МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем технологий

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация 1-40 01 01 01 Программное обеспечение информационных технологий(программирование интернет приложений)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора ZDA-2021»

Выполнил студент Зворыкин Дмитрий Александрович

(Ф.И.О.)

Руководитель проекта ассист. Мущук Артур Николаевич

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

Заведующий кафедрой к.т.н., доц. Пацей Наталья Владимировна

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

Консультанты ассист. Мущук Артур Николаевич

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

Нормоконтролер ассист. Мущук Артур Николаевич

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

Курсовой проект защищен с оценкой

Содержание

[**Введение** 5](#_Toc91021635)

[**Глава 1. Спецификация языка программирования** 6](#_Toc91021636)

[**1.1** **Характеристика языка программирования** 6](#_Toc91021637)

[**1.2** **Алфавит языка** 6](#_Toc91021638)

[**1.3** **Символы сепараторы** 6](#_Toc91021639)

[**1.4** **Применяемые кодировки** 6](#_Toc91021640)

[**1.5** **Типы данных** 7](#_Toc91021641)

[**1.6** **Преобразование типов данных** 8](#_Toc91021642)

[**1.7** **Идентификаторы** 8](#_Toc91021643)

[**1.8** **Литералы** 8](#_Toc91021644)

[**1.9** **Область видимости идентификаторов** 9](#_Toc91021645)

[**1.10** **Инициализация данных** 9](#_Toc91021646)

[**1.11** **Инструкции языка** 9](#_Toc91021647)

[**1.12** **Операции языка** 10](#_Toc91021648)

[**1.13** **Выражения и их вычисления** 10](#_Toc91021649)

[**1.14** **Программные конструкции языка** 11](#_Toc91021650)

[**1.15** **Область видимости** 11](#_Toc91021651)

[**1.16** **Семантические проверки** 11](#_Toc91021652)

[**1.17** **Распределение оперативной памяти на этапе выполнения** 12](#_Toc91021653)

[**1.18** **Стандартная библиотека и её состав** 12](#_Toc91021654)

[**1.19** **Ввод и вывод данных** 12](#_Toc91021655)

[**1.20** **Точка входа** 12](#_Toc91021656)

[**1.21** **Препроцессор** 12](#_Toc91021657)

[**1.22** **Соглашения о вызовах** 12](#_Toc91021658)

[**1.23** **Объектный код** 13](#_Toc91021659)

[**1.24** **Классификация сообщений транслятора** 13](#_Toc91021660)

[**1.25** **Контрольный пример** 13](#_Toc91021661)

[**Глава 2. Структура транслятора** 14](#_Toc91021662)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 14](#_Toc91021663)

[**2.2 Перечень входных параметров** 15](#_Toc91021664)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 16](#_Toc91021665)

[**Глава 3. Разработка лексического анализатора** 17](#_Toc91021666)

[**3.1 Структура лексического анализатора** 17](#_Toc91021667)

[**3.2 Контроль входных символов** 17](#_Toc91021668)

[**3.3 Удаление избыточных символов** 18](#_Toc91021669)

[**3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов** 18](#_Toc91021670)

[**3.5 Основные структуры данных** 19](#_Toc91021671)

[**3.6 Принцип обработки ошибок** 20](#_Toc91021672)

[**3.7 Структура и перечень сообщений лексического анализатора** 20](#_Toc91021673)

[**3.8 Параметры лексического анализатора и режимы его работы** 20](#_Toc91021674)

[**3.9 Алгоритм лексического анализа** 20](#_Toc91021675)

[**3.10 Контрольный пример** 21](#_Toc91021676)

[**Глава 4. Разработка синтаксического анализатора** 22](#_Toc91021677)

[**4.1 Структура синтаксического анализатора** 22](#_Toc91021678)

[**4.2 Контекстно свободная грамматика, описывающая синтаксис языка** 22](#_Toc91021679)

[**4.3 Построение конечного магазинного автомата** 24](#_Toc91021680)

[**4.4 Основные структуры данных** 25](#_Toc91021681)

[**4.5 Описание алгоритма синтаксического разбора** 25](#_Toc91021682)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 26](#_Toc91021683)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 26](#_Toc91021684)

[**4.8 Принцип обработки ошибок** 26](#_Toc91021685)

[**4.9 Контрольный пример** 27](#_Toc91021686)

[**Глава 5. Разработка семантического анализатора** 28](#_Toc91021687)

[**5.1 Структура семантического анализатора** 28](#_Toc91021688)

[**5.2 Функции семантического анализатора** 28](#_Toc91021689)

[**5.3 Структура и перечень сообщений семантического анализатора** 28](#_Toc91021690)

[**5.4 Принцип обработки ошибок** 28](#_Toc91021691)

[**5.5 Контрольный пример** 28](#_Toc91021692)

[**Глава 6. Преобразование выражений** 30](#_Toc91021693)

[**6.1 Выражения, допускаемые языком** 30](#_Toc91021694)

[**6.2 Польская запись** 30](#_Toc91021695)

[**6.3 Программная реализация обработки выражений** 31](#_Toc91021696)

[**6.4 Контрольный пример** 31](#_Toc91021697)

[**Глава 7. Генерация кода** 32](#_Toc91021698)

[**7.1 Структура генератора кода** 32](#_Toc91021699)

[**7.2 Представление типов данных в оперативной памяти** 32](#_Toc91021700)

[**7.3 Статическая библиотека** 33](#_Toc91021701)

[**7.4 Особенности алгоритма генерации кода** 35](#_Toc91021702)

[**7.5 Входные параметры генератора кода** 35](#_Toc91021703)

[**7.6 Контрольный пример** 35](#_Toc91021704)

[**Глава 8. Тестирование транслятора** 36](#_Toc91021705)

[**8.1 Тестирование фазы проверки на допустимость символов** 36](#_Toc91021706)

[**8.2 Тестирование лексического анализатора** 36](#_Toc91021707)

[**8.3 Тестирование синтаксического анализатора** 36](#_Toc91021708)

[**8.4 Тестирование семантического анализатора** 37](#_Toc91021709)

[**Заключение** 38](#_Toc91021710)

[**Список использованной литературы** 39](#_Toc91021711)

[**Приложение А** 40](#_Toc91021712)

[**Приложение Б** 42](#_Toc91021713)

[**Приложение В** 49](#_Toc91021714)

[**Приложение Г** 54](#_Toc91021715)

[**Приложение Д** 57](#_Toc91021716)

[**Приложение Е** 59](#_Toc91021717)

[**Приложение Ж** 61](#_Toc91021718)

[**Приложение К** 62](#_Toc91021719)

[**Приложение И** 66](#_Toc91021720)

# **Введение**

Задача курсового проекта - разработка транслятора своего языка программирования ZDA-2021. Он предназначен для выполнения простейших операций и арифметических действий над числами.

Транслятор преобразует программу на языке ZDA-2021 на язык ассемблера. Язык ассемблера – это машинно-ориентированный язык, представляющий формат записи машинных команд.

Исходя из цели курсового проекта, были определены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– генерация кода на язык ассемблера;

– тестирование транслятора;

Способы решения каждой задачи будут описаны в соответствующих главах курсового проекта.

В первой главе работы определена спецификация языка программирования.

Во второй главе представлена структура транслятора.

В третьей главе описана разработка лексического анализатора, который создаёт таблицы лексем и идентификаторов.

В четвертой главе описана разработка синтаксического анализатора, который выполняет разбор исходного кода в соответствии с правилами языка программирования.

В пятой главе описан семантический анализатор, которые проверяет исходный код программы на наличие семантических ошибок.

В шестой главе описан способ преобразования выражений в польский формат.

В седьмой главе представлена генерация кода в язык ассемблера с помощью таблиц лексем и идентификаторов

В восьмой главе описывается тестирование транслятора

# **Глава 1. Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

Язык программирования ZDA-2021 – это строго типизируемый, процедурный язык высокого уровня, который транслируется в язык ассемблера.

## **1.2 Алфавит языка**

Алфавит языка программирования – набор символов, которые могут использоваться при написании исходного кода.

Язык программирования ZDA-2021 включает в себя кириллицу и символы латинского алфавита, цифры, знаки препинания, знаки арифметических и логических операций.

## **1.3 Символы сепараторы**

Сепараторы необходимы для разделения операций языка. Сепараторы, используемые в языке программирования ZDA-2021, приведены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ;  :  (пробел)  ‘  , | Разделение конструкций |
| =  +  -  \*  % | Арифметические операции |
| >  <  ~  ! | Логические операции |
| {} | Программный блок инструкций |
| () | Параметры функций, изменение приоритетов в выражениях |

## **1.4 Применяемые кодировки**

Для написания исходного кода на языке программирования ZDA-2021 используется кодировка Windows-1251 представленная на рисунке 1.1.

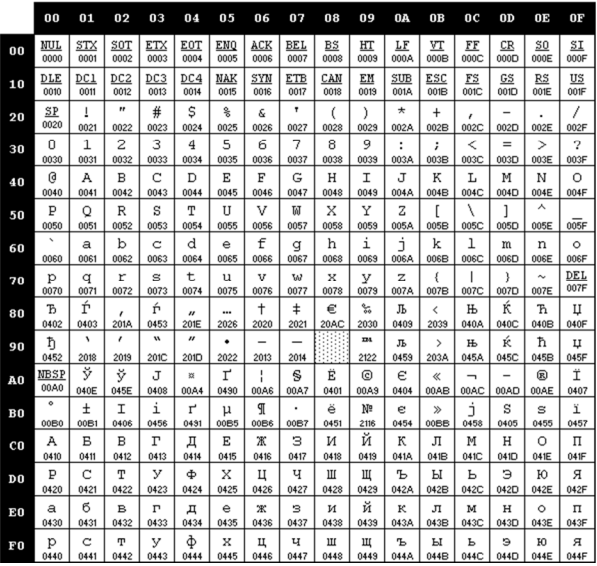


Рисунок 1.1 – Кодировка Windows-1251

## **1.5 Типы данных**

Таблица 1.2 – Типы данных

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| int | Фундаментальный тип данных, используемый для объявления целочисленных данных. Без явно указанной инициализации переменной, присваивается нулевое значение. Ограничен в размере до 1 байта то есть int находится в диапазоне от -128 до +127 |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| string | Фундаментальный тип данных, используемый для объявления строк. Без явно указанной инициализации переменной, присваивается нулевое значение (пустая строка). |
| bool | Фундаментальный тип данных, используемый для объявления логической переменной, которая принимает одно из двух значений: true или false. Без явно указанной инициализации переменной, присваивается нулевое значение (false). |

## **1.6 Преобразование типов данных**

В языке программирования ZDA-2021 преобразование типов данных не поддерживается.

## **1.7 Идентификаторы**

Идентификаторы применяются для наименования переменных, функция и параметров. Идентификаторы, объявленные внутри функционального блока, получают префикс, который отображается в таблице идентификаторов. Зарезервированные идентификаторы не поддерживаются. Предусмотрены несколько правил составления идентификатора:

– состоит из символов латинского алфавита любого регистра и цифр;

– могут начинаться с символа нижнего подчеркивания «\_»;

– максимальная длина идентификатора равна 40. При превышении длины идентификатора она будет урезаться;

– идентификатор не может совпадать с ключевыми словами языка программирования.

– регулярное выражение для разбора идентификатора выглядит следующим образом: [\_]\*[a-zA-Z0-9\_]+

## **1.8 Литералы**

В языке программирования ZDA-2021 существует только 2 типа литералов: целые и символьные. Их краткое описание представлено в таблице 1.3.

