
| = , > , < | Данные операции не имеют приоритета |

Максимальным значением приоритетности является “3”, минимальным “0” соответственно.

**1.13 Выражения и их вычисления**

Выражения могут содержать вызов библиотечных функций. Допускаются выражения которые вычисляют значение из операндов. Вычисление в выражения происходит по приоритетности операций языка.

**1.14 Программные конструкции языка**

Ключевые программные конструкции языка программирования DVV-2018 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка DVV-2018

|  |  |
| --- | --- |
| Точка входа в приложение | dvv  [  …  ] |
| Функция | <тип> func <идентификатор> (<тип> <идентификатор>, …)  {return <выражение/идентификатор>.} |
| Блок if() else | if(проверяемое условие) […]  else […] |

**1.15 Область видимости**

В языке DVV-2018 переменные могут находятся внутри разных областей видимости. Для языка каждая функция является отдельной областью видимости. Объявление пользовательских областей видимости в языке не допустимо.

**1.16 Семантические проверки**

Таблица с перечнем семантических проверок, предусмотренных языком, приведена в таблице 1.8.

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 700 | Тип данных передаваемых в функцию должен соответствовать заявленному. |
| 701 | Количество параметров передаваемых в функцию должно соответствовать заявленному значению. |
| 702 | Тип данных справа в выражении должен совпадать с типом слева |
| 703 | Для сравнения в блоке if без знаков сравнения могут использоваться только литералы и идентификаторы типа bool |
| 704 | Деление в выражении на ноль. |
| 705 | В программе может быть только одна точка входа |
| 706 | Применение знаков арифметических операций к идентификаторам или литералам типа string или bool |
| 707 | Применение знаков логических операций к идентификаторам или литералам типа string или bool |
| 708 | Тип данных возвращаемого значения не соответствует типу данных функции |
| 709 | Повторное объявление переменной в коде. |
| 710 | Использование необъявленной переменной |

**1.17 Распределение оперативной памяти на этапе выполнения**

Размещение всех переменных происходит в куче. Распределение оперативной памяти происходит на этапе генерации. Таблица лексем и таблица идентификаторов сохраняются в структуры с выделенной динамической памятью, которая очищается по окончанию работы транслятора.

**1.18 Стандартная библиотека и её состав**

Функции стандартной библиотеки с описанием представлены в таблице 1.9. Стандартная библиотека написана на языке программирования C#.

Таблица 1.9 – Состав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| Strlen | Integer | str s – строка | Возвращает длину строки s |
| Substr | Bool | string x – строка,  string y – строка | Возвращает значение либо true либо false в зависимости от того является ли строка 2 подстрокой строки 1. |

**1.19 Ввод и вывод данных**

Ввод данных не предусмотрен в языке DVV-2018.А за вывод данных отвечает функция: print (<идентификатор или литерал>);

**1.20 Точка входа**

Точкой входа в языке DVV-2018 является – ключевое слово dvv.

**1.21 Препроцессор**

Препроцессор для языка DVV-2018 не предусмотрен.

**1.22 Соглашение о вызовах**

В языке вызов функций происходит по соглашению о вызовах stdcall. Выбор пал на это соглашение о вызовах так как это stdcall является одним из самых популярных соглашениях о вызовах.

Особенности stdcall:

* параметры передаются через стек;
* запись в стек идет справа налево
* очистку стека производит вызываемый код

**1.23 Объектный код**

Код на языке DVV-2018 транслируется в код на языке C#.

**1.24 Классификация сообщений транслятора**

При возникновении ошибки в коде программы на языке DVV-2018 и выявления её транслятором в файл протокола выводится сообщение. Классификация сообщений приведена в таблице 1.10.

Таблица 1.10 – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 100,104 | Ошибки параметров |
| 106-112 | Ошибки открытия и чтения файлов |
| 113-122 | Ошибки лексического анализа |
| 600-607 | Ошибки синтаксического анализа |
| 700-710 | Ошибки семантического анализа |

## **1.25 Контрольный пример**

Пример программы на языке DVV-2018 представлен на рисунке ниже:

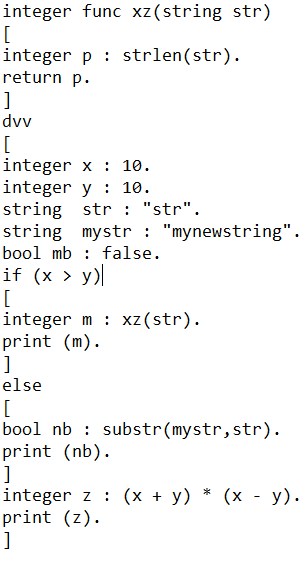


Рис. 1.2 – Контрольный пример

# **Глава 2 Структура транслятора**

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

Транслятор – обслуживающая программа, преобразующая исходную программу, предоставленную на входном языке программирования, в рабочую программу, представленную на объектном языке. Транслятор включает в себя такие компоненты как лексический анализатор, синтаксический анализатор, генератор объектного кода.

Лексический анализатор:

* выполняет аналитический разбор входной последовательности символов на распознанные группы — лексемы.
* входным параметром является исходный код.
* выходными параметрами являются таблица лексем и таблица идентификаторов.

Синтаксический анализатор:

* выполняет сопоставление линейной последовательности лексем естественного или формального языка с его формальной грамматикой.
* входным параметром является таблица лексем.
* выходным параметром является дерево разбора или иначе говоря синтаксическое дерево

Семантический анализатор:

* выполняет последовательности действий алгоритма автоматического понимания текстов, заключающегося в выделении семантических отношений, формировании семантического представления текстов.
* входными параметрами являются таблица идентификаторов и дерево разбора.
* выходными параметрами является семантическая структура

Генератор объектного кода:

* занимается построением для программы на входном языке эквивалентной машинной программы.
* входным параметром является промежуточное представление программы.
* выходным параметром является объектный код.

Подробное описание компонентов транслятора можно найти в главах курсового проекта под названием:

* разработка лексического анализатора
* разработка синтаксического анализатора
* разработка семантического анализатора
* генерация кода.

## **2.2 Перечень входных параметров транслятора**

Перечень входных параметров для языка DVV-2018 представлен в таблице 2.1

Таблица 2.1 – Входные параметры языка DVV - 2018

|  |  |
| --- | --- |
| Имя входного параметра | Выполняемая функция |
| -in | Указывает на файл из которого транслятор будет брать код |
| -log | Указывает на файл в который будет выводиться протокол работы. |
| -id | Указывает на файл в который будет выводиться таблицу идентификаторов |
| -lx | Указывает на файл в который будет выводиться таблица лексем |
| -rl | Указывает на файл в который будет выводиться дерево разбора или иначе говоря синтаксическое дерево |
| -cs | Указывает на файл в который будет выводится объектный код. |

## **2.3 Протоколы формируемые транслятором**

Для языка DVV-2018 предусмотрено формирование трех протоколов представленных в таблице 2.2

Таблица 2.2- Протоколы формируемые транслятором

|  |  |
| --- | --- |
| Имя входного протокола | Информация которая содержится в протоколе |
| <Имяфайла>.txt.log | Дата создания протокола. Перечень входных параметров языка DVV-2018. Исходные данные: Количество символов, Количество строк. |
| <Имяфайла>.txt.id | Таблица идентификаторов. Дата создания протокола. |
| <Имяфайла>.txt.lx | Таблица лексем. Дата создания протокола. |
| <Имяфайла>.txt.rl | Дерево разбора(Синтаксическое дерево) Дата создания протокола. |
| <Имяфайла>.txt.cs | Объектный код |

# **Глава 3 Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор представляет собой программу, которая транслирует исходную программу в набор символ, которая формирует таблицу лексем и таблицу идентификаторов, записывая в таблицы полученную информацию об идентификаторах и литералах.

