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 GDD-2023»

Выполнил студент Глушко Денис Дмитриевич

(Ф.И.О.)

Руководитель проекта ст. преп. Наркевич Аделина Сергеевна

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

Заведующий кафедрой к.т.н., доц. Смелов В. В.

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

Консультанты ст. преп. Наркевич Аделина Сергеевна

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

Нормоконтролер ст. преп. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[**Введение** 6](#_Toc153735388)

[**1. Спецификация языка программирования** 7](#_Toc153735389)

[**1.1.** **Характеристика языка программирования** 7](#_Toc153735390)

[**1.2.** **Алфавит языка** 7](#_Toc153735391)

[**1.3.** **Символы сепараторы** 7](#_Toc153735392)

[**1.4.** **Применяемые кодировки** 7](#_Toc153735393)

[**1.5.** **Типы данных** 8](#_Toc153735394)

[**1.6.** **Преобразование типов данных** 9](#_Toc153735395)

[**1.7.** **Идентификаторы** 9](#_Toc153735396)

[**1.8.** **Литералы** 9](#_Toc153735397)

[**1.9.** **Область видимости идентификаторов** 10](#_Toc153735398)

[**1.10.** **Инициализация данных** 10](#_Toc153735399)

[**1.11.** **Инструкции языка** 10](#_Toc153735400)

[**1.12.** **Операции языка** 10](#_Toc153735401)

[**1.13.** **Выражения и их вычисления** 11](#_Toc153735402)

[**1.14.** **Программные конструкции языка** 11](#_Toc153735403)

[**1.15.** **Область видимости** 12](#_Toc153735404)

[**1.16.** **Семантические проверки** 12](#_Toc153735405)

[**1.17.** **Распределение оперативной памяти на этапе выполнения** 12](#_Toc153735406)

[**1.18.** **Стандартная библиотека и её состав** 12](#_Toc153735407)

[**1.19.** **Ввод и вывод данных** 13](#_Toc153735408)

[**1.20.** **Точка входа** 13](#_Toc153735409)

[**1.21.** **Препроцессор** 13](#_Toc153735410)

[**1.22.** **Соглашения о вызовах** 13](#_Toc153735411)

[**1.23.** **Объектный код** 13](#_Toc153735412)

[**1.24.** **Классификация сообщений транслятора** 14](#_Toc153735413)

[**1.25.** **Контрольный пример** 14](#_Toc153735414)

[**2. Структура транслятора** 15](#_Toc153735415)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 15](#_Toc153735416)

[**2.2 Перечень входных параметров транслятора** 15](#_Toc153735417)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 16](#_Toc153735418)

[**3. Разработка лексического анализатора** 17](#_Toc153735419)

[**3.1 Структура лексического анализатора** 17](#_Toc153735420)

[**3.2 Контроль входных символов** 17](#_Toc153735421)

[**3.3 Удаление избыточных символов** 18](#_Toc153735422)

[**3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов** 18](#_Toc153735423)

[**3.5 Основные структуры данных** 19](#_Toc153735424)

[**3.6 Принцип обработки ошибок** 20](#_Toc153735425)

[**3.7 Структура и перечень сообщений лексического анализатора** 20](#_Toc153735426)

[**3.8 Параметры лексического анализатора и режимы его работы** 20](#_Toc153735427)

[**3.9 Алгоритм лексического анализа** 21](#_Toc153735428)

[**3.10 Контрольный пример** 21](#_Toc153735429)

[**4. Разработка синтаксического анализатора** 22](#_Toc153735430)

[**4.1 Структура синтаксического анализатора** 22](#_Toc153735431)

[**4.2 Контекстно свободная грамматика, описывающая синтаксис языка** 22](#_Toc153735432)

[**4.3 Построение конечного магазинного автомата** 23](#_Toc153735433)

[**4.4 Основные структуры данных** 23](#_Toc153735434)

[**4.5 Описание алгоритма синтаксического разбора** 24](#_Toc153735435)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 24](#_Toc153735436)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 24](#_Toc153735437)

[**4.8 Принцип обработки ошибок** 24](#_Toc153735438)

[**4.9 Контрольный пример** 25](#_Toc153735439)

[**5. Разработка семантического анализатора** 26](#_Toc153735440)

[**5.1 Структура семантического анализатора** 26](#_Toc153735441)

[**5.2 Функции семантического анализатора** 26](#_Toc153735442)

[**5.3 Структура и перечень сообщений семантического анализатора** 26](#_Toc153735443)

[**5.4 Принцип обработки ошибок** 26](#_Toc153735444)

[**5.5 Контрольный пример** 27](#_Toc153735445)

[**6. Преобразование выражений** 28](#_Toc153735446)

[**6.1 Выражения, допускаемые языком** 28](#_Toc153735447)

[**6.2 Польская запись** 28](#_Toc153735448)

[**6.3 Программная реализация обработки выражений** 29](#_Toc153735449)

[**6.4 Контрольный пример** 29](#_Toc153735450)

[**7. Генерация кода** 30](#_Toc153735451)

[**7.1 Структура генератора кода** 30](#_Toc153735452)

[**7.2 Представление типов данных в оперативной памяти** 30](#_Toc153735453)

[**7.3 Статическая библиотека** 31](#_Toc153735454)

[**8.1 Тестирование фазы проверки на допустимость символов** 32](#_Toc153735455)

[**8.2** **Тестирование лексического анализатора** 32](#_Toc153735456)

[**8.3 Тестирование синтаксического анализатора** 32](#_Toc153735457)

[**8.4 Тестирование семантического анализатора** 32](#_Toc153735458)

[**Заключение** 33](#_Toc153735459)

[**Литература** 34](#_Toc153735460)

[**Приложение А** 35](#_Toc153735461)

[**Приложение Б** 38](#_Toc153735462)

[**Приложение В** 41](#_Toc153735463)

[**Приложение Г** 46](#_Toc153735464)

[**Приложение Д** 49](#_Toc153735465)

[**Приложение Е** 51](#_Toc153735466)

[**Приложение Ж** 56](#_Toc153735467)

[**Приложение З** 60](#_Toc153735468)

# **Введение**

Задачей данного курсового проекта является разработка транслятора для своего языка программирования: GDD-2023. Этот язык программирования предназначен для выполнения простейших операций и арифметических действий над строками, символами и числами.

Главная задача транслятора заключается в том, чтобы сделать программу, написанную языке программирования GDD-2023, понятной компьютеру. В данном курсовом проекте мой исходный код транслируется на язык ассемблера. Язык ассемблера – тип языка программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком.

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;

– обработка выражений;

– генерация кода на язык Assembler;

– тестирование транслятора.

# **1. Спецификация языка программирования**

## **Характеристика языка программирования**

Язык программирования GDD-2023 – это универсальный язык высокого уровня. Он является процедурным, компилируемым, не объектно-ориентированным. Язык строго типизируемый, что говорит о невозможности преобразования типов, транслируемым языком программирования.

* 1. **Алфавит языка**

Алфавит языка GDD-2023 основывается на таблице Windows-1251, представленной на рисунке 1.1.