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные литералы, только rvalue. Могут быть представлены в десятичной , шестнадцатеричной, двоичной, восьмеричной системах. Для представления в шестнадцатеричной системе необходимо сделать префикс 0х,двоичной 0b,восьмеричной 0o. |

Продолжение таблицы 1.3

|  |  |
| --- | --- |
| Строковые литералы | Символы, заключенные в одинарные кавычки (‘’), только rvalue. Максимальное число символов в строковом литерале – 255. |

## **1.9 Область видимости идентификаторов**

Область видимости в языке программирования ZDA-2021 работает по принципу C++ («сверху вниз»). Перед использованием переменной необходимо её объявление. Допускается использование переменной только внутри её области видимости. Допускается объявление переменных с одинаковыми именами в разных программных блоках.

## **1.10 Инициализация данных**

Способы инициализации переменных языка программирования ZDA-2021 представлены в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| let <тип данных> <идентификатор>; | Автоматическая инициализация переменной. int – инициализируется нулем, string – пустой строкой, bool – false. |
| let <тип данных> <идентификатор> = <выражение>; | Инициализация переменной с присваиванием значения. |

## **1.11 Инструкции языка**

Все возможные инструкции языка программирования ZDA-2021 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования ZDA-2021

|  |  |
| --- | --- |
| Инструкция | Запись |
| Объявление переменной | let <тип данных> <идентификатор>;  let <тип данных> <идентификатор> = <выражение>; |
| Присваивание | <идентификатор> = <выражение >; |
| Объявление функции | <тип данных> function <идентификатор> ([<тип данных> <идентификатор>]\*,[ <тип данных> <идентификатор>]\*) {…} |
| Блок инструкций | {  …  } |

Продолжение таблицы 1.11

|  |  |
| --- | --- |
| Возврат из подпрограммы | ret <выражение>; |
| Вывод данных | write <выражение>;  writeln <выражение>; |
| Оператор цикла | while (<условие>)  {…} |
| Условный оператор | if (<условие>)  {…} |
| Условный оператор с блоком else | if (<условие>)  {…}  else  {…} |

## **1.12 Операции языка**

Язык программирования ZDA-2021 может выполнять арифметические и логические операции, представленные в таблице 1.6.

Таблица 1.6 – Операции и их приоритеты

|  |  |
| --- | --- |
| Операция | Приоритет операции |
| (  )  >  <  ~  ! | 0 |
| , | 1 |
| +  - | 2 |
| \*  / (деление с остатком)  % (деление по модулю) | 3 |

## **1.13 Выражения и их вычисления**

выражении должны участвовать операторы и операнды одного типа, а также функции, возвращающие значения того же типа. Круглые скобки в выражении используются для изменения приоритета операций. Выражение может содержать вызов функции. Фигурные скобки используются для составления блоков кода функций, операторов. Круглые – для передачи параметров функций.

## **1.14 Программные конструкции языка**

Ключевые программные конструкции языка программирования ZDA-2021 представлены в таблице 1.7

Таблица 1.7 – программные конструкции

|  |  |
| --- | --- |
| Главная функция (точка входа в приложение) | main  {…} |
| Процедура | void <идентификатор> ([<тип данных> <идентификатор>],[ <тип данных> <идентификатор>]\*)  {  …  } |
| Функция | <тип данных> function <идентификатор> ([<тип данных> <идентификатор>],[ <тип данных> <идентификатор>]\*)  {  …  ret <выражение>;  } |

## **1.15 Область видимости**

В языке программирования ZDA-2021 переменные обязаны находиться внутри программного блока функций. Внутри разных областей видимости разрешено объявление переменных с одинаковыми именами. Все переменные, параметры или функции внутри области видимости получают префикс, который отображается в таблице идентификаторов. Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

## **1.16 Семантические проверки**

Перечень семантических проверок, предусмотренных языком, приведён в таблице 1.8.

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы функций не должны повторяться |
| 2 | Идентификаторы в одной области видимости не должны повторяться |
| 3 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении |
| 4 | Тип данных функции должен совпадать с её возвращаемым значением |
| 5 | Идентификатор должен быть объявлен до его использования |

Продолжение таблицы 1.8

|  |  |
| --- | --- |
| 6 | Операнды в арифметическом выражении не могут быть разных типов |
| 7 | Не закрытый литерал |
| 8 | Либо нет main либо больше одного |

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

Все переменные размещаются в стеке. Таблицы лексем и идентификаторов размещены в структуры с выделенной под них оперативной памятью, которая очищается по окончанию работы транслятора.

## **1.18 Стандартная библиотека и её состав**

Функции стандартной библиотеки с их описанием приведены в таблице ниже Стандартная библиотека реализована на языке C++.

|  |  |
| --- | --- |
| int mpow(int num, int exp) | Возвращает число num в степени exp |
| int mrand(int min, int max) | Возвращает псевдослучайное число между num и max |

## **1.19 Ввод и вывод данных**

В языке программирования ZDA-2021 ввод данных не поддерживается. Вывод данных происходит с помощью функции write(<выражение>); или writeln(<выражение>);

## **1.20 Точка входа**

Точкой входа в языке программирования ZDA-2021 является функция main.

## **1.21 Препроцессор**

Препроцессор в языке программирования ZDA-2021 не предусмотрен.

## **1.22 Соглашения о вызовах**

В языке вызов функций происходит по соглашению о вызовах stdcall. Особенности stdcall:

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## **1.23 Объектный код**

Язык программирования ZDA-2021 транслируется в язык ассемблера.

## **1.24 Классификация сообщений транслятора**

В случае возникновения ошибки в исходном коде программы на языке программирования ZDA-2021 и выявлении её транслятором в файл протокола выводится сообщение. Классификация обрабатываемых ошибок приведена в таблице 1.9.

Таблица 1.9 – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-109 | Системные ошибки |
| 200-299 | Ошибки лексического анализа |
| 600-699 | Ошибки синтаксического анализа |
| 300-399 | Ошибки семантического анализа |

## **1.25 Контрольный пример**

Контрольные примеры представлены в приложении А.

# **Глава 2. Структура транслятора**

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

Транслятор преобразует программу, написанную на языке программирования ZDA-2021 в программу на языке ассемблера. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на языке ассемблера. Принцип взаимодействия представлен на рисунке 2.1.



Рисунок 2.1 – Структура транслятора

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся последовательность символов входного языка. Он производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив отдельных слов (в теории компиляции вместо термина «слово» часто используют термин «токен»). Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация. Таблица лексем (ТЛ) и таблица идентификаторов (ТИ) являются входом для следующей фазы компилятора – синтаксического анализа (разбора, парсера).

Цели лексического анализатора:

− убрать все лишние пробелы;

− выполнить распознавание лексем;

− построить таблицу лексем и таблицу идентификаторов;

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

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

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть проверку исходного кода на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов.

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

## **2.2 Перечень входных параметров**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 Входные параметры транслятора языка ZDA-2021

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя файла> | Входной файл с расширением .txt, в котором содержится исходный код на языке ZDA-2021 | Не предусмотрено |
| -log:<имя файла> | Файл для записи протокола работы транслятора | <имя файла>.log |
| -out:<имя файла> | Файл для записи результата работы транслятора | <имя файла>.out.asm |
| -tokens | Ключ для вывода промежуточного представления кода | Отсутствует |
| -TL | Ключ для вывода таблицы лексем в консоль | Отсутствует |
| -TI | Ключ для вывода таблицы идентификаторов в консоль | Отсутствует |
| -rules | Ключ для вывода трассировки синтаксического анализатора в консоль | Отсутствует |

## **2.3 Перечень протоколов, формируемых транслятором и их содержимое**

Перечень протоколов, формируемых транслятором языка программирования ZDA-2021 и их назначением представлен в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка ZDA-2021

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования KDV-2019 . 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, c расширением ".asm" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

# **Глава 3. Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает на вход исходный код языка программирования ZDA-2021, которые предварительно был обработан, т. е. были удалены пробелы, табуляция и комментарии, а символы перехода строки заменены прямой чертой. Если лексический анализатор не может разобрать отдельные последовательности символов, он генерирует исключение. При успешном анализе на выходе формируются таблицы лексем и идентификаторов. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 Структура лексического анализатора ZDA-2021

## **3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2.

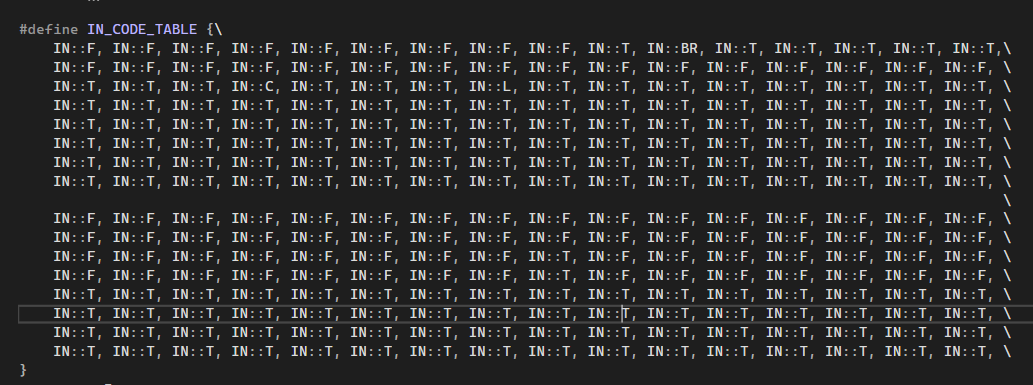


Рисунок 3.2. Таблица контроля входных символов

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенный | T |
| Запрещенный | F |
| Игнорируемый | I |
| Комментарий (#) | C |
| Литерал(‘) | L |
| Перевод строки | BR |

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции, пробела и все символы, идущие после символа начала комментария (#), если он не включен в строковый литерал. Избыточные символы удаляются перед этапом разбиения исходного кода на лексемы.

## **3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов**

Лексемы – это специальные символы, которые соответствуют ключевым словам, символам арифметических и логических операций, сепараторам и т.д. Они необходимы для упрощения дальнейшего анализа исходного кода программы. Соответствие цепочек символов и лексем приведено в таблице 3.1.

Таблица 3.1 Соответствие цепочек с лексемами

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | let | d |
| int, string, bool | t |
| main | m |
| function | f |
| void | v |
| ret | r |
| write | o |
| writeln | b |
| while | u |
| if | w |
| else | ! |
| Данные | Лексема | l |
| Идентификатор | i |

Продолжение таблицы 3.1

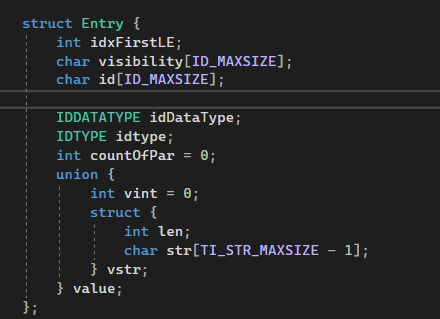
|  |  |  |
| --- | --- | --- |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| Операции | Арифметические (+,-,/,\*,%) и логические (>, <, ~, !) | p |
| Присваивания (=) | = |

Пример реализации таблицы лексем представлен в приложении Б.

Также в приложении В находятся конечные автоматы ключевых слов.

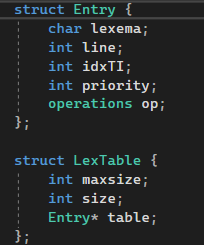
## **3.5 Основные структуры данных**

Основная структура данных, используемая для хранения таблицы идентификаторов представлена в листинге 3.1.



Листинг 3.1. — Структура таблицы идентификаторов

Основная структура данных, используемая для хранения таблицы лексем представлена в листинге 3.2.



Листинг 3.2. — Структура таблицы лексем

## **3.6 Принцип обработки ошибок**

При нахождении ошибок, транслятор пытается продолжить свою работу до завершения текущего этапа, чтобы вывести набор ошибок. При нахождении критической ошибки работа транслятора прекращается.

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

Перечень сообщений лексического анализатора представлен на рисунке 3.3.