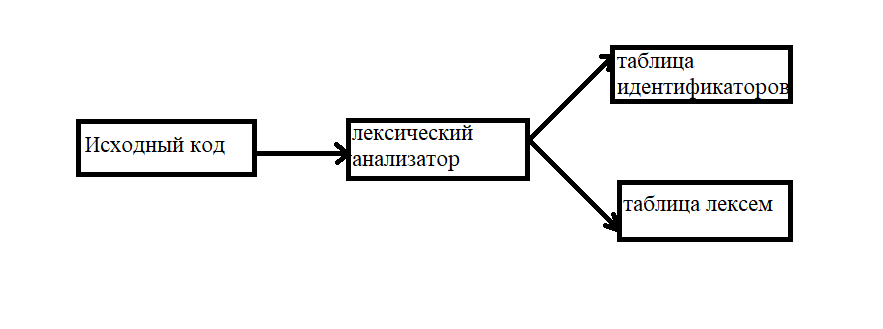


Рис. 3.1 Структура лексического анализатора

## **3.2 Контроль входных символов**

Таблица предназначенная для контроля входных символов представлена на рисунке 3.2.

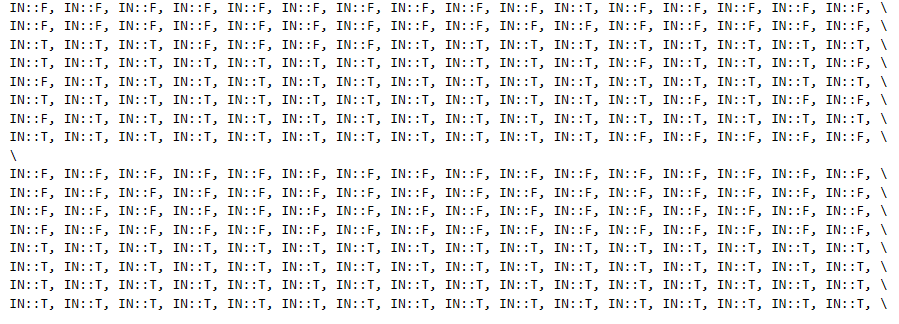


Рис 3.2 Входная таблица символов языка DVV-2018

Принцип работы таблицы состоит в следующем: каждому элементу в таблице соответствует значение в 16 системе счисления такой же, как и в кодовой таблице ASCII. Допустимые символы на рисунке обозначены буквой - T, запрещенные – F.

**3.3 Удаление избыточных символов**

Алгоритм удаления избыточных символов(пробел и переход на новую строку) происходит путем их игнорирования. Если при считывании из файла в буфер попадают избыточные символы ,то сначала проверяется не входят ли данные символы в строковый литерал, а иначе процесс сравнивания содержимого буфера с конечными автоматами пропускается.

## **3.4 Перечень ключевых слов**

Перечень ключевых слов, сепараторов, конечных автоматов языка DVV-2018 представлен в приложении А.

**3.5 Основные структуры данных**

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов.

Структуры таблицы лексем:

struct Entry//строка таблицы лексем

{

char lexema;

int sn;//номер строки в тексте

int indxTI;//индекс в таблице идентификаторов или LT\_TI\_NULLIDX

int parm; // количество параметров в функции

char buf[LT\_MAXSIZE \* 2]; // буффер для связи с таблицей лексем

int value; // значение бля связи с таблицей

int znak; // значение для обозначения всех операторов

};

struct LexTable//экземпяр таблицы лексем

{

int maxsize;//емкость таблицы лексем( < LT\_MAXSIZE)

int size;//текущий размер таблицы лексем ( < maxsize)

Entry\* table;//массив строк таблицы лексем

};

Структуры таблицы идентификаторов:

struct Entry

{

short idxfirstLE;//индекс первой строки в таблице лексем

char id[ID\_MAXSIZE \* 2];//идентификатор

IDDATATYPE iddatatype;//тип данных

IDTYPE idtype;//тип идентификатора

union

{

char vbool[6];//значение bool

int vint;//значение инт

struct

{

int len;//длина строки стр

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

}vstr[TI\_STR\_MAXSIZE];//значение стр

}value;//значение идентификатора

};

struct IdTable//экземпляр таблицы идентификаторов

{

int maxsize;//макс емкость таблицы идентификаторов( < TI\_MAXSIZE)

int size;//текущий размер таблицы идентификаторов ( < maxsize)

Entry\* table;//массив строк таблицы идентификаторов

};

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

Для лексического анализатора языка DVV-2018 в таблице ошибок зарезервированы ошибки с 113 по 122. Структура ошибок одинакова для всех частей транслятора при возникновении ошибки в консоль выводится:

* номер ошибки;
* сообщение об ошибке;
* строка где возникла ошибка;
* позиция в строке, где возникла ошибка.

В таблице 3.1 представлены номера и текст ошибок лексического анализатора.

Таблица 3.1 – Перечень ошибок лексического анализатора

|  |  |
| --- | --- |
| Номер ошибки | Выводимое сообщение |
| 113 | Превышен максимальный размер таблицы лексем |
| 114 | Превышен максимальный размер таблицы идентификаторов |
| 115 | Недопустимы номер возвращаемой строки таблицы лексем |
| 117 | Отсутствует таблица лексем |
| 118 | Отсутствует таблица идентификаторов |
| 119 | Недопустимый номер возвращаемой строки таблицы идентификаторов |
| 120 | Размер длины строки превышен(256) |
| 121 | Превышено максимальное значение целочисленного литерала |
| 122 | Превышен максимальный размер имени идентификатора |

## **3.7. Принцип обработки ошибок**

При обнаружении в коде ошибки лексический анализатор прекращает свою работу и выводит в консоль одну из ошибок представленных в таблице 3.1. В языке задан лимит на количество ошибок для лексического анализа их количество равно 8.

## **3.8 Параметры лексического анализатора**

Язык DVV-2018 допускает использование параметров для управления работой лексического анализатора. Принцип их использования описан в табл. 2.1.

**3.9. Алгоритм лексического анализа**

Алгоритм лексического анализа представлен ниже:

* из исходно файла подается текст пока в тексте не будет встречен один из символов сепараторов.
* при достижении символа сепаратора буфер усекается на нем.
* если это был сепаратор который не является пробелом или переходом на другую строку то он добавляется во второй буфер.
* дальше идет проверка текста записанного в первый буфер на то совпадает он с конечным автоматом.
* при совпадении происходит запись лексемы соответствующей совпавшему автомату в таблицу лексем
* если совпавший автомат был идентификатором или лексемой он также заносится в таблицу идентификаторов
* после этого производится проверка на совпадение записанного во второй буфер.
* при совпадении происходит запись лексемы соответствующей совпавшему автомату в таблицу лексем.

**3.10 Контрольный пример**

Результат работы Лексического Анализа на основе контрольного примера представленного в пункте 1.25 расположены ниже.

Таблица Лексем:

0. tfi(ti)

1. [

2. ti:s(i).

3. ri.

4. ]

5. d

6. [

7. ti:l.

8. ti:l.

9. ti:l.

10. ti:l.

11. ti:l.