Буква –> {a | b | … | z | A | B | … | Z}

Цифра –> {0 | 1 | … | 9}

Специальный символ -> {[ | ] | ( | ) | { | } | , | ” | ; | $ | ? | пробел | < | > | ~ | # | = | + | – | \* | / | %}

Символ языка –> {Буква | Цифра | Специальный символ}

## **Символы сепараторы**

Сепараторы необходимы для разделения операций языка. Сепараторы, используемые в языке программирования GDD-2023, приведены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| [] | используются для записи параметров, условия для условных блоков, возвращаемого значения, вывода |
| {} | определяют границы тела функции и условных блоков |
| () | увеличивают приоритет выражения |
| пробел | разделяет лексемы |
| , | разделяет параметры функции |
| < > ~ # $ ? | разделяют операнды в условном выражении |
| ; | разделяет инструкции |
| “ | определяет границы строковых литералов |
| ‘ | определяет границы символьных литералов |
| = | разделяет присваиваемое выражение и переменную |
| + – / \* % | разделяют операнды в арифметических выражениях |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования GDD-2023 используется кодировка Windows-1251, представленной на рисунке 1.1.

Изображение выглядит как текст, снимок экрана, число, Прямоугольник

Автоматически созданное описание

Рисунок 1.1 – Алфавит входных символов

## **Типы данных**

В GDD-2023 используется четыре типа данных: целочисленные (int\_), символьный(chr\_), строковые (str\_), логический (bool\_). Пользовательские типы данных не поддерживаются. Допускается использование фундаментальных типов данных, определенных в таблице 1.2.

Таблица 1.2 – Типы данных языка GDD-2023

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| int\_ | Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных. Автоматически инициализируется нулевым значением. |
| chr\_ | Фундаментальный тип данных. Предусмотрен для объявления символов. Автоматическая инициализация нулевым значением. |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| str\_ | Фундаментальный тип данных. Предусмотрен для объявления строк. Автоматическая инициализация нулевым значением. Максимальное количество символов в строке – 255. |
| bool­\_ | Фундаментальный тип данных. Предусмотрен для объявления логической переменной, которая имеет одно из двух значений: true, false. Автоматическая инициализация значением false. |

## **Преобразование типов данных**

В языке программирования GDD-2023 преобразование типов данных не поддерживается.

## **Идентификаторы**

Для именования функций, параметров и переменных используются идентификаторы. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Идентификаторы не должны совпадать с ключевыми словами. Не предусмотрены зарезервированные идентификаторы. Имя идентификатора составляется по следующим образом:

* состоит из символов латинского алфавита нижнего регистра [a..z].
* максимальная длина идентификатора равна 10 и не должна превышать это значение.

## **Литералы**

В языке существует 3 типа литералов: целого, символьного и строкового типов. Краткое описание литералов представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные литералы десятичного, двоичного и восьмеричного представления, автоматически инициализируются 0. |
| Символьные литералы | Символ, заключённые в “” (двойные кавычки), автоматически инициализируются NULL. |
| Строковые литералы | Символы, заключённые в “” (двойные кавычки), автоматически инициализируются пустой строкой, строковые переменные. Максимальное число символов в литерале . |

## **Область видимости идентификаторов**

Область видимости «сверху вниз» (по принципу С++). В языке GDD-2023 требуется обязательное объявление переменной перед её использованием. Все переменные должны находиться внутри программного блока языка. Имеется возможность объявления одинаковых переменных в разных блоках.

## **Инициализация данных**

Таблица 1.4 – Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| create <тип данных> <идентификатор>; | Автоматическая инициализация: переменные типа int\_ инициализируются нулём, переменные типа chr\_ и str\_ – пустой строкой, bool\_ – true/false; |
| <идентификатор> = <значение>; | Присваивание переменной значения. |

## **Инструкции языка**

Все возможные инструкции языка программирования GDD-2023 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования GDD-2023

|  |  |
| --- | --- |
| Инструкция | Запись на языке GDD-2023 |
| Объявление переменной | create <тип данных> <идентификатор>; |
| Присваивание | <идентификатор> = <значение>|<идентификатор>; |
| Объявление внешней функции | function <тип данных> <идентификатор> [<тип данных> <идентификатор>, …] {…} |
| Блок инструкций | {  …  } |
| Возврат из подпрограммы | give <идентификатор> / <литерал>; |
| Вывод данных | write [<идентификатор> / <литерал>]; |
| Условный оператор | when [<условие>]  { … }  or  { … } |
| Оператор цикла | cycle [<условие>]  { … } |

## **Операции языка**

Язык программирования GDD-2023 может выполнять арифметические операции, представленные в таблице 1.6.

Таблица 1.6 – Приоритетности операций языка программирования GDD-2023

|  |  |
| --- | --- |
| Операция | Приоритетность операции |
| (  ) | 0 |
| , | 1 |
| \*  / (деление нацело)  % (деление с остатком) | 3 |
| +  - | 2 |

Максимальным значением приоритетности является “3”, минимальным “0” соответственно. Также языком поддерживается операция сравнения операндов. Данная операция не имеет приоритета.

## **Выражения и их вычисления**

Круглые скобки в выражении используются для изменения приоритета операций. Также не допускается запись двух подряд идущих арифметических операций. Выражение может содержать вызов функции. Фигурные скобки используются для составления блоков кода функций, операторов. Квадратные – для передачи параметров функций.

## **Программные конструкции языка**

Ключевые программные конструкции языка программирования GDD-2023 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка GDD-2023

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

## **Область видимости**

В языке GDD-2023 переменные обязаны находится внутри программного блока функций. Все объявления и операции с переменными происходят внутри какого-либо блока. Каждая переменная или параметр функции получают префикс – название функции, внутри которой они находятся. Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

## **Семантические проверки**

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

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы функций не должны повторяться |
| 2 | Операнды в операторах ветвления и выхода из функции должны быть целочисленного типа |
| 3 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении |
| 4 | Тип данных передаваемых значений в функцию стандартной библиотеки должен соответствовать заявленному. |
| 5 | Идентификатор должен быть объявлен до его использования. |
| 6 | Операнды в арифметическом выражении не могут быть разных типов |

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

Все переменные размещаются в стеке. Таблица лексем и таблица идентификаторов сохраняются в структуры с выделенной под них динамической памятью, которая очищается по окончанию работы транслятора.

## **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 |
| combine | str\_ | str\_ a, str\_ b | Функция производит конкатенацию строк a и b, возвращает строку |
| lexcomp | int\_ | str\_ a, str\_ b | Функция производит сравнение числовые коды символов в строках. Возвращает значение равное 0, если |

Продолжение таблицы 1.9

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | числовые коды строк равны, больше нуля, если числовой код a больше b, меньше нуля в противном случае. |
| measure | int\_ | str\_ a | Функция вычисляет длину строки a |
| randomize | int\_ | int\_ min, int\_ max | Функция возвращает случайно сгенерированное число в промежутке [min, max] |
| exponent | int\_ | int\_ a, int\_ n | Функция возводит число a в степень n, возвращает результат |
| OutputStr | - | str\_ a | Функция выводит в консоль строку a |
| OutputInt | - | int\_ a | Функция выводит в консоль число a |

## **Ввод и вывод данных**

Ввод данных не поддерживается языком программирования GDD-2023. Вывод в стандартный поток вывода: speak [<идентификатор или литерал>];

В зависимости от типа параметра определяется функция: OutputStr или OutputInt, которые входят в состав стандартной библиотеки и описаны в таблице 1.9.

## **Точка входа**

Точкой входа является функция main.

## **Препроцессор**

Препроцессор в языке программирования GDD-2023 не предусмотрен.