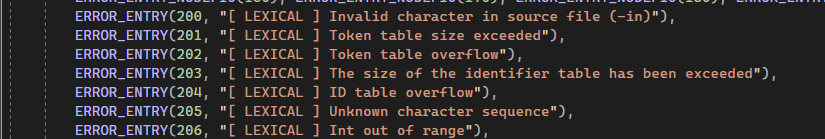


рисунок 3.3 – Перечень ошибок лексического анализатора

## **3.8 Параметры лексического анализатора и режимы его работы**

Входным параметром функции лексического анализатора является исходный текст программы, написанный на языке ZDA-2021, а также файл протокола.

## **3.9 Алгоритм лексического анализа**

Лексически анализ выполняется программой, входящей в состав транслятора, называемой лексическим анализатором. Цель лексического анализатора – распознавание цепочек символов языка и деление их на лексемы. Распознавание цепочек происходит благодаря конечным автоматам.

Пример. Конечный автомат для распознавания ключевого слова string приведен на рисунке 3.3. S0 – начальное состояние, S6 – конечное состояние автомата.

Рисунок 3.2 Пример графа переходов для цепочки string

s t r i n g

## **3.10 Контрольный пример**

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

# **Глава 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**

В синтаксическом анализаторе транслятора языка ZDA-2021 используется контекстно-свободная грамматика , где

T – множество терминальных символов (было описано в разделе 1.2 данной пояснительной записки),

N – множество нетерминальных символов (первый столбец таблицы 4.1),

P – множество правил языка (второй столбец таблицы 4.1),

S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика имеет нормальную форму Грейбах, т.к. она не леворекурсивная (не содержит леворекурсивных правил) и правила  имеют вид:

1. , где ; (или , или )
2. , где — начальный символ, при этом если такое правило существует, то нетерминал  не встречается в правой части правил.

Правила языка ZDA-2021 представлены в приложении Г.

TS – терминальные символы, которыми являются сепараторы, знаки арифметических операций и некоторые строчные буквы.

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

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов ZDA-2021

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | tfiFBS  viF{N}S  m{N} | Проверка правильности структуры программы |
| F | (P)  () | Проверка наличия параметров функции |
| P | ti  ti,P | Проверка на правильность параметров функции при её объявлении |
| B | {NrI;}  {rI;}  {N} | Проверка наличия тела функции |
| I | i l | Проверка на недопустимое выражение |
| N | dti;N  dti=E;N  i=E;N  u(R){X}N  w(R){X}N  w(R){X}!{X}N  oI;N  bI;N  rI;N  iK;N  dti;  dti=E;  i=E;  u(R){X}  w(R){X}  w(R){X}!{X}  oE;  bE;  rI;  iK; | Проверка на правильность конструкции в теле функции |
| R | i  ivi  ivl  lvi | Проверка на правильность в условном выражении |
| K | (W)  () | Проверка на правильность вызова функции |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| E | i  l  (E)  iK  iM  lM  (E)M  iKM | Проверка на правильность арифметического выражения |
| W | i  l  i,W  l,W | Проверка на правильность параметров вызываемой функции |
| M | vE  vEM | Проверка на правильность арифметических действий |
| X | dti;X  dti=E;X  i=E;X  oI;X  bI;X  rI;X  iK;X  dti;  dti=E;  i=E;  oE;  bE;  rE;  iK;  w(R){X}  w(R){X}X  w(R){X}!{X}  w(R){X}!{X}X | Проверка на правильность конструкции в теле цикла или условного выражения |

## **4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку, описание которой представлено в таблице 4.2. Структура данного автомата показана в приложении Д.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Компонента | | Определение | Описание | |
|  | Множество состояний автомата | | | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | | | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |
|  | Алфавит специальных магазинных символов | | | Алфавит магазинных символов содержит стартовый символ и маркер дна стека |
|  | Функция переходов автомата | | | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | | | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ А) |
|  | 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 включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающей правила языка ZDA-2021. Данные структуры представлены в приложении Д.

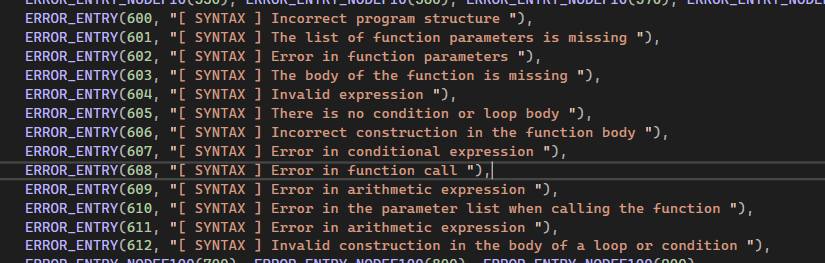
## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата:

1. В магазин записывается стартовый символ;
2. На основе полученных ранее таблиц формируется входная лента;
3. Запускается автомат;
4. Выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
5. Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала;
6. Если в магазине встретился нетерминал, переходим к пункту 4;
7. Если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

## **4.6 Структура и перечень сообщений синтаксического анализатора**

Перечень сообщений синтаксического анализатора представлен в рисунке 4.1.

рисунок 4.1 – Перечень сообщений синтаксического анализатора

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

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

Выходными параметрами являются трассировка прохода таблицы лексем и правила разбора, которые записываются в файл протокола.

## **4.8 Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

1. Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
2. Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка.
3. Все ошибки записываются в общую структуру ошибок.
4. В случае нахождения ошибки, после всей процедуры трассировки в протокол будет выведено диагностическое сообщение.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода предоставлен в приложении Е в виде фрагмента трассировки и дерева разбора исходного кода.

# **Глава 5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ в трансляторе выделен в отдельную фазу, идущую после синтаксического анализа. Он имеет ряд семантических проверок, которые помогают определить логические несоответствия конструкций языка программирования ZDA-2021.

## 

## **5.2 Функции семантического анализатора**

За семантический анализ отвечает функция Analyze. На вход она принимает таблицы лексем и идентификаторов, а также поток вывода в протокол.

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

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

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.1.

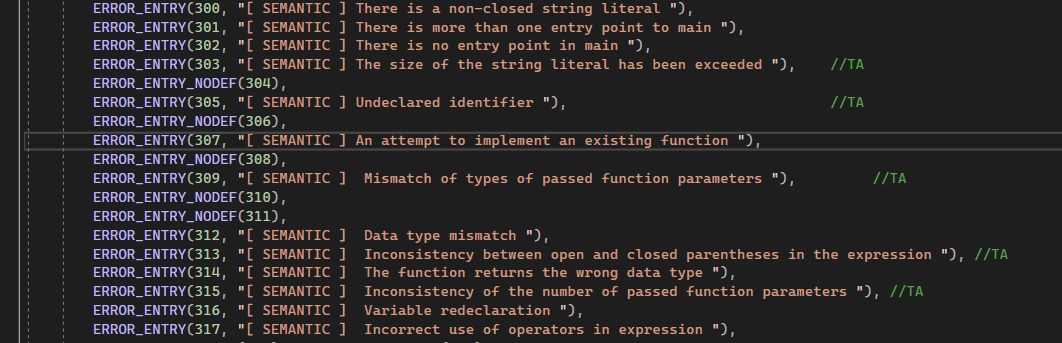


рисунок 5.1 – Перечень сообщений семантического анализатора

## **5.4 Принцип обработки ошибок**

Принцип обработки ошибок семантического анализатора основан на перебирании лексем. Когда встречается лексема, последующая цепочка которой может содержать семантическую ошибку, начинается обращение к таблице идентификаторов. Например, при нахождении лексемы «=», анализатор проверяет последующую цепочку до конца строки на ряд семантических ошибок.

## **5.5 Контрольный пример**

Результат работы контрольного примера расположен в приложении Б, где показан результат лексического анализатора, т.к. представленные таблицы лексем и идентификаторов проходят лексическую и семантическую проверки одновременно.

Таблица 5.3 – Тестирование функций

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| {  let int i = 19;    } | Ошибка 302: [ SEMANTIC ] No entry point |
| main  {  Let int i = ‘AAA’;  } | Ошибка 312: [ SEMANTIC ] Data type mismatch |
| int function i()  {  ret 0;  }  int function i()  {  ret 1;  }  main  {  let int res = i();  } | Ошибка 307: [ SEMANTIC ] An attempt to implement an existing function |

# **Глава 6. Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке ZDA-2021 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, /, % (остаток от деления), логические операции, как >, <, ! (не равно), ~ (равно) и (), а также вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке ZDA-2021

|  |  |
| --- | --- |
| Приоритет | Операция |
| 0 | ( ) > < ~ ! |
| 1 | , |
| 2 | + - |
| 3 | \* / % |

## **6.2 Польская запись**

Выражения в языке ZDA-2021 преобразовываются к обратной польской записи.

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

Обратная польская запись — это форма записи математических выражений, в которой операции расположены после своих операндов. Выражение в обратной польской нотации читается слева направо: операция выполняется над двумя операндами, непосредственно стоящими перед знаком этой операции.

Алгоритм построения:

– исходная строка: выражение;

– результирующая строка: польская запись;

– стек: пустой;

– результирующая строка: польская запись;

– исходная строка просматривается слева направо;

– операнды переносятся в результирующую строку в порядке их следования;

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

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

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

– отрывающая скобка помещается в стек;

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

– закрывающая скобка с приоритетом, равным 4, выталкивает все до открывающей с таким же приоритетом и генерирует @ – специальный символ, в которого записывается информация о вызываемой функции, а в поле приоритета для данной лексемы записывается число параметров вызываемой функции;

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

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| a+b\*func(i) |  |  |
| +b\*func(i) | a |  |
| b\*func(i) | a | + |
| \*func(i) | ab | + |
| func(i) | ab | +\* |
| (i) | ab | +\* |
| i) | ab | +\* |
| ) | abi | +\* |
|  | abi@1\*+ |  |

## **6.3 Программная реализация обработки выражений**

Программная реализация функции перевода в обратную польскую инверсию содержится в функции startPolish, которая принимает параметром таблицу лексем. Она содержит цикл, который при нахождении символа присваивания (=) вызывает функцию polishNotation и преобразует последующее выражение до конца строки.

После завершение функции startPolish происходит синхронизация индексов таблицы идентификаторов с таблицей лексем, так как лексемы меняют свое положение.

## **6.4 Контрольный пример**

Пример преобразования выражения к польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления.

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

# **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 ассемблерным кодом.



Рисунок 7.1 Структура генератора кода

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

Некоторые элементы таблицы идентификаторов располагаются в сегментах языка ассемблера. Идентификаторы языка программирования ZDA-2021 расположены в сегменте данных (.data). Литералы расположены в сегменте констант (.const). Соответствие между типами данных языка программирования ZDA-2021 и языка ассемблера приведены в таблице 7.1. Сгенерированный код приведен в приложении К.

Таблица 7.1 – Соответствия типов идентификаторов языка ZDA-2021 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке ZDA-2021 | Тип идентификатора на языке ассемблера | Пояснение |
| int | SDWORD | Хранит целочисленный тип данных со знаком(но ограничение на один байт реализовано через библиотечные функции). |
| string | BYTE | Каждый символ строки типа string хранится в поле размером 1 байт. |

## **7.3 Статическая библиотека**

Статическая библиотека реализована на языке программирования C++. Её реализация находится в проекте StaticLibrary, в свойствах которого был выбран пункт «статическая библиотека .lib».