12. w(i>i)

13. [

14. ti:i(i).

15. p(i).

16. ]

17. e

18. [

19. ti:s(i,i).

20. p(i).

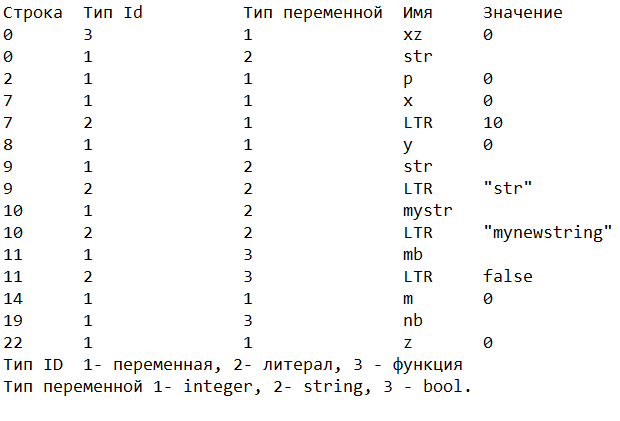
21. ]

22. ti:(i+i)\*(i-i).

23. p(i).

24. ]

Рис 3.2 Таблица идентификаторов



# **Глава 4 Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализатор представляет собой программу, проверяющую таблицу лексем по определенным правилам грамматики Грейбах. При удачной проверке таблицы лексем – формирует протокол работы, представляющий из дерево разбора или синтаксическое дерево.

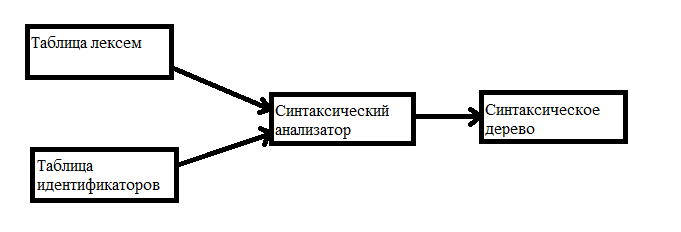


Рис. 4.1 Структура синтаксического анализатора

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

Формальная грамматика или просто грамматика в теории формальных языков — способ описания формального языка, то есть выделения некоторого подмножества из множества всех слов некоторого конечного алфавита. Различают порождающие и распознающие (или аналитические) грамматики — первые задают правила, с помощью которых можно построить любое слово языка, а вторые позволяют по данному слову определить, входит ли оно в язык или нет.

Стартовым символом для грамматики используемой в языке DVV-2018 является нетерминал S. А правила перехода нетерминальных автоматов указаны в приложении Б.

## **4.3 Построение конечного магазинного автомата**

В теории автоматов, автомат с магазинной памятью — это конечный автомат, который использует стек для хранения состояний. Память работает как стек, то есть для чтения доступен последний записанный в неё элемент. Таким образом, функция перехода является отображением. То есть, по комбинации текущего состояния, входного символа и символа на вершине магазина автомат выбирает следующее состояние и, возможно, символ для записи в магазин. В случае, когда в правой части автоматного правила присутствует, в магазин ничего не добавляется, а элемент с вершины стирается. Если магазин пуст, то срабатывают правила в левой части.

Формальное описание конечного магазинного автомата представлено на Рис 4.2.

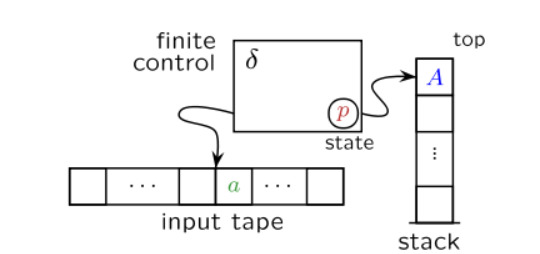


Рис 4.2. Формальное описание конечного магазинного автомата

**4.4Основные структуры данных**

Основные структуры данных для синтаксического анализа в языке DVV-2018 будут представлены ниже.

Структура Грейбах:

struct Greibach

{

short size;

GRBALPHABET startN;

GRBALPHABET stbottomT;

Rule\* rules;

Greibach() { short size = 0; startN = 0; stbottomT = 0; rules = 0; }

Greibach(GRBALPHABET pstartN, GRBALPHABET pstbottomT, short psize, Rule r, ...);

short getRule(GRBALPHABET pnn, Rule& prule);

Rule getRule(short n);

};

Структура правила:

struct Rule

{

GRBALPHABET nn;

int iderror;

short size;

struct Chain

{

short size;

GRBALPHABET\* nt;

Chain() { size = 0; nt = 0; }

Chain(short psize, GRBALPHABET s, ...);

char\* getCChain(char\* b);

static GRBALPHABET T(char t) { return GRBALPHABET(t); }

static GRBALPHABET N(char n) { return -GRBALPHABET(n); }

static bool isT(GRBALPHABET s) { return s > 0; }

static bool isN(GRBALPHABET s) { return !isT(s); }

static char alphabet\_to\_char(GRBALPHABET s) { return isT(s) ? char(s) : char(-s);}

}\*chains;

Rule() { nn = 0x00; size = 0; }

Rule(GRBALPHABET pnn, int iderror, short psize, Chain c, ...);

char\* getCRule(char\* b, short nchain);

short getNextChain(GRBALPHABET t, Rule::Chain& pchain, short j);

};

## **4.5 Описание алгоритма синтаксического разбора.**

Принцип работы синтаксического разбора для языка DVV-2018 приведен ниже:

* 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**

Для синтаксического анализатора языка DVV-2018 в таблице ошибок зарезервированы ошибки с 600 по 607. Структура ошибок одинакова для всех частей транслятора при возникновении ошибки в консоль выводится:

* Номер ошибки
* Сообщение об ошибке
* Строка где возникла ошибка

В таблице 4.2 представлены номера и текст ошибок синтаксического анализатора.

Таблица 4.2 – Ошибки синтаксического анализатора.

|  |  |
| --- | --- |
| Номер ошибки | Выводимое сообщение |
| 600 | Неверная структура программы |
| 601 | Ошибка в написании инструкций или конструкций языка |
| 602 | Ошибка инициализации |
| 603 | Ошибка выражения |
| 604 | Ошибка знака в выражении |
| 605 | Ошибка знака в сравнении |
| 606 | Ошибка в условии if |
| 607 | Ошибка объявления |

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

Язык DVV-2018 допускает использование параметров для управления работой синтаксического анализатора. Принцип их использования описан в табл. 2.1.

## **4.8. Принцип обработки ошибок**

При обнаружении в коде ошибки синтаксический анализатор прекращает свою работу и выводит в консоль одну из ошибок представленных в таблице 4.2. В языке

DVV-2018 для синтаксического анализа допустимо использование только 8 ошибок.

## **4.9. Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке DVV-2017 представлен в приложении В. Дерево разбора исходного кода представлено в графическом материале.

# **Глава 5 Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

В языке DVV-2018 семантический анализ выделен в отельную фазу работы транслятора, но некоторые семантические проверки проходят одновременно с работой лексического анализатора : проверка деления на ноль, проверка на единственность точки входа в приложение, проверка на переобъявление переменных. Входными параметрами для анализатора являются таблица лексем и таблица идентификаторов. Эта фаза транслятора не создает протоколов, а может только генерировать ошибки при нарушении семантических проверок.

## **5.2 Функции семантического анализатора**

Семантические проверки языка DVV-2018 представлены в таблице 5.1.