## **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.

## **Объектный код**

GDD-2023 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке GDD-2023 и выявления её транслятором в текущий файл протокола выводится сообщение. Их классификация сообщений приведена в таблице 1.10.

Таблица 1.10. – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-110 | Системные ошибки, ошибки параметров |
| 200-220 | Ошибки лексического анализа |
| 600-620 | Ошибки синтаксического анализа |
| 300-400 | Ошибки семантического анализа |

## **Контрольный пример**

Контрольный пример представлен в Приложении А.

# **2. Структура транслятора**

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

Транслятор преобразует программу, написанную на языке GDD-2023 в программу на языке ассемблера. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.



Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка и формирование таблицы лексем и таблицы идентификаторов.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций. Входным параметром для синтаксического анализа является таблица лексем. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора.

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

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке GDD-2023 в код на языке Ассемблера.

## **2.2 Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка GDD-2023

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с расширением .txt, в котором содержится исходный код на GDD-2023 | Не предусмотрено |
| -log:<имя\_файла> | Файл для записи полного 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 | По умолчанию отсутствует |
| -lex | Ключ для вывода таблицы лексем в консоль | По умолчанию отсутствует |
| -id | Ключ для вывода трассировки синтаксического анализа в файл | По умолчанию отсутствует |

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

Таблица с перечнем протоколов, формируемых транслятором языка

GDD-2023 и их назначением представлена в таблице 2.2

Таблица 2.2 – Протоколы, формируемые транслятором языка GDD-2023

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала с параметром <log> | Содержит информацию о времени выполнения приложения; входных параметрах в приложение; код на языке GDD-2023 с сепараторами и без избыточных пробелов, табуляций и переходов на новую строку; таблицу идентификаторов; таблицу лексем; промежуточное представление кода; трассировку синтаксического анализа; дерево разбора, время выполнения разбора; промежуточное представление кода после приведения его к польской нотации. |
| Выходной файл c параметром <out> | Содержит сгенерированный код на языке Ассемблера. |

# **3. Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке GDD-2023. На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 – Структура лексического анализатора GDD-2023

## **3.2 Контроль входных символов**

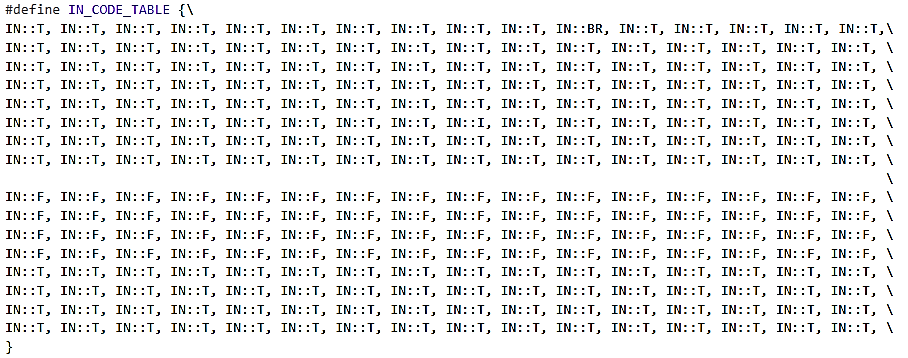
Таблица для контроля входных символов представлена на рисунке 3.2

Рисунок 3.2. – Таблица контроля входных символов

Принцип работы таблицы заключается в соответствии значения каждому элементу в шестнадцатеричной системе счисления значению в таблице ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, I – игнорируемый символ, BR – символ перехода на новую строку (\n).

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на лексемы.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем исходный код, занесенный в структуру In.

2. Встреча пробела или знака табуляции вне пределов строкового литерала является своего рода встречей символа-сепаратора.

3. В отличие от других символов-сепараторов не записываем в таблицу лексем эти символы, т.е. игнорируем.

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

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

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

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | create | c |
| int\_, chr\_, str\_, bool\_ | t |
| main | m |
| function | f |
| procedure | p |
| give | r |
| write | s |
| cycle | u |
| when | w |
| or | ! |
| Иное | Идентификатор | i |
| Литерал | l |
| Функции стандартной библиотеки | combine | k |
| lexcomp | s |
| measure | d |
| exponent | e |
| breakl | b |
| randomize | z |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| [ | [ |
| ] | ] |
| Операторы | + | + |
| - | - |
| \* | \* |
| / | / |
| % | % |
| Логические (~ # > <) | g |
| = | = |

Пример реализации таблицы лексем представлен в Приложении Б.

Также в Приложении B находятся конечные автоматы, соответствующие лексемам языка GDD-2023.

## **3.5 Основные структуры данных**

Описание основных структур данных, используемых для хранения таблиц идентификаторов, представлено на рис. 3.3.

|  |
| --- |
|  |

Рисунок 3.3. — Структуры таблиц идентификаторов GDD-2023

Описание основных структур данных, используемых для хранения таблиц лексем, представлено на рис. 3.4.

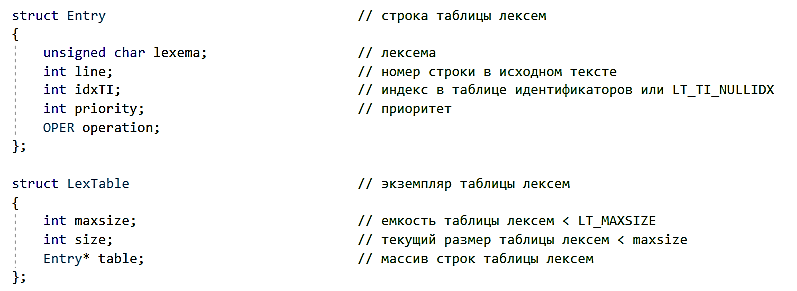


Рисунок 3.4 — Структуры таблиц лексем GDD-2023

## **3.6 Принцип обработки ошибок**

При возникновении ошибки типа предупреждение транслятор продолжает свою работу, а предупреждения записываются в специальную структуру с номером ошибки и диагностическим сообщением.

Когда возникает критическая ошибка – работа транслятора прекращается.

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

Перечень сообщений лексического анализатора представлен на рисунке 3.5.

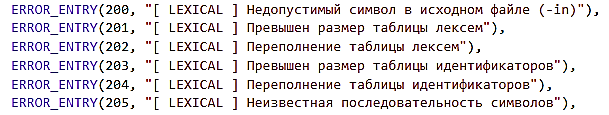


Рисунок 3.5 – Перечень ошибок лексического анализатора

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

Входным параметром лексического анализатора является исходный текст программы, написанный на языке GDD-2023, а также файл протокола.

## **3.9 Алгоритм лексического анализа**

Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором. Цель лексического анализа — выделение и классификация лексем в тексте исходной программы. Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. Регулярные выражения — аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа.

Пример. Регулярное выражение для ключевого слова main.

Граф конечного автомата для этой лексемы представлен на рисунке 3.6. S0 – начальное состояние, S4 – конечное состояние автомата. В виде кода представлен на рисунке 3.7.

|  |
| --- |
| n  m  a  i |

Рисунок 3.6 — Граф переходов для цепочки “main”

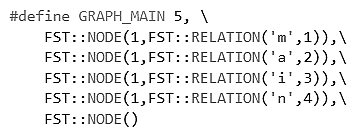


Рисунок 3.7 — Граф переходов для цепочки “main”

## **3.10 Контрольный пример**

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

# **4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций. Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа. Выходом – дерево разбора. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора

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

В синтаксическом анализаторе транслятора языка GDD-2023 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 GDD-2023 представлена 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.