Подключение библиотеки в языке ассемблера происходит с помощью директивы includelib на этапе генерации кода. Далее с помощью оператора EXTRN объявляются имена функций из библиотеки. Оператор EXTRN выполняет две функции. Во-первых, он сообщает ассемблеру, что указанное символическое имя является внешним для текущего ассемблирования. Вторая функция оператора EXTRN состоит в том, что он указывает ассемблеру тип соответствующего символического имени. Так как ассемблирование является очень формальной процедурой, то ассемблер должен знать, что представляет из себя каждый символ. Это позволяет ему генерировать правильные команды

|  |  |
| --- | --- |
| void BREAKL() {  std::cout << std::endl;  } | Переводит на новую строку |
| void OutputInt(int a) {  if (a > 127 || a < -128)  {  std::cout << "Int out of range";  throw;  }  std::cout << a;  } | Выводит в консоль число a и осуществляет проверку на вхождение a в диапазон байта |
| void OutputStr(char\* ptr)  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  if (ptr == nullptr) {  std::cout << std::endl;  return;  }  for (int i = 0; ptr[i] != '\0'; i++)  std::cout << ptr[i];  } | Выводить в консоль литерал |
| void OutputIntLn(int a) {  if (a > 127 || a < -128)  {  std::cout << "Int out of range";  throw;  }  std::cout << a << std::endl;  }  void OutputStrLn(char\* ptr) {  OutputStr(ptr);  std::cout << std::endl;  } | Тоже самое что и две функции выше но в конце в консоль выводит символ перехода на новую строку |

Продолжение таблицы

|  |  |
| --- | --- |
| int mpow(int num, int exp) {  return pow(num, exp);  } | Возращает число num в степени exp |
| int mrand(int min, int max) {  return rand() % (max - min) + min;  } | Возращает случайное число от min до max |
| int sum(int a, int b)  {  if ((a + b) > 127 || (a + b) < -128)  {  std::cout << "Int out of range";  throw;  }  return a + b;  } | Возвращает сумму a и b с проверкой  результата на вхождение в диапазон  байта |
| int multip(int a, int b)  {  if ((a \* b) > 127 || (a \* b) < -128)  {  std::cout << "Int out of range";  throw;  }  return a \* b;  } | Возвращает результат умножения a и b с проверкой  результата на вхождение в диапазон  байта |
| int minus(int b, int a)  {  if ((a - b) > 127 || (a - b) < -128)  {  std::cout << "Int out of range";  throw;  }  return (a - b);  } | Возвращает результат разности a и b с проверкой  результата на вхождение в диапазон  байта |

## **7.4 Особенности алгоритма генерации кода**

В языке ZDA-2021 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 ZDA-2021. 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 И. Результат работы контрольного примера приведён на рисунке 7.2.

# **Глава 8. Тестирование транслятора**

## **8.1 Тестирование фазы проверки на допустимость символов**

В языке программирования ZDA-2021 не разрешается использовать запрещенные входным алфавитом символы. Результат использования запрещенного символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  let int € = 12;    } | Ошибка 200: [ LEXICAL ] Invalid character in source file (-in) |

## **8.2 Тестирование лексического анализатора**

На этапе лексического анализа 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  {  let int @test = 1;      } | Ошибка 205: [ LEXICAL ] Unknown character sequence |

## **8.3 Тестирование синтаксического анализатора**

На этапе синтаксического анализа 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 |
| int function f (int test = 123)  {  ret test;    }  main  {  let int i = f(10);  } | 600: Line 1, [ SYNTAX ] Incorrect program structure |

## **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора приведены в пункте 5.5.

# **Заключение**

В данном курсовом проекте были выполнены все требования. В ходе работы было изучено много нового, а также закреплены знания, которые были получены ранее. Данный курсовой проект позволил совместить закрепление знаний сразу по двум языкам программирования, таких как C++ и Ассемблер. Был получен примитивный язык программирования ZDA-2021, который имеет ограниченный набор конструкций.

Окончательная версия языка ZDA-2021 включает:

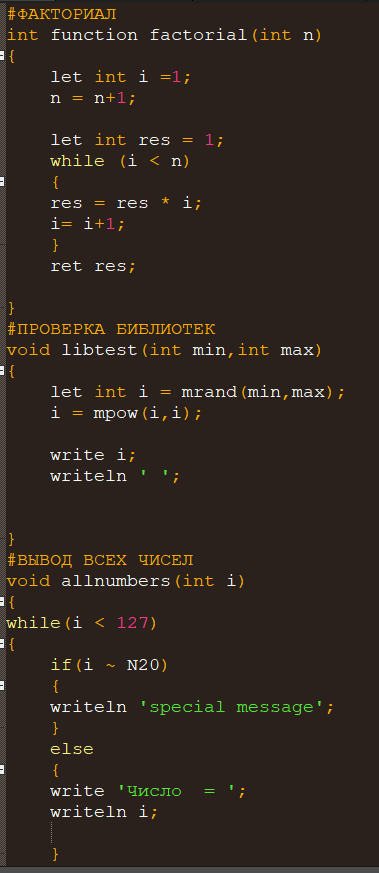
1. 3 типа данных;
2. Поддержка операторов вывода;
3. Вызов и создание функций стандартной библиотеки;
4. Наличие 4 арифметических и 4 операторов сравнения для вычисления выражений;
5. Поддержка функций, процедур, операторов цикла и условия;
6. Возможность вызова функции в выражении;
7. Структурированная и классифицированная система для обработки ошибок пользователя.

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

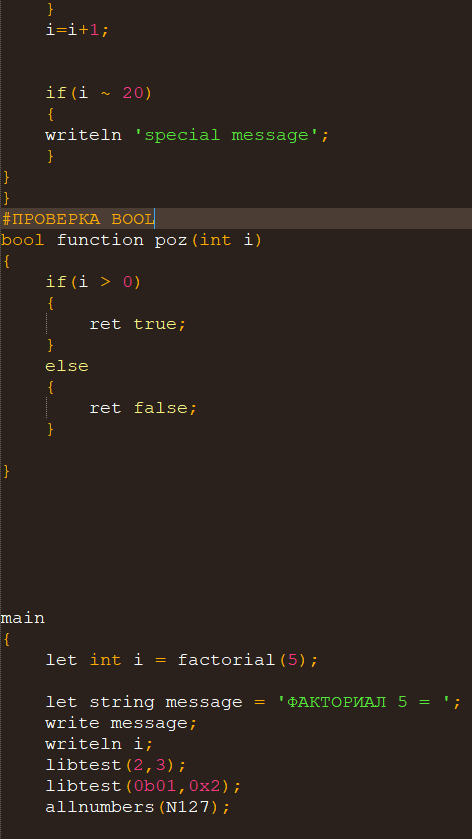
# **Список использованной литературы**

1. Ахо А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
2. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.
3. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.
4. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.