Таблица 5.1 – Семантические проверки

|  |  |  |
| --- | --- | --- |
| Семантическая проверка | Фаза выполнения | Краткое описание |
| Тип данных функции | Выполняется отдельно | Проверка на тип данных передаваемых в функцию при ее вызове |
| Количество параметров | Выполняется отдельно | Проверка на количество параметров передаваемых в функцию при ее вызове |
| Типы данных в выражении | Выполняется отдельно | Проверка на то, что ты одному типу данных не мог быть присвоен другой тип |
| Проверка условия блока if | Выполняется отдельно | Проверка на использование в if без логических знаков только данных типа bool |
| Арифметические и логические операции | Выполняется отдельно | Проверка на то, что бы в арифметических операциях нельзя было использовать данные типа string и bool |
| Тип возвращаемого значения | Выполняется отдельно | Проверка на тип возвращаемого функцией значения |
| Повторное объявление | Совместно с лексическим анализом | Проверка на поиск повторно объявленных переменных в коде |
| Необъявленная переменная | Выполняется отдельно | Проверка на поиск переменных которые используется в выражениях, но не объявлены |
| Деление на ноль | Совместно с лексическим анализом | Проверка в выраженияx на деление на ноль |
| Точка входа | Совместно с лексическим анализом | Проверка кода на единственность точки входа в приложение |

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

Для семантического анализатора языка DVV-2018 в таблице ошибок зарезервированы ошибки с 700 по 710. Данные ошибки представлены в пункте 1.16 пояснительной записки. Структура ошибок одинакова для всех частей транслятора при возникновении ошибки в консоль выводится:

* Номер ошибки
* Сообщение об ошибке
* Строка где возникла ошибка

## **5.4 Принцип обработки ошибок**

При обнаружении в коде ошибки семантический анализатор прекращает свою работу и выводит в консоль одну из ошибок представленных в таблице 1.8. В языке

DVV-2018 для семантического анализа допустимо использование только 10 ошибок.

## **5.5 Контрольный пример**

Распечатка с выданными ошибками семантического анализатора при корректировке контрольного примера приведена в приложении Г.

# **Глава 6 Вычисление выражений**

## **6.1 Выражения, допускаемы языком**

В языке DVV-2018 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \* , / , (), и вызовы функций как операнды арифметических выражений. Приоритет операций в языке DVV – 2018 представлена в таблице 1.6.

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

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок.

Обратная польская запись – это форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. 

Алгоритм построения:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку в порядке их следования;
* операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* 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 алгоритма преобразования выражений к обратной польской записи представлена в приложении Д.

## **6.4 Контрольный пример**

Контрольный пример представлен в Приложении E.

# **Глава 7 Генерация Кода**

## **7.1 Структура генератора кода**

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

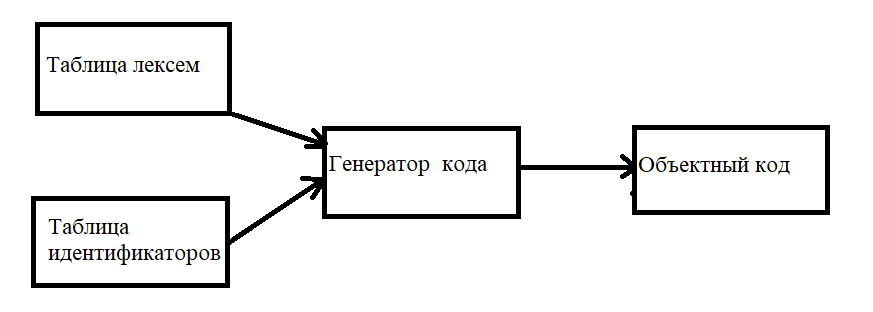


Рис 7.1 – Структура генератора кода

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

Описание памяти используемой языком DVV-2018 происходит в пункте 1.17 курсового проекта. Соответствия между типами данных идентификаторов на языке DVV-2018 и на языке С# приведены в таблице 7.1.

Таблица 7.1 – Соответствие типов данных языка DVV-2018 и C#

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке DVV-2018 | Тип идентификатора на языке c# | Описание типа |
| integer | Int | Хранит целочисленный тип данных со знаком. |
| string | String | Хранит указатель на начало строки. |
| bool | Bool | Хранит значение true либо false |

## **7.3 Статическая библиотека**

Описание содержимого статической библиотеки языка DVV-2018 находится в 1.9. Подключение статической библиотеки к полученному файлу на языке C# происходит на этапе компоновки.

## **7.4 Особенности алгоритма генерации кода**

Алгоритм генерации кода с языка DVV-2018 на язык C# происходит следующим способом:

* Запись в протокол “<Имяфайла>.txt.cs” исходного кода для языка C#
* Получение генератором кода таблицы таблицы лексем, таблицы идентификаторов, ссыли на файл куда производить запись
* Проверка лексемы из таблицы лексем на совпадение с одной из лексем языка DVV-2018
* При совпадении производить запись в файл, а иначе происходит выбор другой лексемы и возвращение к пункту 3.
* Выполняем пункты 3 и 4 пока таблица лексем не опустеет
* При получении готового файла “<Имяфайла>.txt.cs” происходит его компоновка со стандартной библиотекой и компиляция в exe файл

## **7.5 Входные параметры генератора кода**

Язык DVV-2018 допускает использование параметров для управления работой генератора кода. Принцип их использования описан в табл. 2.1.

## **7.6 Контрольный пример**

Сгенерированный контрольный пример описанный в пункте 1.25 можно просмотреть в приложении Ж.

# **Глава 8 Тестирование транслятора**

## **8.1 Общие положения**

Главной целью тестирования транслятора языка DVV-2018 на всех этапах его выполнения - это проверка текста программы на языке DVV-2018 на правильность написания. Если при проведении тестирования транслятора на любой его фазе будет выполнен один и тоже алгоритм действий, то выполнение программы прерывается и выдается код одной из ошибок из таблицы 1.10 , соответствующая этому коду ошибки строка, а так же строку и символ, где произошла ошибка. При удачном тестировании каждого этапа транслятора языка DVV-2018 в протоколы будут выводиться результаты тестирования. Со списком протоколов языка DVV-2018 можно ознакомиться в таблице 2.2.

## **8.2 Результаты тестирования**

Для языка DVV-2018 существует несколько фаз тестирования кода:

* Тестирование проверки на допустимость символов
* Тестирование лексического анализатора
* Тестирование синтаксического анализатора
* Тестирование семантического анализатора

По одной ошибке от каждой фазы тестирования кода представлено в таблице 8.1

Таблица 8.1 – Ошибки фаз тестирования

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| integer z : (x \ y) . | Ошибка 111: Недопустимый символ в исходном файле (-in) Cтрока 22, символ 21 |
| integer xqertyuiopkhfngmgg : 10. | Ошибка 122: Превышен максимальный размер имени идентификатора Строка: 7, символ: 34 |
| print m. | Ошибка 601: Ошибка в написании инструкций или конструкций языка Строка: 15, символ: 6 |
| string x : 10. | Ошибка 702: Тип данных справа в выражении должен совпадать с типом слева  Строка: 7, символ: 14 |

# **Заключение:**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования DVV - 2018. Таким образом, были выполнены основные задачи данной курсового проекта:

* разработана спецификации языка программирования;
* разработана структура транслятора
* разработан лексический анализатор;
* разработан синтаксический анализатор;
* разработан семантический анализатор;
* выполнена обработка выражений;
* выполнена генерация кода в C#;
* проведено тестирование транслятора.