Перечень правил, составляющих грамматику языка и описание нетерминальных символов GDD-2023 показаны в приложении Г.

## **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 GDD-2023. Данные структуры представлены в Приложении Д.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

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

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

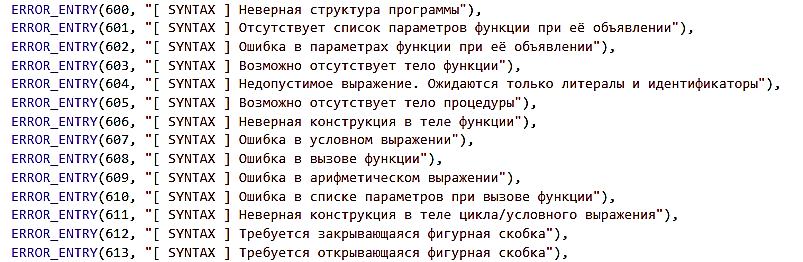
Перечень сообщений синтаксического анализатора представлен на рисунке 4.1.

Рисунок 4.1 – Перечень сообщений синтаксического анализатора

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

Входным параметром синтаксического анализатора является таблица лексем, полученная на этапе лексического анализа, 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.

# **5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ в трансляторе языка GDD-2023 выделен в отдельную фазу, и реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях: установки флага или нахождении в особом месте программы (оператор выхода из функции, оператор ветвления, вызов функции стандартной библиотеки и т.д.).

## **5.2 Функции семантического анализатора**

За семантический анализ отвечает функция Analyze. Ее входными параметрами является таблица лексем и поток вывода в протокол.

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

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

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.1.

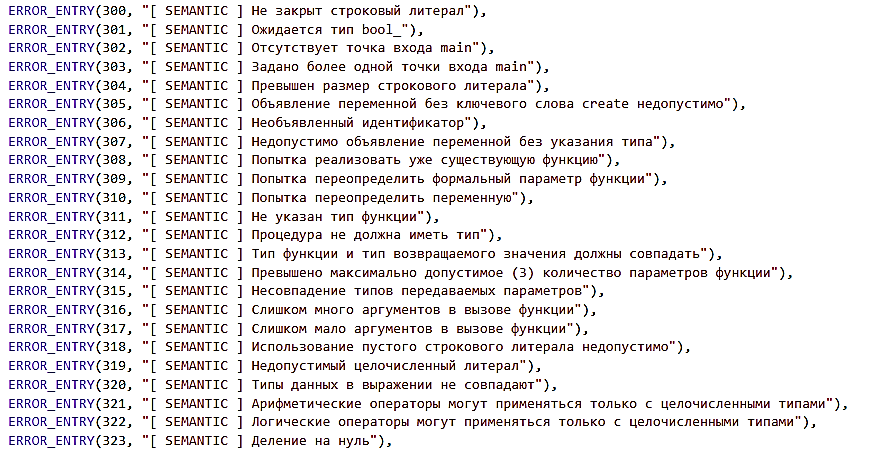


Рисунок 5.1 – Перечень сообщений семантического анализатора

## **5.4 Принцип обработки ошибок**

Принцип обработки ошибок идентичен принципу обработки ошибок на этапе лексического анализа (раздел 3.6).

## **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 |
| {  create int\_ a = 5;  write [a]; breakl;  } | ERROR CODE 302: [ SEMANTIC ] Отсутствует точка входа main  Строка -1 позиция -1 |
| main  {  create int\_ a = 5;  write [a]; breakl;  write ["Bye!];  } | ERROR CODE 300: [ SEMANTIC ] Не закрыт строковый литерал  Строка -1 позиция -1 |
| main  {  create a = 5;  write [a]; breakl;  write ["Bye!"];  } | ERROR CODE 307: [ SEMANTIC ] Недопустимо объявление переменной без указания типа  Строка 2 позиция 0 |

# **6. Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке GDD-2023 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, /, %(остаток от деления) и (), и вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке GDD-2023

|  |  |
| --- | --- |
| Приоритет | Операция |
| 0 | ( ) |
| 1 | , |
| 2 | + - |
| 3 | \* / % |
| 4 | [ ] – скобки параметров функции |

## **6.2 Польская запись**

Выражения в языке GDD-2023 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 операторы 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 |
| (l+l)\*l/l |  |  |
| l+l)\*l/l |  |  |
| +l)\*l/l | l |  |
| l)\*l/l | l | + |
| )\*l/l | ll | + |
| \*l/l | ll+ |  |
| l/l | ll+ | \* |
| /l | ll+l | \* |
| l | ll+l\* | / |
|  | ll+l\*l | / |
|  | ll+l\*l/ |  |

## **6.3 Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений в обратный польский формат основана функциях Poliz и StartPoliz. Функция StartPoliz принимает как параметр адрес таблицы лексем и содержит цикл, в ходе которого перебираются все лексемы исходного кода. Если последовательность лексем соответствует началу выражения, вызывается функция Poliz, где и проводится точечное преобразование выражений к польской нотации.

## **6.4 Контрольный пример**

Пример преобразования выражения к польской записи представлен в таблице 6.4. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления.

В Приложении Ж приведен измененное представление промежуточного кода, отображающее результаты преобразования выражений в польский формат.

# **7. Генерация кода**

## **7.1 Структура генератора кода**

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



Рисунок 7.1 Структура генератора кода

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

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка GDD-2023 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке GDD-2023 и на языке ассемблера приведены в таблице 7.1. Сгенерированный код предоставлен в Приложении З.

Таблица 7.1 – Соответствия типов идентификаторов языка GDD-2023 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке GDD-2023 | Тип идентификатора на языке ассемблера | Пояснение |
| int\_ | SDWORD | Хранит целочисленный тип данных со знаком. |
| chr\_ | DWORD | Хранит целочисленный тип данных без знака, размером 1 байт. |
| str\_ | DWORD | Каждый символ строки хранится размером в 1 байт. |
| bool\_ | SDWORD | Хранит целочисленный тип данных без знака. |

## **7.3 Статическая библиотека**

Функции из стандартной библиотеки содержатся в проекте LIB, в свойствах которого указан тип конфигурации «статическая библиотека». Подключение библиотеки происходит с помощью includelib на этапе генерации кода путем вывода в поток out. Таким же образом c помощью оператора EXTRN объявляются названия 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.