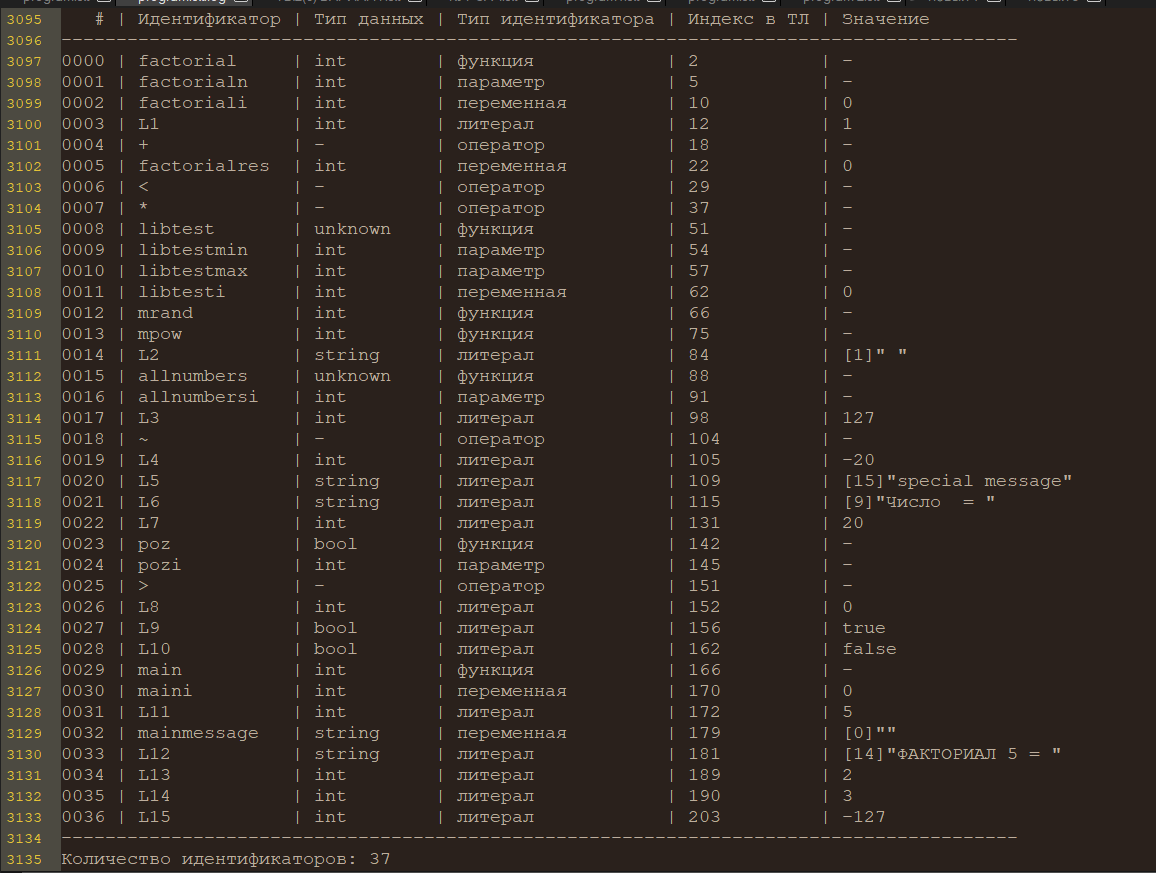
# **Приложение А**

****

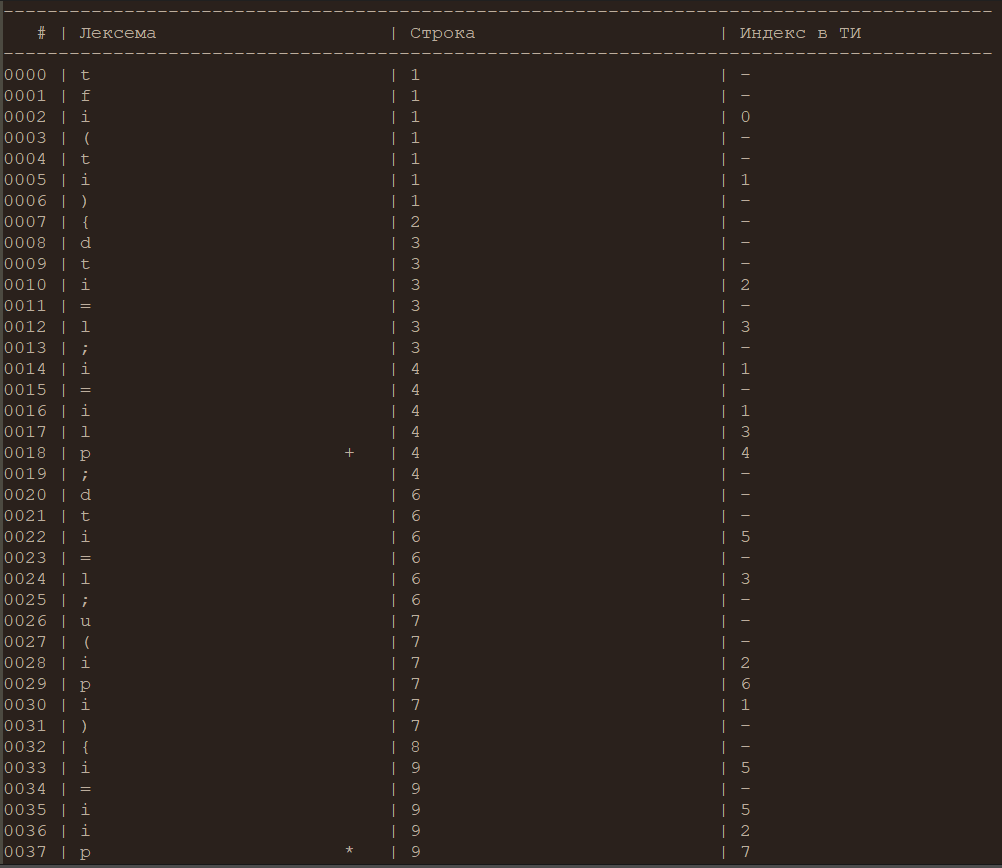
**Продолжение**

****

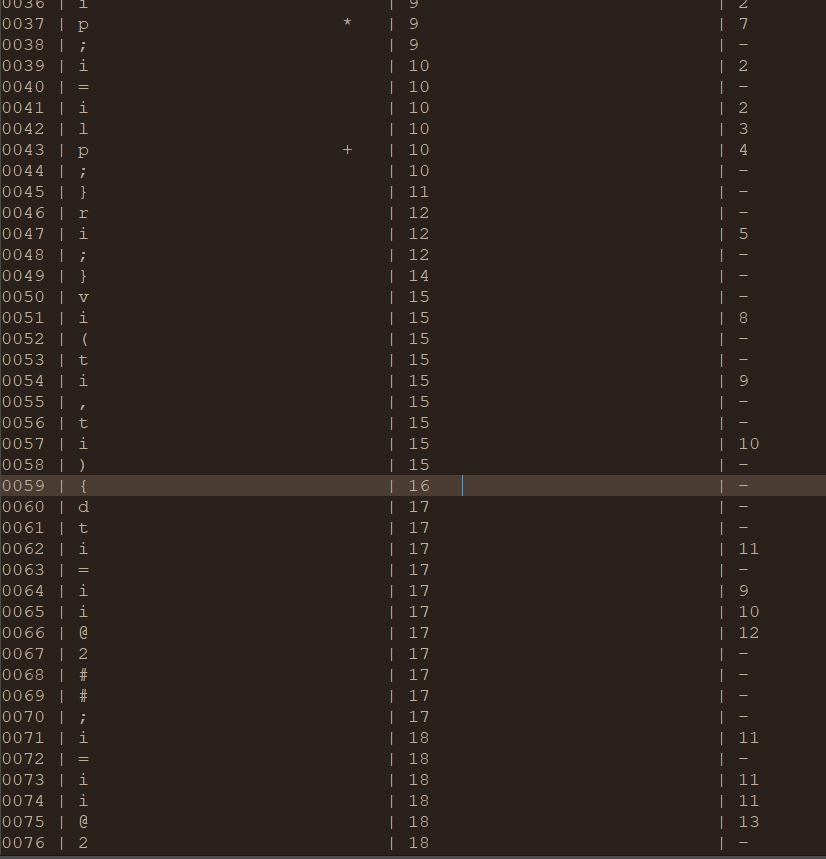
# **Приложение Б**

****

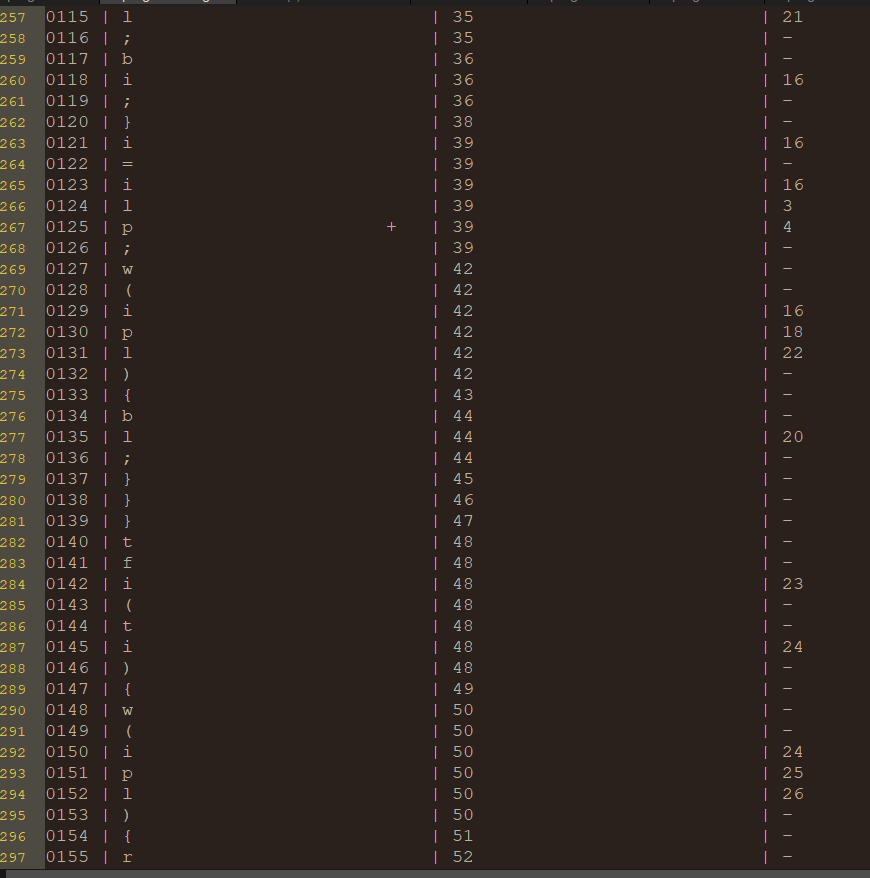
Продолжение

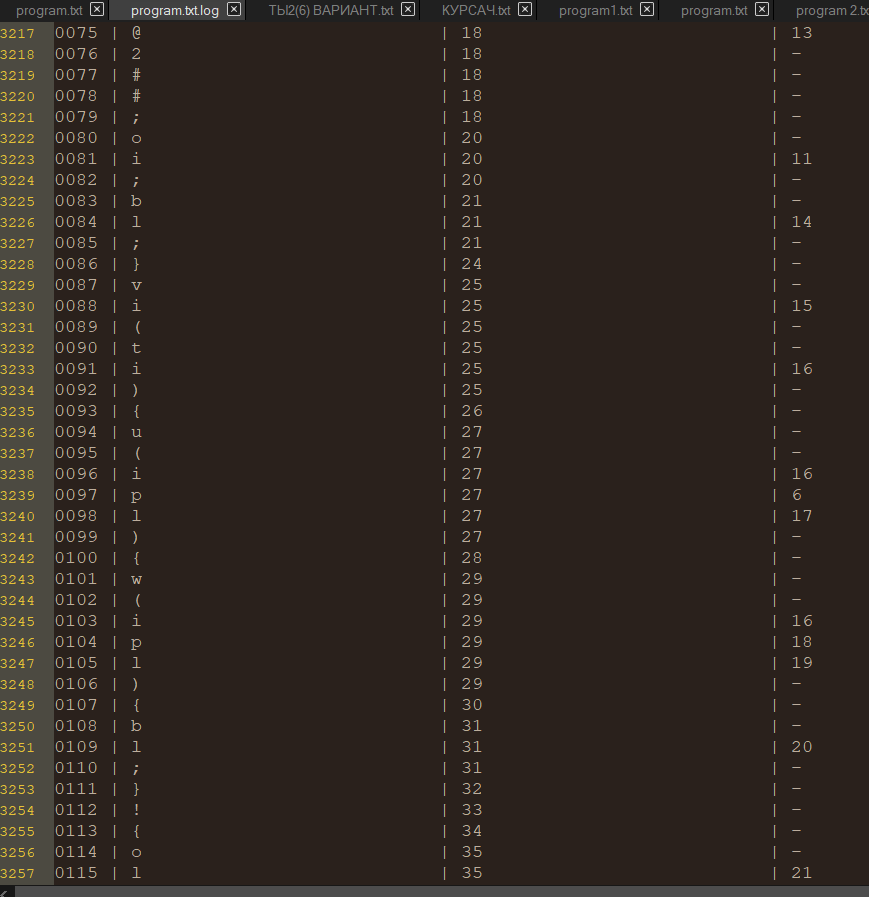
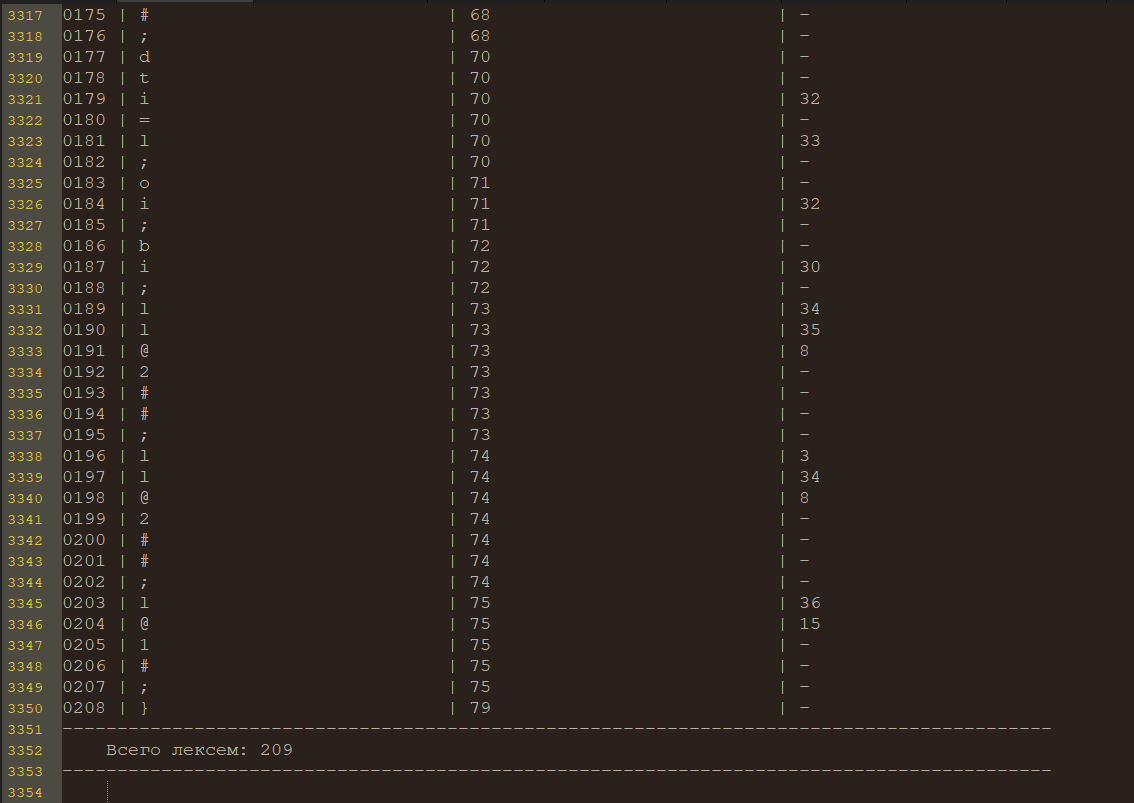