Итогом выполнения курсового проекта стал язык программирования DVV-2018 содержащий в себе:

* 3 типа данных
* Поддержку операций вывода
* Блок if() else
* 4 арифметических и 3 логических оператора
* Возможность создавать пользовательские функции
* Библиотеку с 2 стандартными функциями для языка DVV-2018
* Структурированную систему для обработки ошибок пользователя

# **Графический материал**

# **Список использованных источников:**

* Вирт Н. Построение коммпиляторов Москва: ДМК Пресс, 2010. -192с.
* Хантер Р. Основные концепции компиляторов. Издательский дом “Вильямс” 2002 г . 256с.
* Axo A.B., Лам М.С., Сети Р., Ульман Дж.Д. Компиляторы: принципы, технологии и инструментарий 2-е издание. Издательский дом “Вильямс” 2008. – 1184с.
* Axo A.B., Сети Р. Ульман Дж.Д. Компиляторы: Принципы, технологии и нструменты. Издательский дом “Вильямс” 2003. -768с.
* Свердлов С.З. Языки программирования и методы трансляции. СПБ.:Питер, 2007. -638с.

**Приложение А**

**Ключевые слова и соответствующие им конечные автоматы**

|  |  |
| --- | --- |
| Ключевое слово | Конечный автомат |
| Integer | #define FST\_INTEGER 8, \  FST::NODE(1, FST::RELATION('i', 1)),\  FST::NODE(1, FST::RELATION('n', 2)),\  FST::NODE(1, FST::RELATION('t', 3)),\  FST::NODE(1, FST::RELATION('e', 4)),\  FST::NODE(1, FST::RELATION('g', 5)),\  FST::NODE(1, FST::RELATION('e', 6)),\  FST::NODE(1, FST::RELATION('r', 7)),\  FST::NODE() |
| String | #define FST\_STRING 7, \  FST::NODE(1, FST::RELATION('s', 1)),\  FST::NODE(1, FST::RELATION('t', 2)),\  FST::NODE(1, FST::RELATION('r', 3)),\  FST::NODE(1, FST::RELATION('i', 4)),\  FST::NODE(1, FST::RELATION('n', 5)),\  FST::NODE(1, FST::RELATION('g', 6)),\  FST::NODE() |
| Bool | #define FST\_BOOL 5, \  FST::NODE(1, FST::RELATION('b', 1)),\  FST::NODE(1, FST::RELATION('o', 2)),\  FST::NODE(1, FST::RELATION('o', 3)),\  FST::NODE(1, FST::RELATION('l', 4)),\  FST::NODE() |
| Return | #define FST\_RETURN 7, \  FST::NODE(1, FST::RELATION('r', 1)),\  FST::NODE(1, FST::RELATION('e', 2)),\  FST::NODE(1, FST::RELATION('t', 3)),\  FST::NODE(1, FST::RELATION('u', 4)),\  FST::NODE(1, FST::RELATION('r', 5)),\  FST::NODE(1, FST::RELATION('n', 6)),\  FST::NODE() |
| Print | #define FST\_PRINT 6, \  FST::NODE(1, FST::RELATION('p', 1)),\  FST::NODE(1, FST::RELATION('r', 2)),\  FST::NODE(1, FST::RELATION('i', 3)),\  FST::NODE(1, FST::RELATION('n', 4)),\  FST::NODE(1, FST::RELATION('t', 5)),\  FST::NODE() |
| func | #define FST\_FUNC 5, \  FST::NODE(1, FST::RELATION('f', 1)), \  FST::NODE(1, FST::RELATION('u', 2)), \  FST::NODE(1, FST::RELATION('n', 3)), \  FST::NODE(1, FST::RELATION('c', 4)), \  FST::NODE() |
| if | #define FST\_IF 3, \  FST::NODE(1, FST::RELATION('i',1)),\  FST::NODE(1, FST::RELATION('f',2)),\  FST::NODE() |
| else | #define FST\_ELSE 5, \  FST::NODE(1, FST::RELATION('e', 1)),\  FST::NODE(1, FST::RELATION('l', 2)),\  FST::NODE(1, FST::RELATION('s', 3)),\  FST::NODE(1, FST::RELATION('e', 4)),\  FST::NODE() |
| strlen | #define FST\_STRLEN 7,\  FST::NODE(1, FST::RELATION('s',1)),\  FST::NODE(1, FST::RELATION('t',2)),\  FST::NODE(1, FST::RELATION('r',3)),\  FST::NODE(1, FST::RELATION('l',4)),\  FST::NODE(1, FST::RELATION('e',5)),\  FST::NODE(1, FST::RELATION('n',6)),\  FST::NODE() |
| substr | #define FST\_SUBSTR 7,\  FST::NODE(1, FST::RELATION('s',1)),\  FST::NODE(1, FST::RELATION('u',2)),\  FST::NODE(1, FST::RELATION('b',3)),\  FST::NODE(1, FST::RELATION('s',4)),\  FST::NODE(1, FST::RELATION('t',5)),\  FST::NODE(1, FST::RELATION('r',6)),\  FST::NODE() |
| <id> | #define FST\_ID 2, \  FST::NODE(52, \  FST::RELATION('a', 0), FST::RELATION('b', 0), FST::RELATION('c', 0), FST::RELATION('d', 0), FST::RELATION('e', 0), FST::RELATION('f', 0),\  FST::RELATION('g', 0), FST::RELATION('h', 0), FST::RELATION('i', 0), FST::RELATION('j', 0), FST::RELATION('k', 0), FST::RELATION('l', 0),\  FST::RELATION('m', 0), FST::RELATION('n', 0), FST::RELATION('o', 0), FST::RELATION('p', 0), FST::RELATION('q', 0), FST::RELATION('r', 0),\  FST::RELATION('s', 0), FST::RELATION('t', 0), FST::RELATION('u', 0), FST::RELATION('v', 0), FST::RELATION('w', 0), FST::RELATION('x', 0),\  FST::RELATION('y', 0), FST::RELATION('z', 0),\  \  FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1),\  FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1),\  FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1),\  FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1),\  FST::RELATION('y', 1), FST::RELATION('z', 1)),\  FST::NODE() |
| <intliteral> | #define FST\_INTLIT 2, \  FST::NODE(20, \  FST::RELATION('1', 0), FST::RELATION('2', 0), FST::RELATION('3', 0), FST::RELATION('4', 0), FST::RELATION('5', 0), FST::RELATION('6', 0),\  FST::RELATION('7', 0), FST::RELATION('8', 0), FST::RELATION('9', 0), FST::RELATION('0', 0),\  \  FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1), FST::RELATION('6', 1),\  FST::RELATION('7', 1), FST::RELATION('8', 1), FST::RELATION('9', 1), FST::RELATION('0', 1)),\  FST::NODE() |