**8. Тестирование транслятора**

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

В языке GDD-2023 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| Fu№ction int\_ aver [int\_ n, int\_ m] | ERROR CODE 200: [ LEXICAL ] Недопустимый символ в исходном файле (-in)  Строка 1 позиция 4 |

## **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 |
| function intfhgff\_ aver [int\_ n, int\_ m] | ERROR CODE 205: [ LEXICAL ] Неизвестная последовательность символов  Строка 0 позиция 9 |

## **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 |
| main  {  create int\_ hey = 5;  write [hey]; breakl; | ERROR CODE 612: [ SYNTAX ] Требуется закрывающаяся фигурная скобка  Строка -1 позиция -1 |

## **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора приведены в пункте 5.5.

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования GDD-2023 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка GDD-2023;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для

эффективной работы лексического анализатора;

1. Осуществлена программная реализация лексического анализатора,

распознающего допустимые цепочки спроектированного языка;

1. Разработана контекстно-свободная, приведённая к нормальной форме

Грейбах, грамматика для описания синтаксически верных конструкций языка;

1. Осуществлена программная реализация синтаксического анализатора;
2. Разработан семантический анализатор, осуществляющий проверку

используемых инструкций на соответствие логическим правилам;

1. Разработан транслятор кода на язык ассемблера;
2. Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка GDD-2023 включает:

1. 4 типа данных;
2. Поддержка оператора вывода строки;
3. Наличие 5 арифметических операторов для вычисления выражений
4. Наличие 6 логических операторов для использования в условиях цикла и

условной конструкции

1. Поддержка функций; Операторов цикла и условия;
2. Наличие библиотеки стандартных функций языка
3. Структурированная и классифицированная система для обработки ошибок

пользователя.

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

# **Литература**

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**

|  |
| --- |
| function int\_ func [int\_ a, int\_ b]  {  create int\_ ress = 0;  ress = (a + b)\* 2 /2;  ress = ress \* 2;  give [ress];  }  function int\_ funcc []  {  create int\_ test = 3;  write[test]; breakl;  give[test];  }  procedure cyclen [int\_ n]  {  create int\_ i = 0;  write ["Counting to length: "];  write [n]; breakl;  cycle [i < n]  {  i = i + 1;  write [i]; write [" "];  }  breakl;  }  main  {  create str\_ start = combine ["Test Course Project ", "GDD2023 "];  create str\_ startt = combine [start, "Start in 3 2 1"];  write [startt]; breakl;  write ["Numbers: "]; breakl;  create int\_ bin = 1b10;  write [bin]; breakl;  create int\_ oct = 0o10;  write [oct]; breakl;  write ["bin - oct = "];  create int\_ dif = bin - oct;  write [dif]; breakl;  create bool\_ booly = true;  write["Bool: "];  write[booly]; breakl;  create chr\_ char = "A";  write [char]; breakl;  create int\_ exp = exponent [2, 4];  write ["exp[2,4] = "];  write [exp]; breakl;  when[exp ~ 16]  {  write["pow 2,4 really equal 16"]; breakl;  }  or  {  write["pow 2,4 not really equal 16"]; breakl;  }  create int\_ length = measure ["Now"];  write ["length: "];  write [length]; breakl;  create int\_ randd = randomize [3, 40];  write ["Random: "];  write [randd]; breakl;  create int\_ lexc = lexcomp ["Gddddd", "Gddd"];  write ["Lexcomp: "];  write [lexc]; breakl;  create int\_ testx = func[1,5];  write [testx]; breakl;  create int\_ testy = funcc[];  write [testy]; breakl;  create int\_ t = 4;  cyclen[t];  when[t $ 10]  {  write ["End of example if <= 10"]; breakl;  }  or  {  write ["End of example if > 10"]; breakl;  }  } |

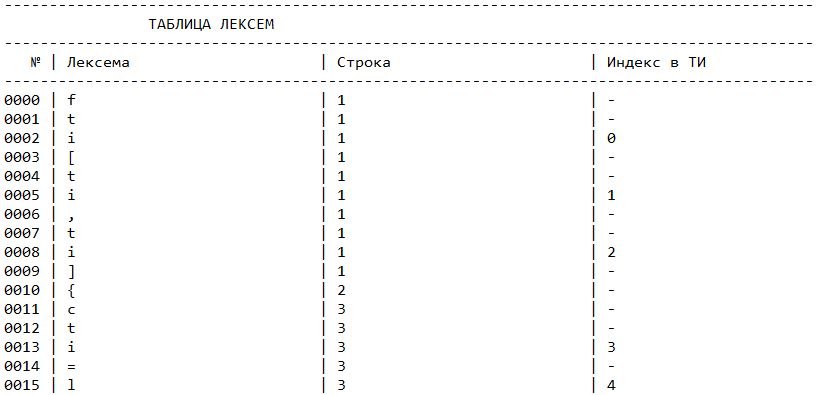
## **Приложение Б**

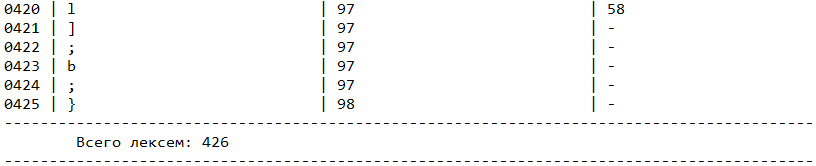
|  |
| --- |
|  |

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Начало таблицы лексем:

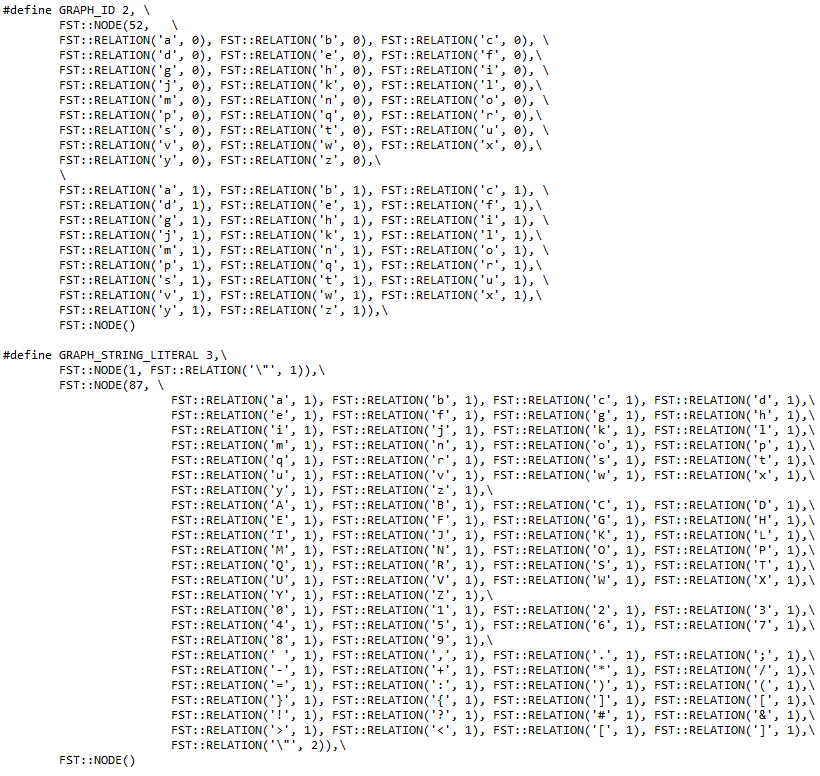
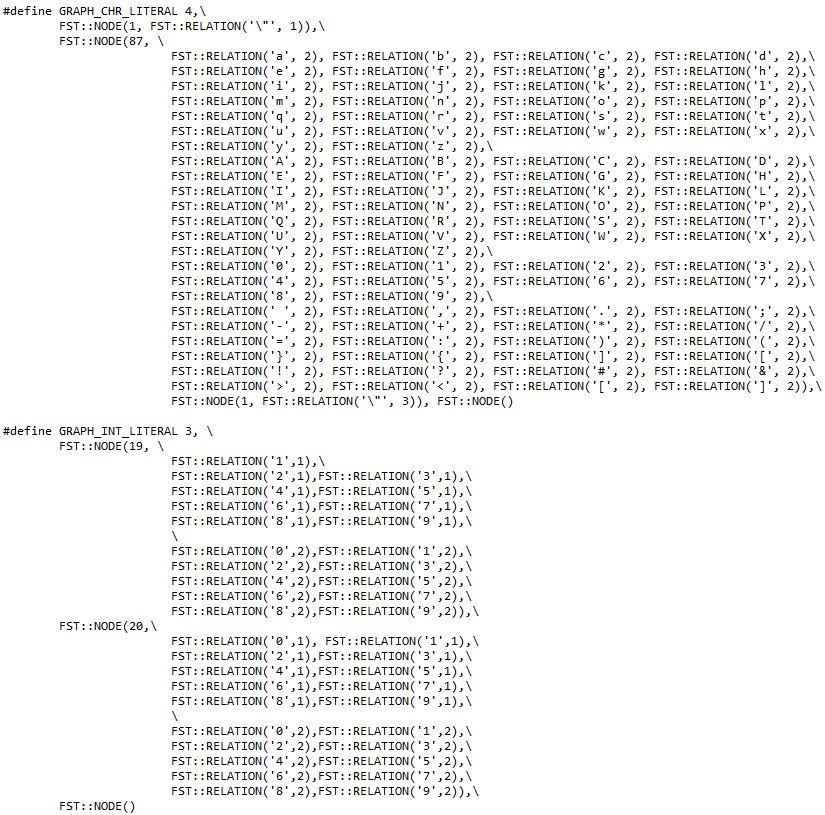
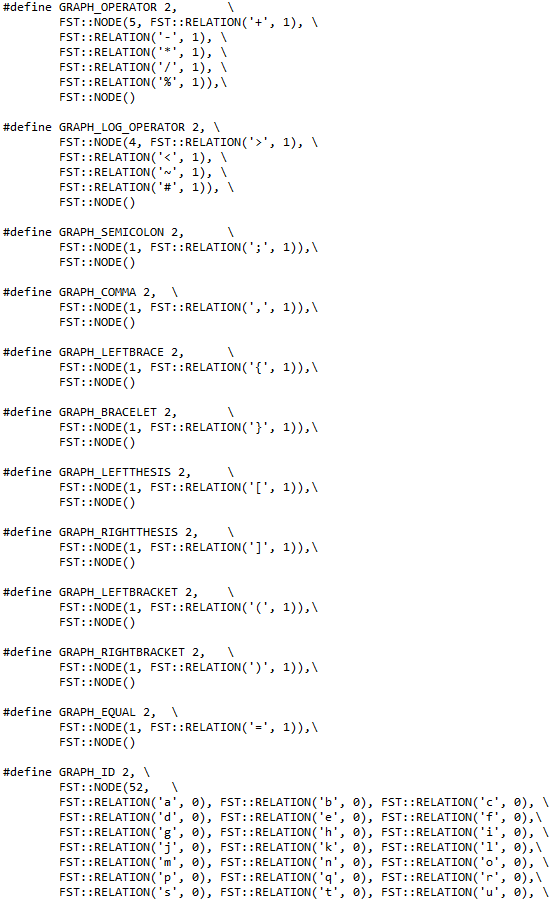


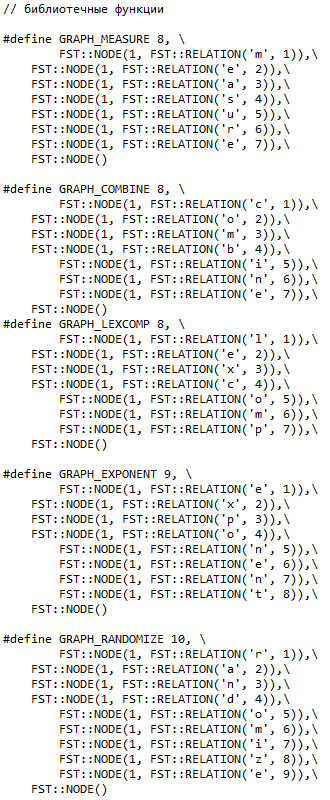
Окончание таблицы лексем: 

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

## **Приложение В**

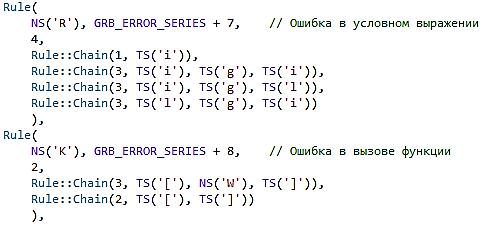
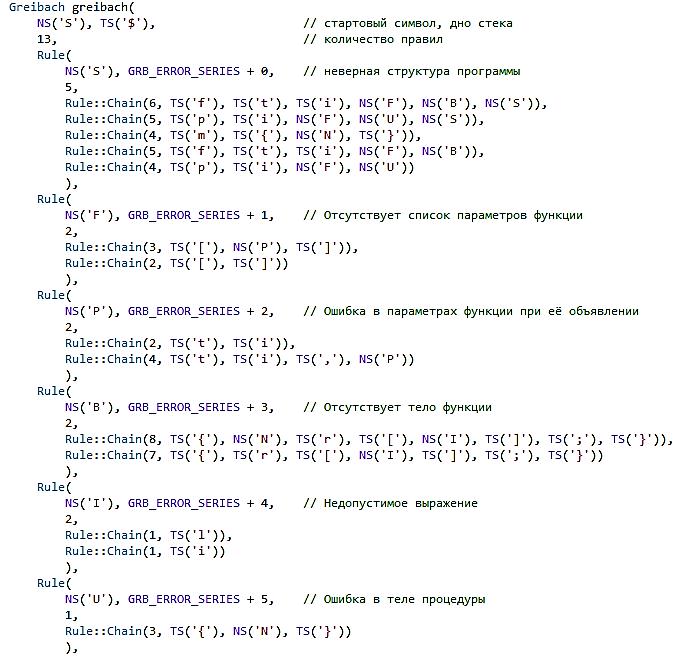