Продолжение



Продолжение

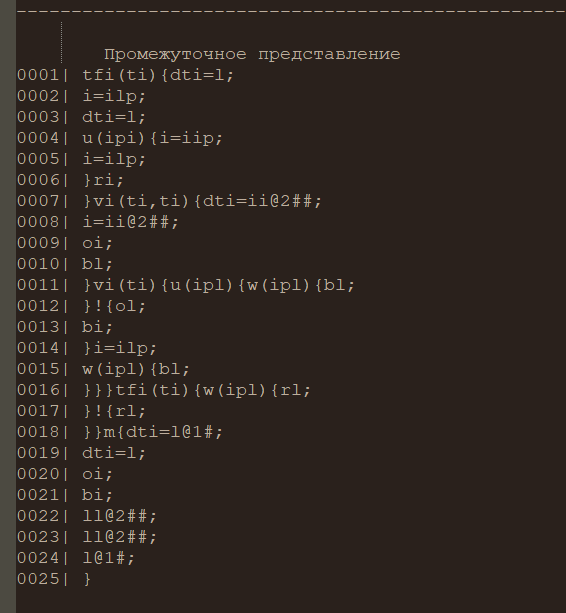




Продолжение

Продолжение

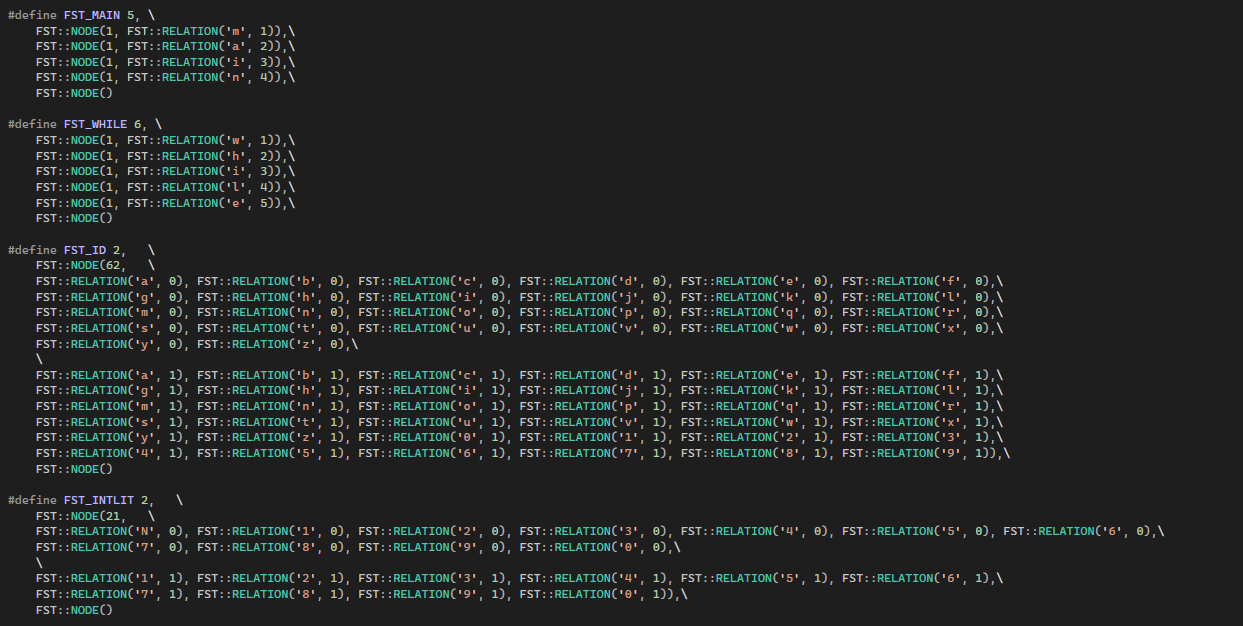
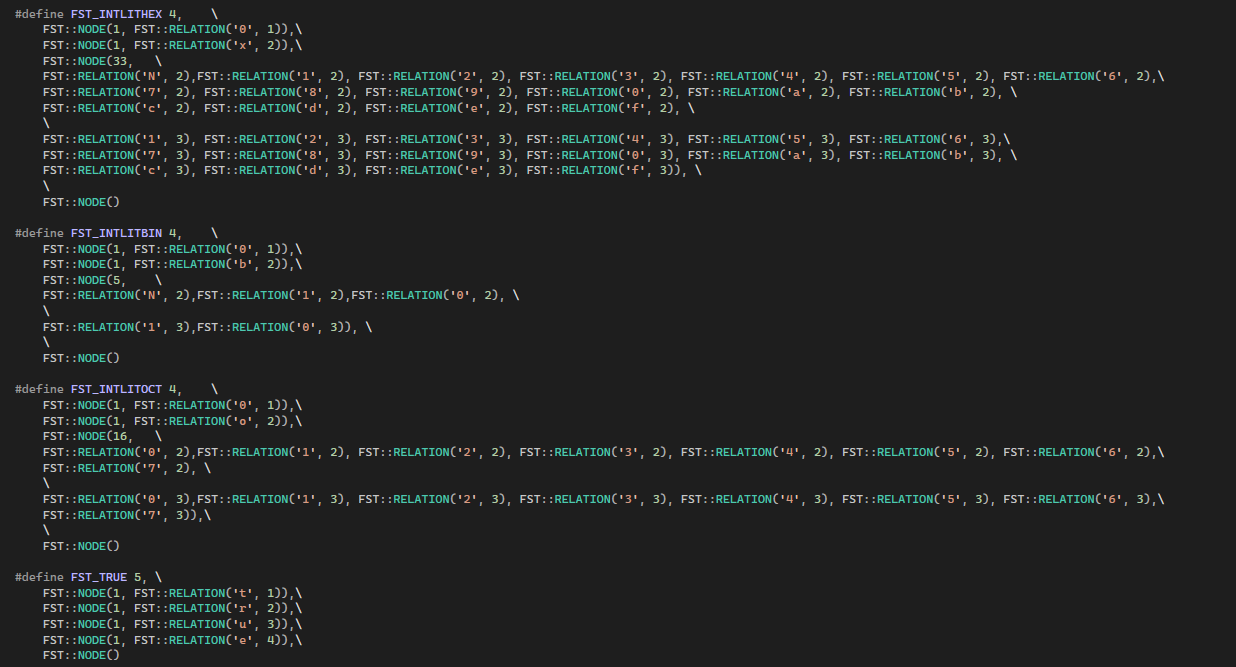
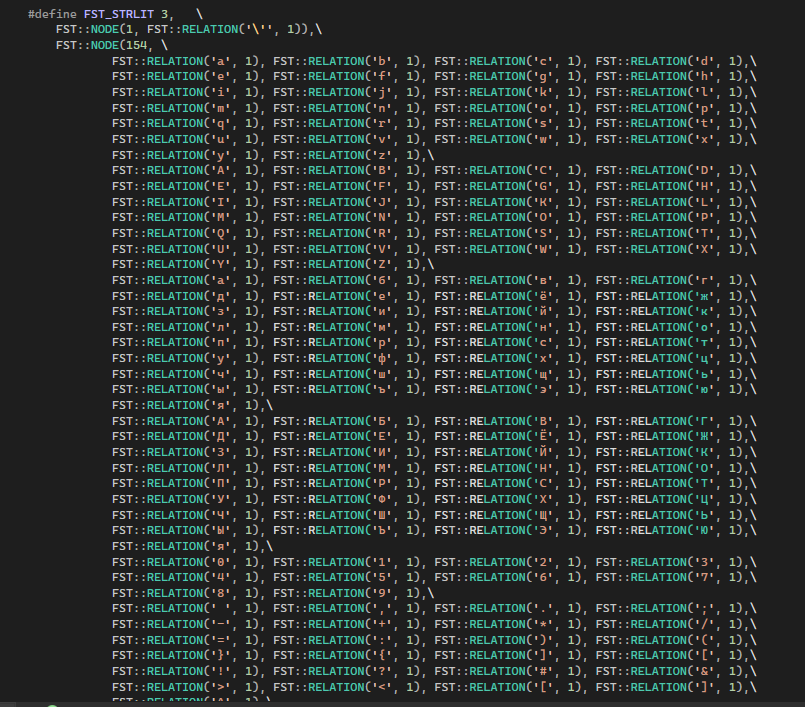
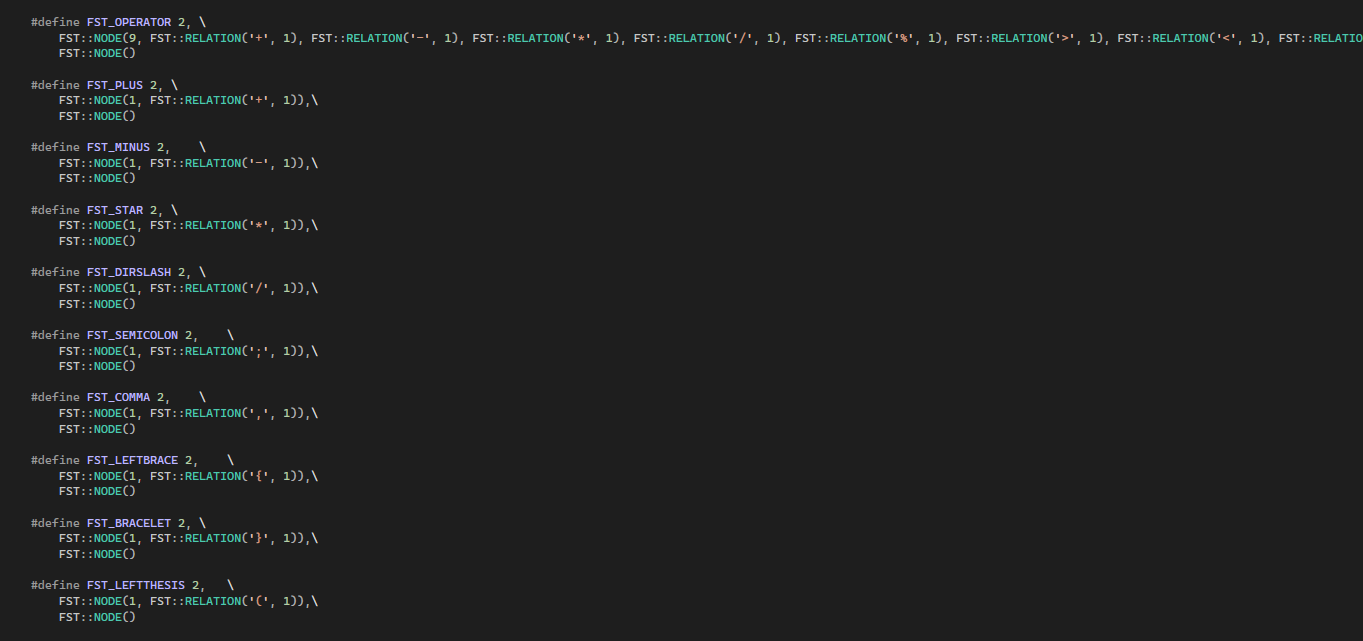
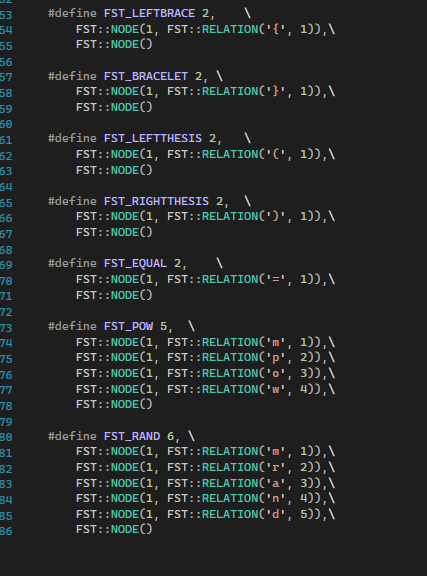
Продолжение