| <stringliteral> | #define FST\_STRLIT 4, \  FST::NODE(2, FST::RELATION('\'', 1), FST::RELATION('\'', 2)),\  FST::NODE(158, \  FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1),\  FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1),\  FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1),\  FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1),\  FST::RELATION('y', 1), FST::RELATION('z', 1), FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), FST::RELATION('4', 1),\  FST::RELATION('5', 1), FST::RELATION('6', 1), FST::RELATION('7', 1), FST::RELATION('8', 1), FST::RELATION('9', 1), FST::RELATION('0', 1),\  \  FST::RELATION('а', 1), FST::RELATION('б', 1), FST::RELATION('в', 1), FST::RELATION('г', 1), FST::RELATION('д', 1), FST::RELATION('е', 1),\  FST::RELATION('ё', 1), FST::RELATION('ж', 1), FST::RELATION('з', 1), FST::RELATION('и', 1), FST::RELATION('й', 1), FST::RELATION('к', 1),\  FST::RELATION('л', 1), FST::RELATION('м', 1), FST::RELATION('н', 1), FST::RELATION('о', 1), FST::RELATION('п', 1), FST::RELATION('р', 1),\  FST::RELATION('с', 1), FST::RELATION('т', 1), FST::RELATION('у', 1), FST::RELATION('ф', 1), FST::RELATION('х', 1), FST::RELATION('ц', 1),\  FST::RELATION('ч', 1), FST::RELATION('ш', 1), FST::RELATION('щ', 1), FST::RELATION('ъ', 1), FST::RELATION('ы', 1), FST::RELATION('ь', 1),\  FST::RELATION('э', 1), FST::RELATION('ю', 1), FST::RELATION('я', 1), FST::RELATION(' ', 1), FST::RELATION('.', 1), FST::RELATION(',', 1),\  FST::RELATION('?', 1), FST::RELATION('!', 1), FST::RELATION(';', 1), FST::RELATION(':', 1), FST::RELATION('-', 1), FST::RELATION(')', 1),\  FST::RELATION('(', 1), \  \  FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2), FST::RELATION('d', 2), FST::RELATION('e', 2), FST::RELATION('f', 2),\  FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('i', 2), FST::RELATION('j', 2), FST::RELATION('k', 2), FST::RELATION('l', 2),\  FST::RELATION('m', 2), FST::RELATION('n', 2), FST::RELATION('o', 2), FST::RELATION('p', 2), FST::RELATION('q', 2), FST::RELATION('r', 2),\  FST::RELATION('s', 2), FST::RELATION('t', 2), FST::RELATION('u', 2), FST::RELATION('v', 2), FST::RELATION('w', 2), FST::RELATION('x', 2),\  FST::RELATION('y', 2), FST::RELATION('z', 2), FST::RELATION('1', 2), FST::RELATION('2', 2), FST::RELATION('3', 2), FST::RELATION('4', 2),\  FST::RELATION('5', 2), FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2), FST::RELATION('9', 2), FST::RELATION('0', 2),\  \  FST::RELATION('а', 2), FST::RELATION('б', 2), FST::RELATION('в', 2), FST::RELATION('г', 2), FST::RELATION('д', 2), FST::RELATION('е', 2),\  FST::RELATION('ё', 2), FST::RELATION('ж', 2), FST::RELATION('з', 2), FST::RELATION('и', 2), FST::RELATION('й', 2), FST::RELATION('к', 2),\  FST::RELATION('л', 2), FST::RELATION('м', 2), FST::RELATION('н', 2), FST::RELATION('о', 2), FST::RELATION('п', 2), FST::RELATION('р', 2),\  FST::RELATION('с', 2), FST::RELATION('т', 2), FST::RELATION('у', 2), FST::RELATION('ф', 2), FST::RELATION('х', 2), FST::RELATION('ц', 2),\  FST::RELATION('ч', 2), FST::RELATION('ш', 2), FST::RELATION('щ', 2), FST::RELATION('ъ', 2), FST::RELATION('ы', 2), FST::RELATION('ь', 2),\  FST::RELATION('э', 2), FST::RELATION('ю', 2), FST::RELATION('я', 2), FST::RELATION(' ', 2), FST::RELATION('.', 2), FST::RELATION(',', 2),\  FST::RELATION('?', 2), FST::RELATION('!', 2), FST::RELATION(';', 2), FST::RELATION(':', 2), FST::RELATION('-', 2), FST::RELATION(')', 2),\  FST::RELATION('(', 2)),\  \  FST::NODE(1, FST::RELATION('\'', 3)),\  FST::NODE() |
| true | #define FST\_TRUE 5,\  FST::NODE(1, FST::RELATION('t',1)),\  FST::NODE(1, FST::RELATION('r',2)),\  FST::NODE(1, FST::RELATION('u',3)),\  FST::NODE(1, FST::RELATION('e',4)),\  FST::NODE() |
| false | #define FST\_FALSE 6,\  FST::NODE(1, FST::RELATION('f',1)),\  FST::NODE(1, FST::RELATION('a',2)),\  FST::NODE(1, FST::RELATION('l',3)),\  FST::NODE(1, FST::RELATION('s',4)),\  FST::NODE(1, FST::RELATION('e',5)),\  FST::NODE() |
| . | #define FST\_POINT 2,\  FST::NODE(1, FST::RELATION('.',1)),\  FST::NODE() |
| , | #define FST\_COMMA 2,\  FST::NODE(1, FST::RELATION(',',1)),\  FST::NODE() |
| [ | #define FST\_LEFTBRACE 2,\  FST::NODE(1, FST::RELATION('[',1)),\  FST::NODE() |
| ] | #define FST\_RIGHTBRACE 2,\  FST::NODE(1, FST::RELATION(']',1)),\  FST::NODE() |
| ( | #define FST\_LEFTHESIS 2, \  FST::NODE(1, FST::RELATION('(', 1)),\  FST::NODE() |
| ) | #define FST\_RIGHTHESIS 2, \  FST::NODE(1, FST::RELATION(')', 1)),\  FST::NODE() |
| > | #define FST\_MORE 2, \  FST::NODE(1, FST::RELATION('>', 1)),\  FST::NODE() |
| < | #define FST\_LESS 2, \  FST::NODE(1, FST::RELATION('<', 1)),\  FST::NODE() |
| = | #define FST\_EQUALLY 2, \  FST::NODE(1, FST::RELATION('=', 1)),\  FST::NODE() |
| : | #define FST\_COMPARE 2, \  FST::NODE(1, FST::RELATION(':', 1)),\  FST::NODE() |
| + | #define FST\_PLUS 2, \  FST::NODE(1, FST::RELATION('+', 1)),\  FST::NODE() |
| - | #define FST\_MINUS 2, \  FST::NODE(1, FST::RELATION('-', 1)),\  FST::NODE() |
| \* | #define FST\_STAR 2, \  FST::NODE(1, FST::RELATION('\*', 1)),\  FST::NODE() |
| / | #define FST\_DIRSLASH 2, \  FST::NODE(1, FST::RELATION('/', 1)),\  FST::NODE() |