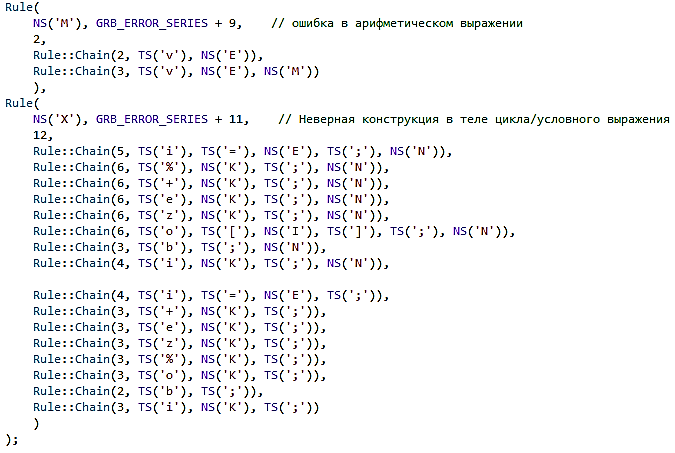
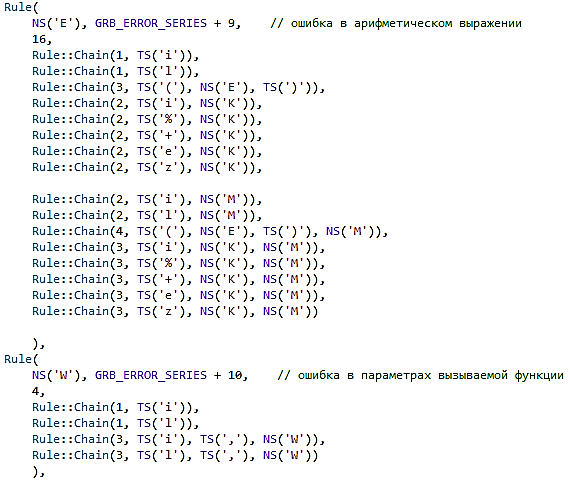
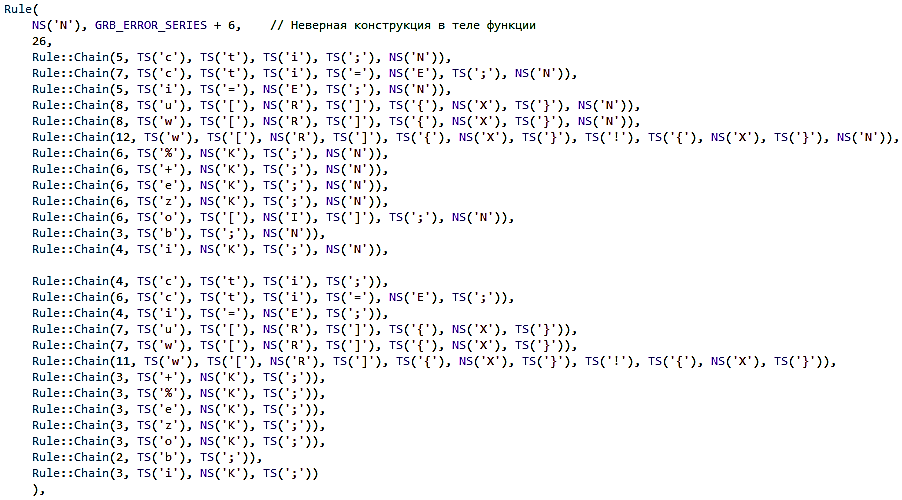


Изображение выглядит как текст, снимок экрана, число

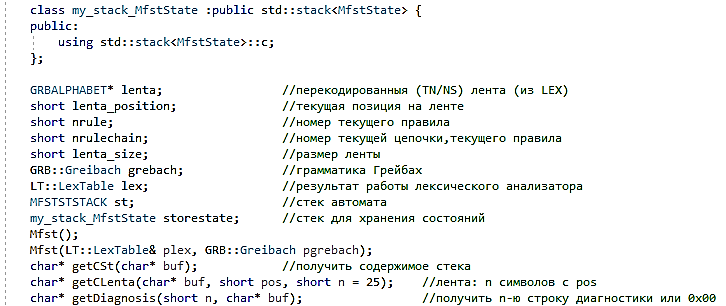
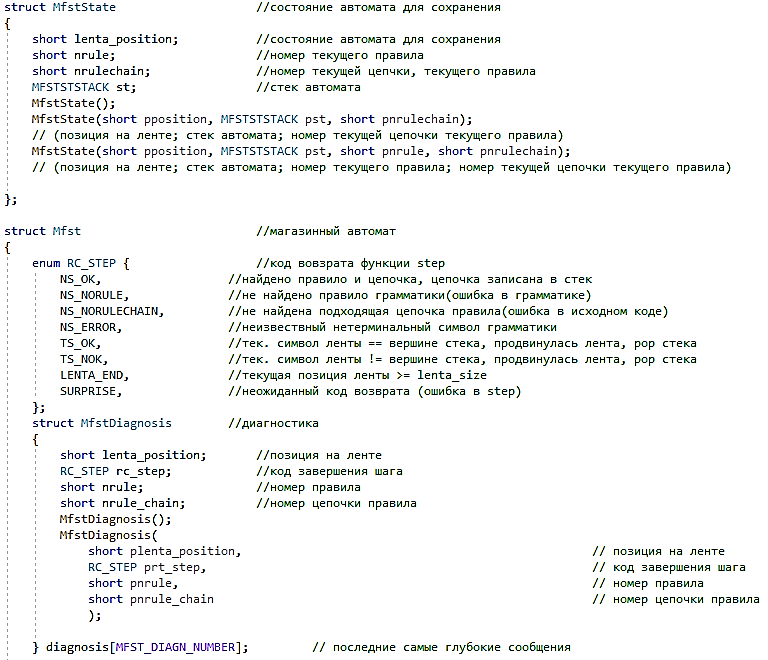
Автоматически созданное описание

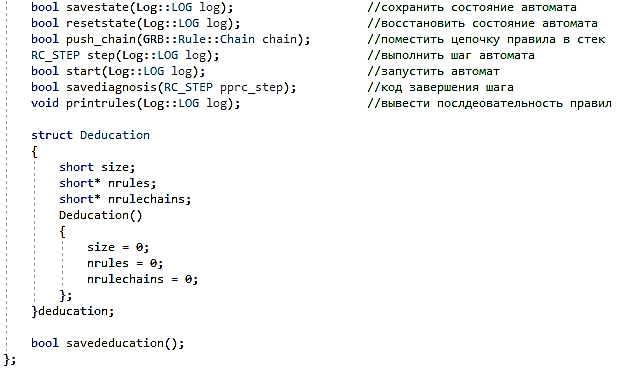
## **Приложение Г**





## **Приложение Д**





## **Приложение Е**

Начало разбора

|  |
| --- |
| -------------------------------------------------------------------  ТРАССИРОВКА СИНТАКСИЧЕСКОГО АНАЛИЗА  -------------------------------------------------------------------  Шаг-: Правило------------ Входная лента---------------- Стек-------  0---: S->ftiFBS-----------------------------------------S$---------  0---: SAVESTATE:----------1  0---: -------------------------------------------------ftiFBS$----  1---: -------------------ti[ti,ti]{cti=l;i=ivi;i=i-----tiFBS$-----  2---: -------------------i[ti,ti]{cti=l;i=ivi;i=iv-----iFBS$------  3---: -------------------[ti,ti]{cti=l;i=ivi;i=ivl-----FBS$-------  4---: F->[P]--------------[ti,ti]{cti=l;i=ivi;i=ivl-----FBS$-------  4---: SAVESTATE:----------2  4---: -------------------[ti,ti]{cti=l;i=ivi;i=ivl-----[P]BS$-----  5---: -------------------ti,ti]{cti=l;i=ivi;i=ivl;-----P]BS$------  6---: P->ti---------------ti,ti]{cti=l;i=ivi;i=ivl;-----P]BS$------  6---: SAVESTATE:----------3  6---: -------------------ti,ti]{cti=l;i=ivi;i=ivl;-----ti]BS$-----  7---: -------------------i,ti]{cti=l;i=ivi;i=ivl;r-----i]BS$------  8---: -------------------,ti]{cti=l;i=ivi;i=ivl;r[-----]BS$-------  9---: 2-------------------  9---: RESSTATE------------  9---: -------------------ti,ti]{cti=l;i=ivi;i=ivl;-----P]BS$------  10--: P->ti,P-------------ti,ti]{cti=l;i=ivi;i=ivl;-----P]BS$------  10--: SAVESTATE:----------3 |