Продолжение

# **Приложение В**

# 

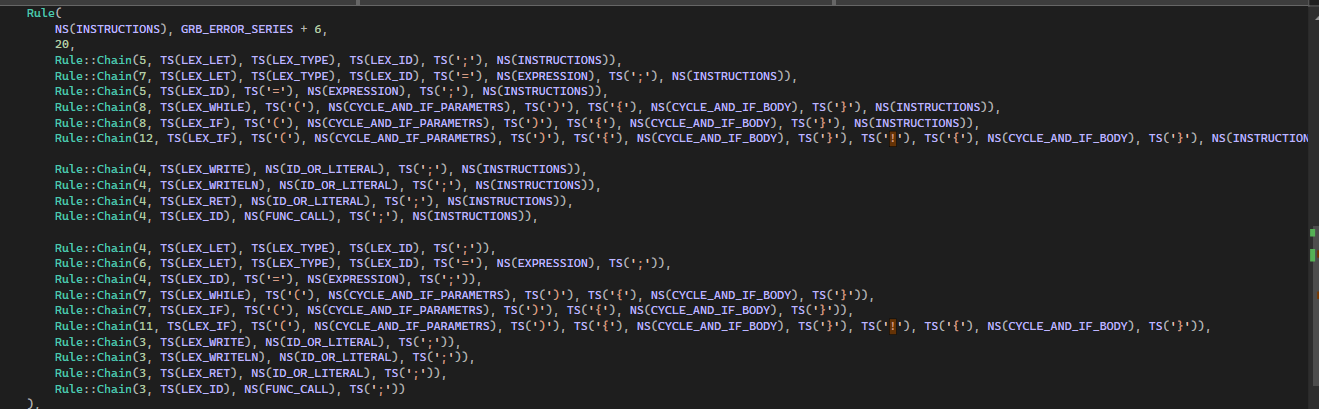
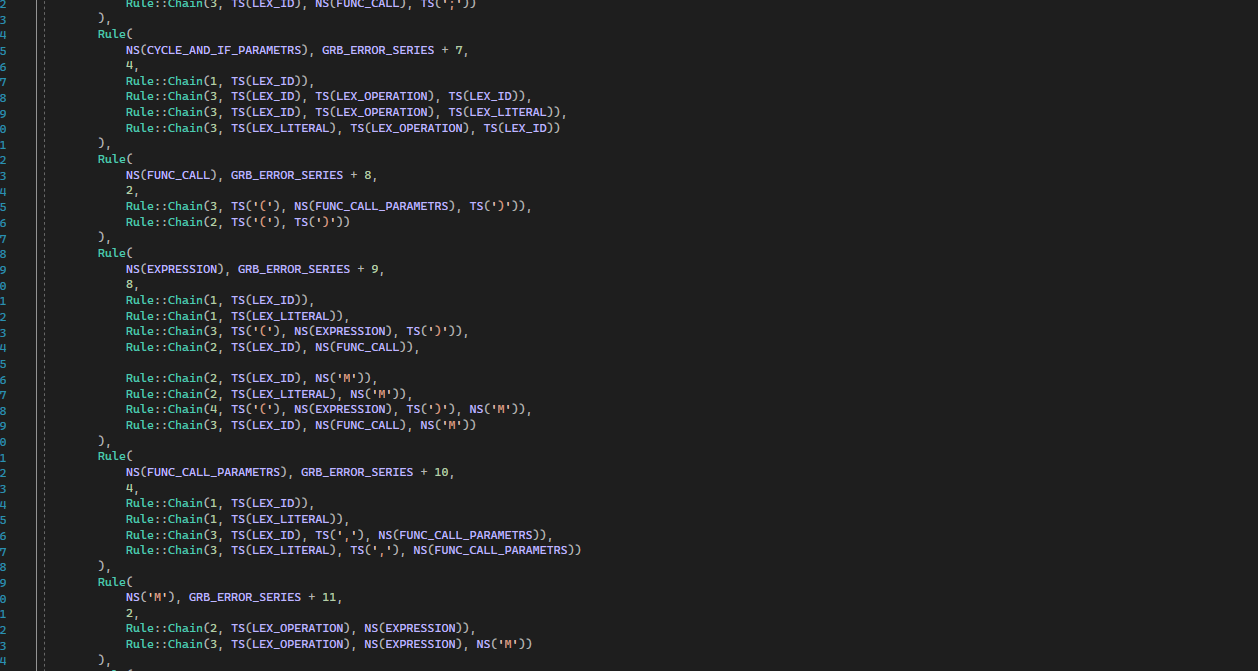
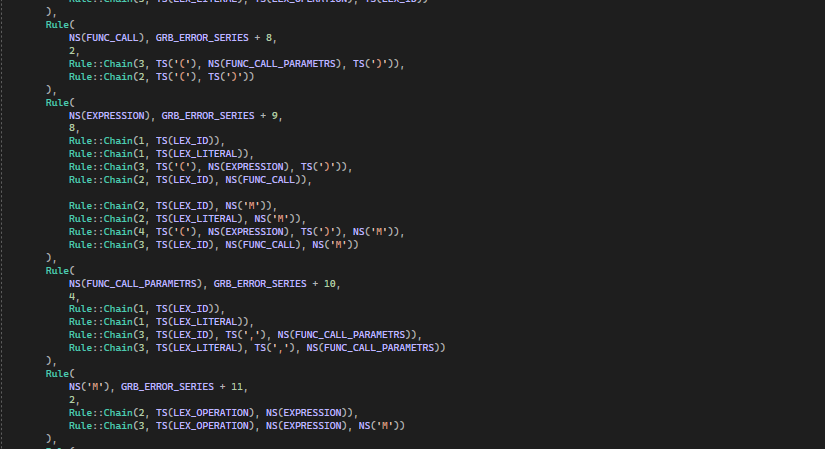
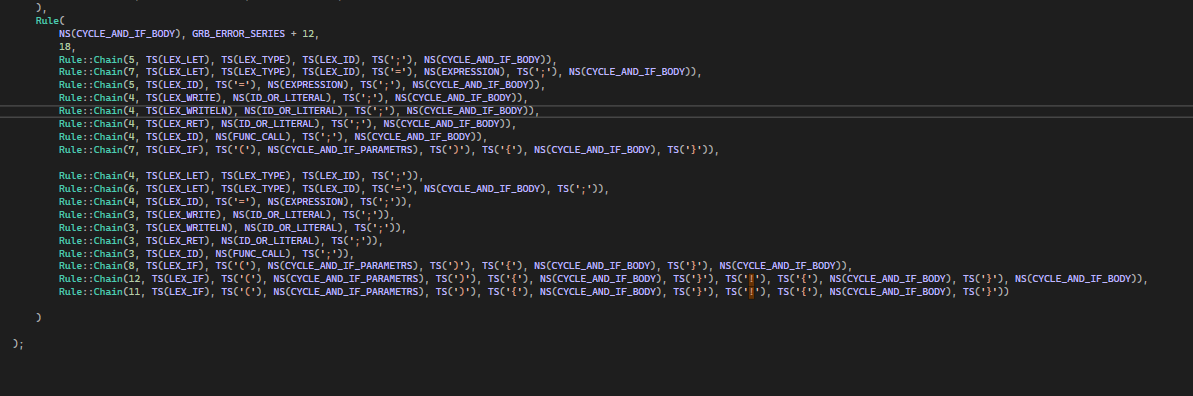
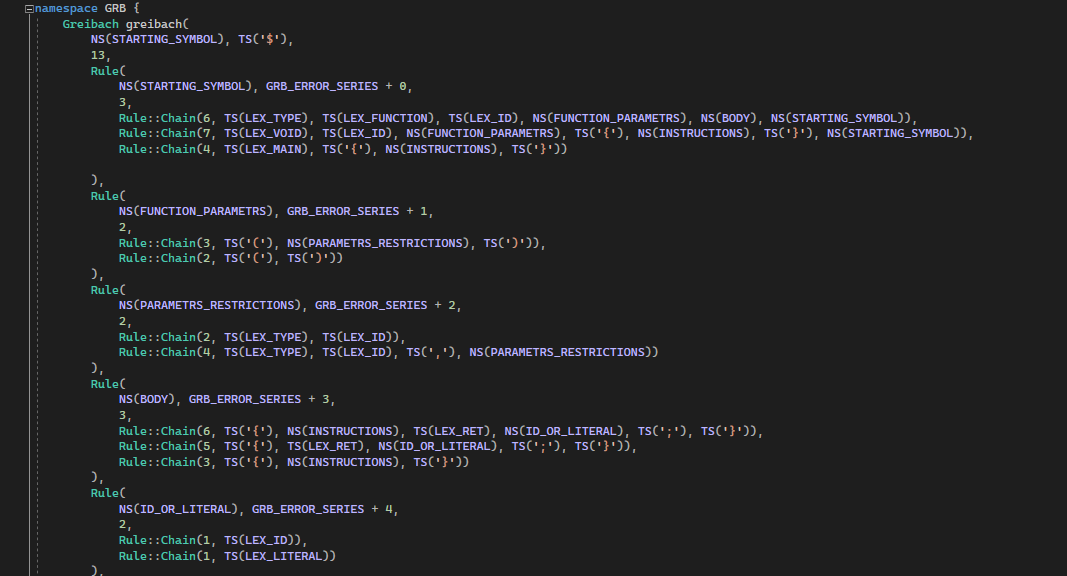
     

Продолжение

Продолжение

Продолжение

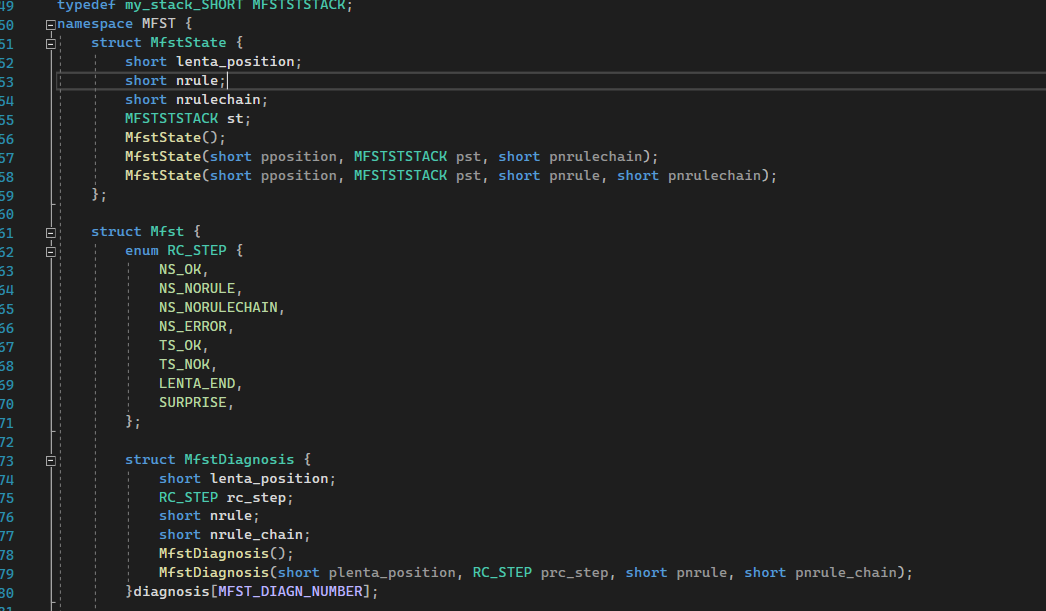
# **Приложение Г**

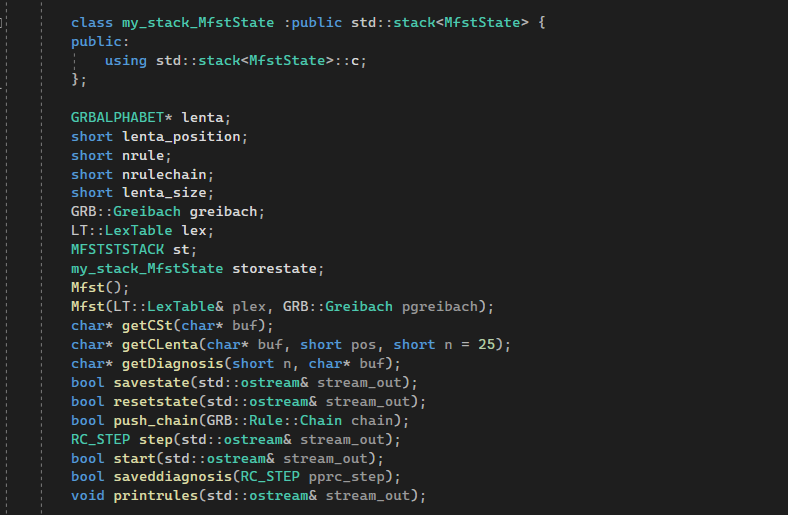
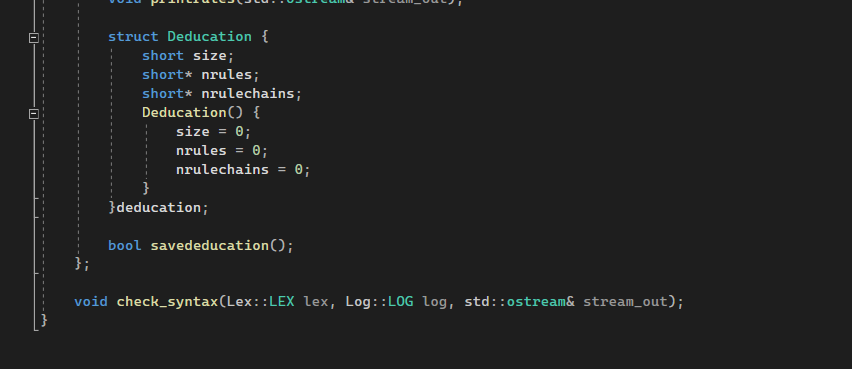
    