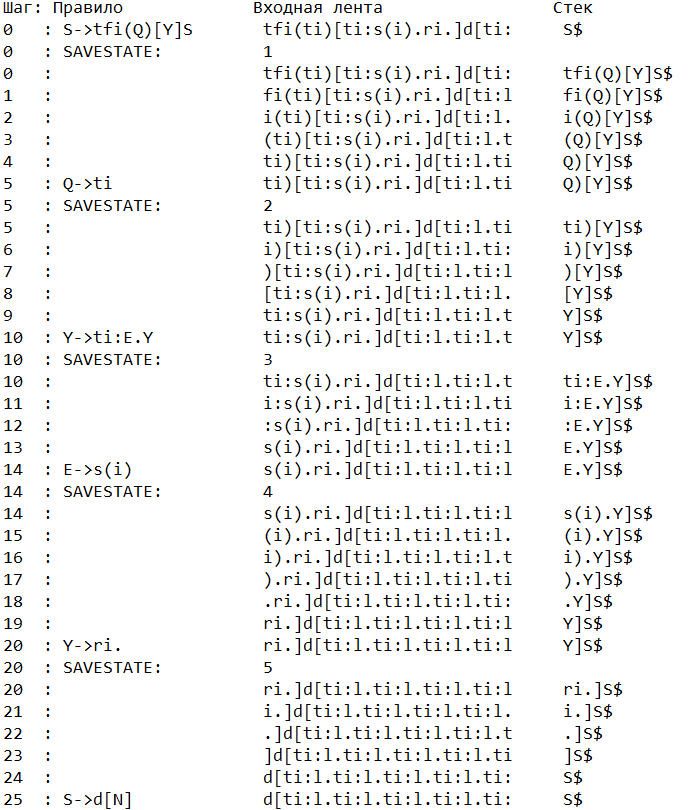
# **Приложение Б**

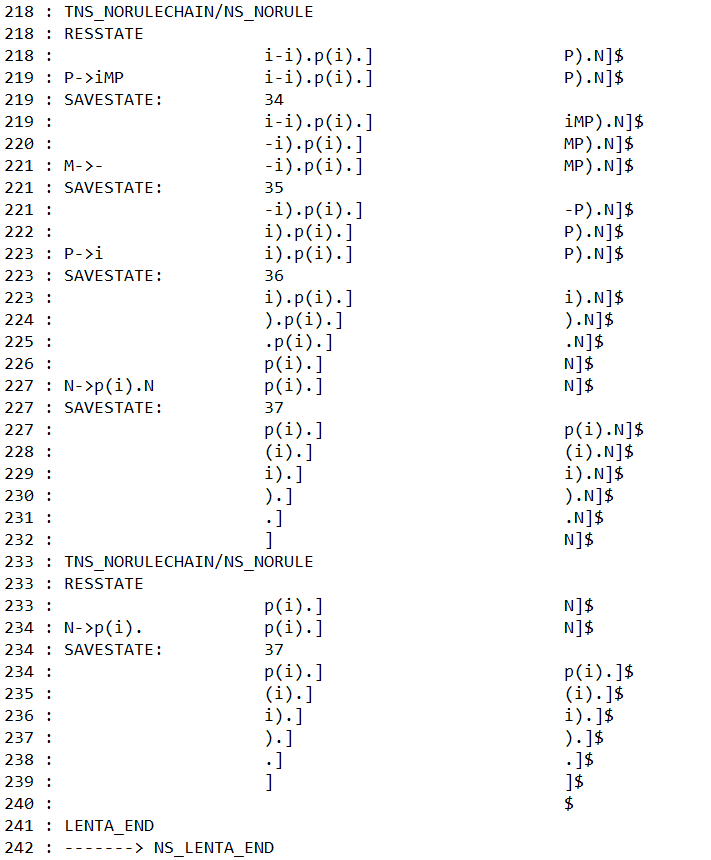
**Правила перехода нетерминальных автоматов**

|  |  |  |
| --- | --- | --- |
| Нетерминалы | Правила | Назначение правил |
| S→ | tfi(Q)[Y]S  d[N] | Определение правильности написания общей структуры программы |
| N→ | ti:E.N  ti:E.  tO.N  tO.  ti,O.N  ti,O.  ti:P.N  ti:P.  i:P.N  i:E.N  i:E.  w(K)[C]e[C]N  w(K)[C]N  w(K)[C]e[C]  w(K)[C]  p(i).N  p(i).  p(l).N  p(l). | Определение правильности написания инструкций и конструкций языка. |
| O→ | I  i , O | Определения правильности написания операции объявления |
| Q→ | Ti  ti,Q | Определяет правильность написания параметров функций. |
| E→ | I  l  f(i)  f(i)E | Определение правильности написания операции инициализации |
| Y→ | ti:E.N  ti:E.  tO.N  tO.  ti,O.N  ti,O.  ti:P.N  ti:P.  i:P.N  i:E.N  i:E.  w(K)[C]e[C]N  w(K)[C]N  w(K)[C]e[C]  w(K)[C]  p(i).N  p(i).  p(l).N  p(l).  ri.  rl. | Определяет правильность написания инструкций и конструкций языка в теле пользовательских функций |
| P→ | I  l  ip  lP  iMP  lMP  (P)  (P)MP  (P)N  (P) | Определение правильности написания выражения |
| M→ | +  -  \*  / | Определение правильности написания арифметических операторов в выражении |
| B→ | >  <  = | Определение правильности написания логических операторов в сравнении |
| K→ | L  i  iBi  iBl  lBi  lBl | Определение правильности написания в условии конструкции языка if() else |

# **Приложение В**

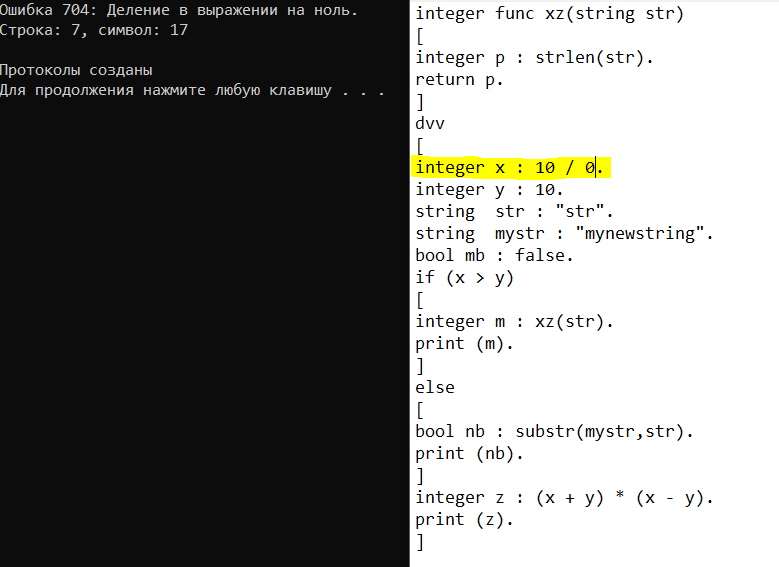
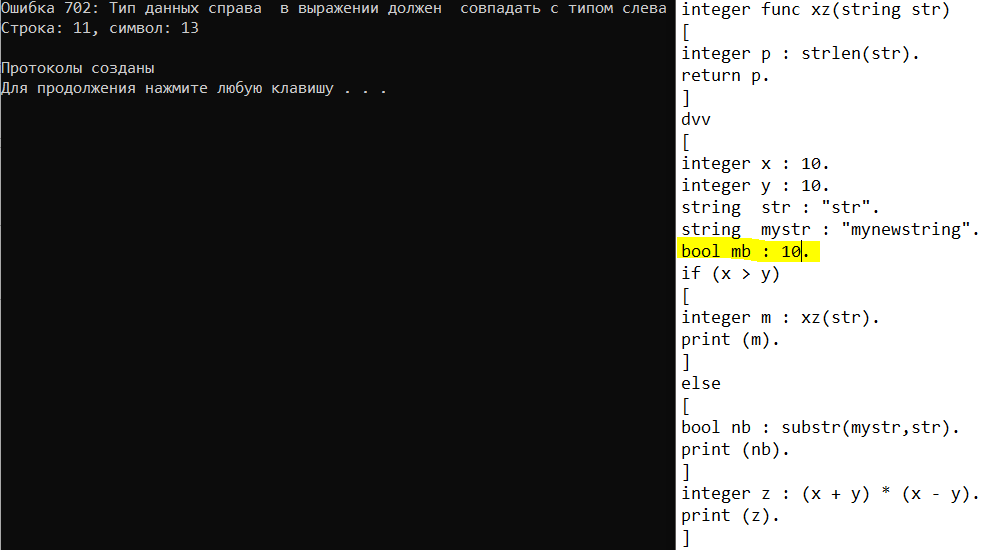
**Синтаксический анализ**