Продолжение

Продолжение

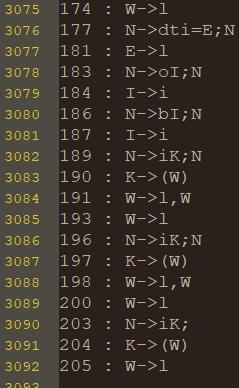
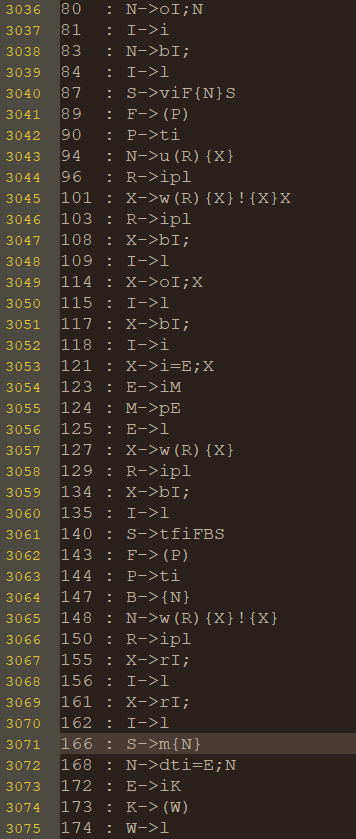
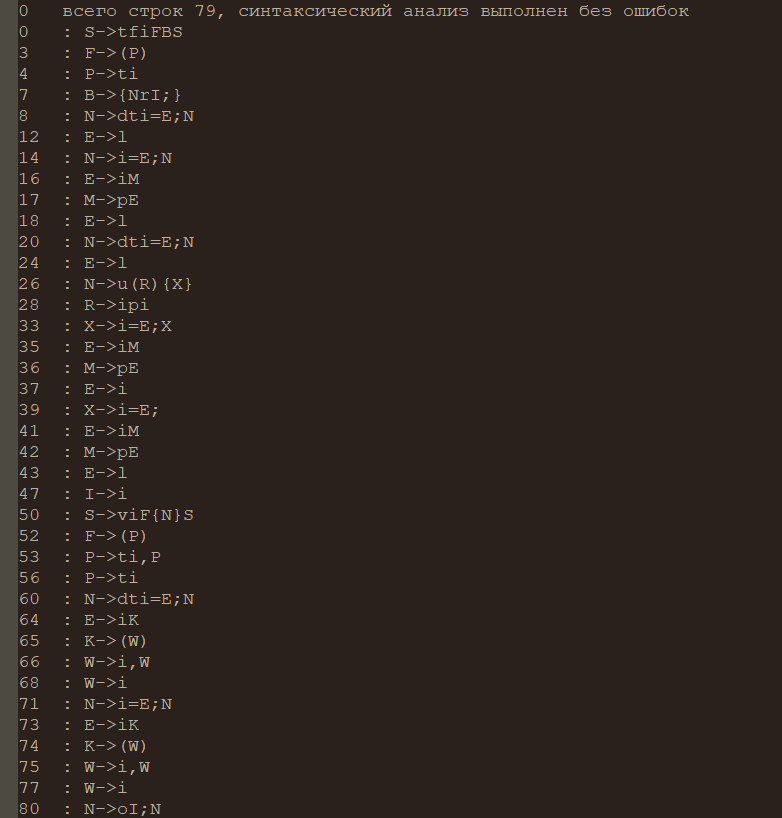
# **Приложение Д**



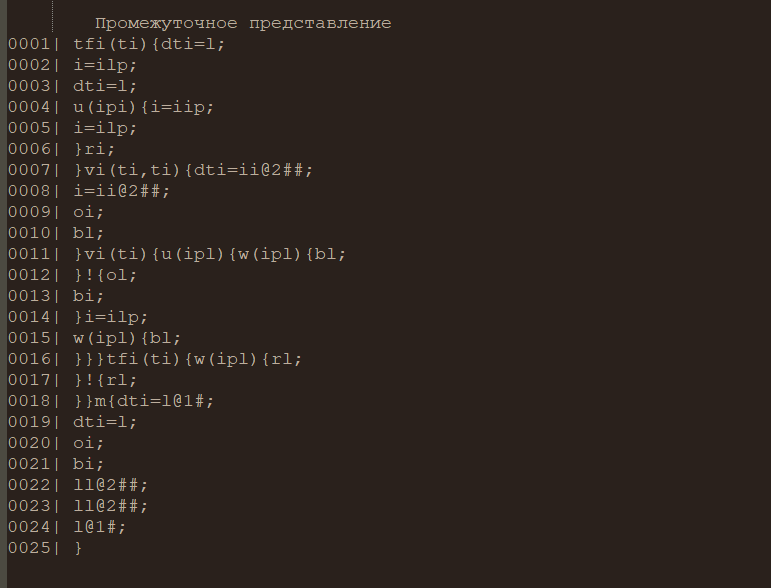
 

Продолжение

# **Приложение Е**



# **Приложение Ж**



# **Приложение К**

.586

.model flat, stdcall

includelib libucrt.lib

includelib kernel32.lib

includelib ../Debug/StaticLibrary.lib

ExitProcess PROTO :DWORD

EXTRN BREAKL: proc

EXTRN OutputInt: proc

EXTRN OutputStr: proc

EXTRN OutputIntLn: proc

EXTRN OutputStrLn: proc

EXTRN sum: proc

EXTRN multip: proc

EXTRN mpow: proc

EXTRN mrand: proc

EXTRN minus: proc

.stack 4096

.const

L1 SDWORD 1

L2 BYTE " ", 0

L3 SDWORD 127

L4 SDWORD -20

L5 BYTE "special message", 0

L6 BYTE "Число = ", 0

L7 SDWORD 20

L8 SDWORD 0

L9 SDWORD 1

L10 SDWORD 0

L11 SDWORD 5

L12 BYTE "ФАКТОРИАЛ 5 = ", 0

L13 SDWORD 2

L14 SDWORD 3

L15 SDWORD -127

.data

buffer BYTE 256 dup(0)

factoriali SDWORD 0

factorialres SDWORD 0

libtesti SDWORD 0

maini SDWORD 0

mainmessage DWORD ?

.code

factorial PROC factorialn : SDWORD

push L1

pop factoriali

push factorialn

push L1

call sum

push eax

pop factorialn

push L1

pop factorialres

mov eax, factoriali

Продолжение

cmp eax, factorialn

jl cycle0

jmp cyclenext0

cycle0:

push factorialres

push factoriali

call multip

push eax

pop factorialres

push factoriali

push L1

call sum

push eax

pop factoriali

mov eax, factoriali

cmp eax, factorialn

jl cycle0

cyclenext0:

push factorialres

jmp local0

local0:

pop eax

ret

factorial ENDP

libtest PROC libtestmin : SDWORD, libtestmax : SDWORD

push libtestmin

push libtestmax

pop edx

pop edx

push libtestmax

push libtestmin

call mrand

push eax

pop libtesti

push libtesti

push libtesti

pop edx

pop edx

push libtesti

push libtesti

call mpow

push eax

pop libtesti

push libtesti

call OutputInt

push offset L2

call OutputStrLn

ret

libtest ENDP

allnumbers PROC allnumbersi : SDWORD

mov eax, allnumbersi

cmp eax, L3

jl cycle1

jmp cyclenext1

cycle1:

mov eax, allnumbersi

cmp eax, L4

jz m0

jnz m1

Продолжение

je m1

m0:

push offset L5

call OutputStrLn

jmp e0

m1:

push offset L6

call OutputStr

push allnumbersi

call OutputIntLn

e0:

push allnumbersi

push L1

call sum

push eax

pop allnumbersi

mov eax, allnumbersi

cmp eax, L7

jz m2

jnz m3

je m3

m2:

push offset L5

call OutputStrLn

m3:

mov eax, allnumbersi

cmp eax, L3

jl cycle1

cyclenext1:

ret

allnumbers ENDP

poz PROC pozi : SDWORD

mov eax, pozi

cmp eax, L8

jg m4

jl m5

je m5

m4:

push 1

jmp local1

jmp e1

m5:

push 0

jmp local1

e1:

local1:

pop eax

ret

poz ENDP

main PROC

push L11

pop edx

push L11

call factorial

push eax

pop maini

push offset L12

pop mainmessage

push mainmessage

Продолжение

call OutputStr

push maini

call OutputIntLn

push L14

push L13

call libtest

push L13

push L1

call libtest

push L15

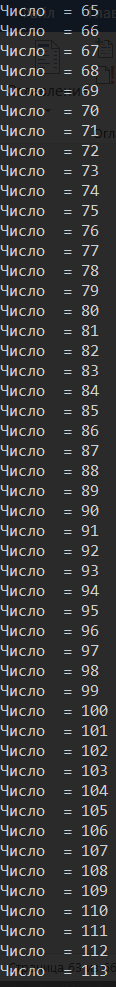
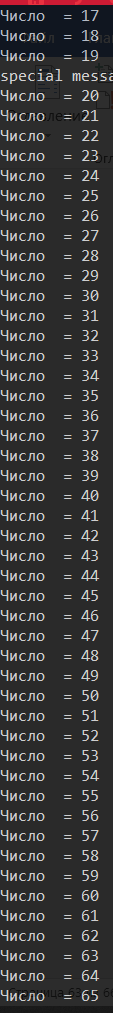
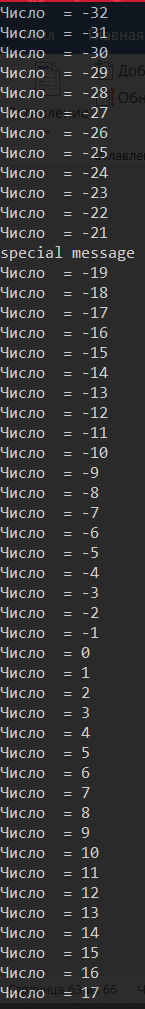
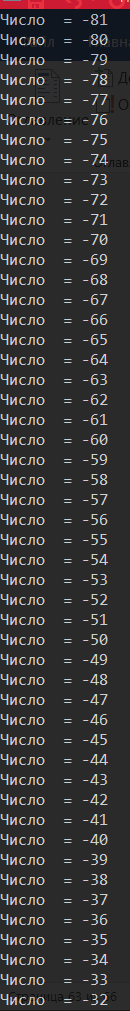
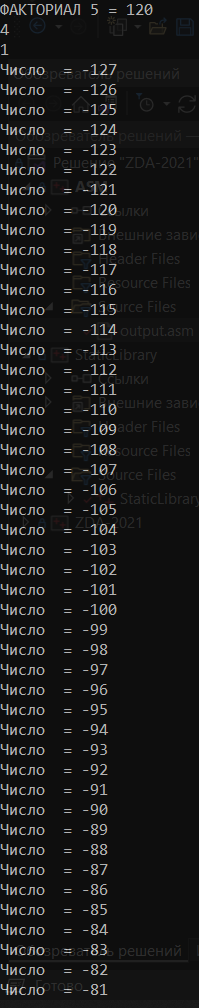
call allnumbers

call ExitProcess

main ENDP

end main

# **Приложение И**



Продолжение