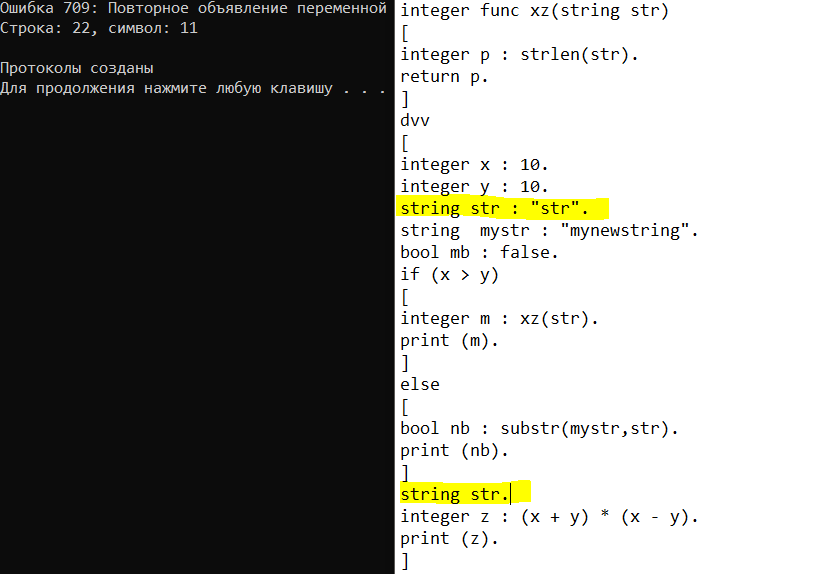




# **Приложение Г**

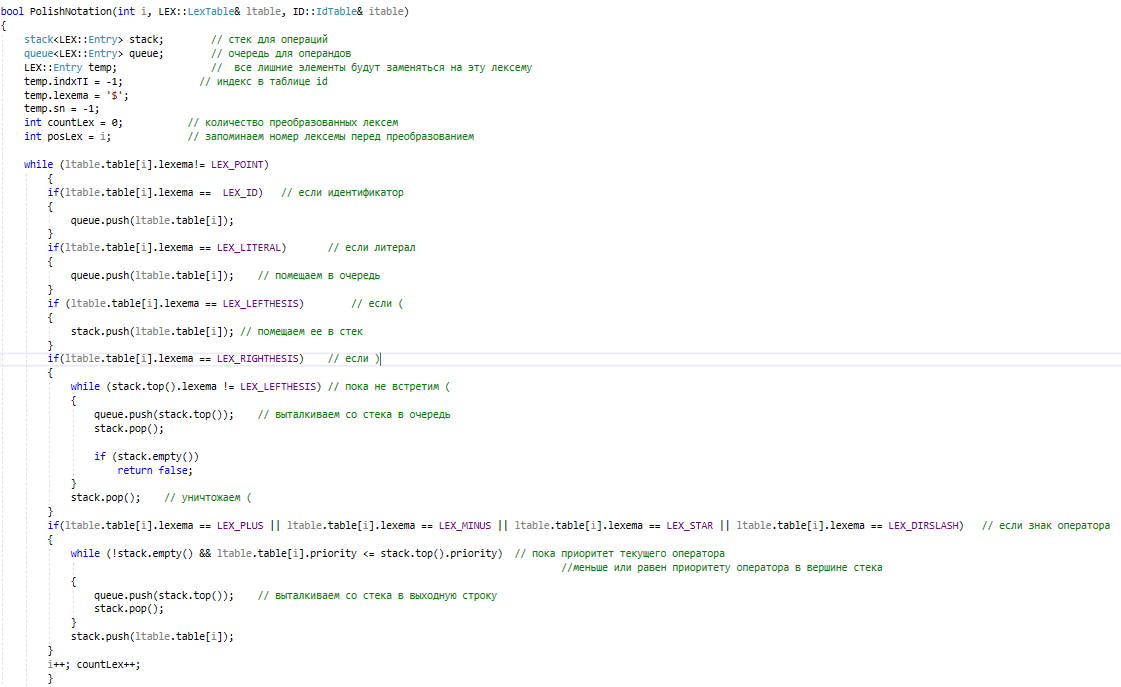
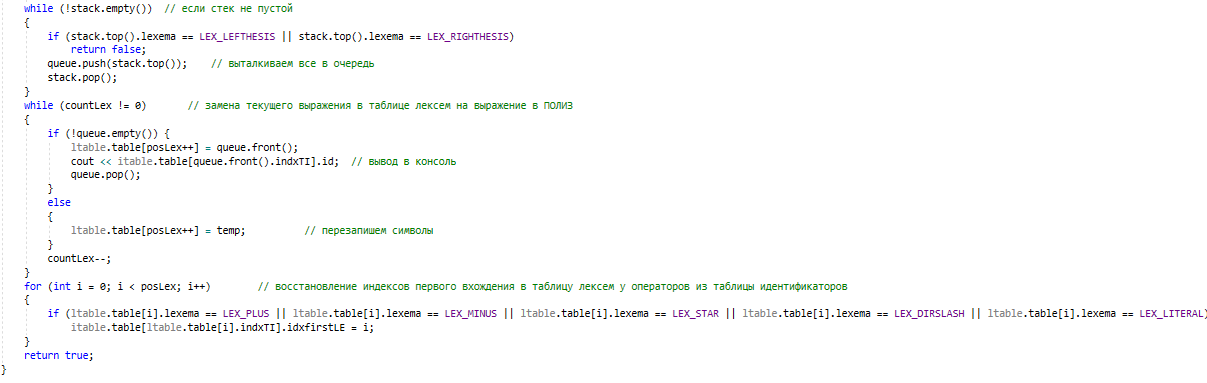
**Ошибки семантического анализатора**





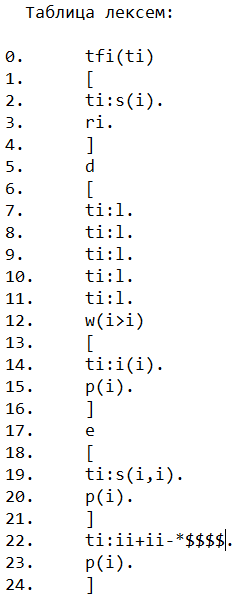
# **Приложение Д**

**Алгоритм преобразования выражений**

# **Приложение Е**

**Преобразованная таблица лексем**



# **Приложение К**

**Объектный код**

using System;

namespace CourseProject

{

class DVV2018

{

static int xz(string str)

{

int p = ClassLibrary.MyClass.strlen(str);

return p ;

}

static void Main(string[] args)

{

int x = 10;

int y = 10;

string str = "str";

string mystr = "mynewstring";

bool mb = false;

if (x > y)

{

int m = xz(str);

Console.WriteLine(m);

}

else

{

bool nb = ClassLibrary.MyClass.substr(mystr,str);

Console.WriteLine(nb);

}

int z = (x + y) \* (x - y);

Console.WriteLine(z);

while (Console.ReadKey().Key != ConsoleKey.Enter) {}

}

}

}