Конец разбора

|  |  |  |
| --- | --- | --- |
| |  | | --- | | 1141: K->[W]--------------[l];}}}}}}}}}}}}}}}}}}}}}-----K;}$-------  1141: SAVESTATE:----------140  1141: -------------------[l];}}}}}}}}}}}}}}}}}}}}}-----[W];}$-----  1142: -------------------l];}}}}}}}}}}}}}}}}}}}}}}-----W];}$------  1143: W->l----------------l];}}}}}}}}}}}}}}}}}}}}}}-----W];}$------  1143: SAVESTATE:----------141  1143: -------------------l];}}}}}}}}}}}}}}}}}}}}}}-----l];}$------  1144: -------------------];}}}}}}}}}}}}}}}}}}}}}}}-----];}$-------  1145: -------------------;}}}}}}}}}}}}}}}}}}}}}}}}-----;}$--------  1146: -------------------}}}}}}}}}}}}}}}}}}}}}}}}}-----}$---------  1147: -------------------}}}}}}}}}}}}}}}}}}}}}}}}}-----$----------  1148: 6-------------------  1149: ------>LENTA\_END----  -------------------------------------------------------------------  Синтаксический анализ выполнен без ошибок.  ------------------------------------------------------------------- | | ------------------------------------------------------------------------------------------  Дерево разбора:  ------------------------------------------------------------------------------------------  0---: S->ftiFBS-----------  3---: F->[P]--------------  4---: P->ti,P-------------  7---: P->ti---------------  10--: B->{Nr[I];}---------  11--: N->cti=E;N----------  15--: E->l----------------  17--: N->i=E;N------------  19--: E->(E)M-------------  20--: E->iM---------------  21--: M->+E---------------  22--: E->i----------------  24--: M->\*E---------------  25--: E->lM---------------  26--: M->/E---------------  27--: E->l----------------  29--: N->i=E;-------------  31--: E->iM---------------  32--: M->\*E---------------  33--: E->l----------------  37--: I->i----------------  41--: S->ftiFBS-----------  44--: F->[]---------------  46--: B->{Nr[I];}---------  47--: N->cti=E;N----------  51--: E->l----------------  53--: N->o[I];N-----------  55--: I->i----------------  58--: N->b;---------------  62--: I->i----------------  66--: S->piFUS------------  68--: F->[P]--------------  69--: P->ti---------------  72--: U->{N}--------------  73--: N->cti=E;N----------  77--: E->l----------------  79--: N->o[I];N-----------  81--: I->l----------------  84--: N->o[I];N-----------  86--: I->i----------------  89--: N->b;N--------------  91--: N->u[R]{X}N---------  93--: R->igi--------------  98--: X->i=E;N------------  100-: E->iM---------------  101-: M->+E---------------  102-: E->l----------------  104-: N->o[I];N-----------  106-: I->i----------------  109-: N->oK;--------------  110-: K->[W]--------------  111-: W->l----------------  115-: N->b;---------------  118-: S->m{N}-------------  120-: N->cti=E;N----------  124-: E->kK---------------  125-: K->[W]--------------  126-: W->l,W--------------  128-: W->l----------------  131-: N->cti=E;N----------  135-: E->kK---------------  136-: K->[W]--------------  137-: W->i,W--------------  139-: W->l----------------  142-: N->o[I];N-----------  144-: I->i----------------  147-: N->b;N--------------  149-: N->o[I];N-----------  151-: I->l----------------  154-: N->b;N--------------  156-: N->cti=E;N----------  160-: E->l----------------  162-: N->o[I];N-----------  164-: I->i----------------  167-: N->b;N--------------  169-: N->cti=E;N----------  173-: E->l----------------  175-: N->o[I];N-----------  177-: I->i----------------  180-: N->b;N--------------  182-: N->o[I];N-----------  184-: I->l----------------  187-: N->cti=E;N----------  191-: E->iM---------------  192-: M->-E---------------  193-: E->i----------------  195-: N->o[I];N-----------  197-: I->i----------------  200-: N->b;N--------------  202-: N->cti=E;N----------  206-: E->l----------------  208-: N->o[I];N-----------  210-: I->l----------------  213-: N->o[I];N-----------  215-: I->i----------------  218-: N->b;N--------------  220-: N->cti=E;N----------  224-: E->l----------------  226-: N->o[I];N-----------  228-: I->i----------------  231-: N->b;N--------------  233-: N->cti=E;N----------  237-: E->eK---------------  238-: K->[W]--------------  239-: W->l,W--------------  241-: W->l----------------  244-: N->o[I];N-----------  246-: I->l----------------  249-: N->o[I];N-----------  251-: I->i----------------  254-: N->b;N--------------  256-: N->w[R]{X}!{X}N-----  258-: R->igl--------------  263-: X->o[I];N-----------  265-: I->l----------------  268-: N->b;---------------  273-: X->o[I];N-----------  275-: I->l----------------  278-: N->b;---------------  281-: N->cti=E;N----------  285-: E->dK---------------  286-: K->[W]--------------  287-: W->l----------------  290-: N->o[I];N-----------  292-: I->l----------------  295-: N->o[I];N-----------  297-: I->i----------------  300-: N->b;N--------------  302-: N->cti=E;N----------  306-: E->zK---------------  307-: K->[W]--------------  308-: W->l,W--------------  310-: W->l----------------  313-: N->o[I];N-----------  315-: I->l----------------  318-: N->o[I];N-----------  320-: I->i----------------  323-: N->b;N--------------  325-: N->cti=E;N----------  329-: E->sK---------------  330-: K->[W]--------------  331-: W->l,W--------------  333-: W->l----------------  336-: N->o[I];N-----------  338-: I->l----------------  341-: N->o[I];N-----------  343-: I->i----------------  346-: N->b;N--------------  348-: N->cti=E;N----------  352-: E->iK---------------  353-: K->[W]--------------  354-: W->l,W--------------  356-: W->l----------------  359-: N->o[I];N-----------  361-: I->i----------------  364-: N->b;N--------------  366-: N->cti=E;N----------  370-: E->iK---------------  371-: K->[]---------------  374-: N->o[I];N-----------  376-: I->i----------------  379-: N->b;N--------------  381-: N->cti=E;N----------  385-: E->l----------------  387-: N->iK;N-------------  388-: K->[W]--------------  389-: W->i----------------  392-: N->w[R]{X}!{X}------  394-: R->igl--------------  399-: X->o[I];N-----------  401-: I->l----------------  404-: N->b;---------------  409-: X->o[I];N-----------  411-: I->l----------------  414-: N->b;---------------  ------------------------------------------------------------------------------------------ | |

## **Приложение Ж**

|  |
| --- |
| -------------------------------------------------------------------  ПОЛЬСКАЯ НОТАЦИЯ ВЫПОЛНЕНА  -------------------------------------------------------------------  ПРОМЕЖУТОЧНОЕ ПРЕДСТАВЛЕНИЕ КОДА  -------------------------------------------------------------------  0001 fti(0)[ti(1),ti(2)]{  0003 cti(3)=l(4);  0004 i(3)=i(1)i(2)+l(5)\*l(5)/##;  0005 i(3)=i(3)l(5)\*;  0006 r[i(3)];  0007 }  0009 fti(6)[]  0010 {  0011 cti(7)=l(8);  0012 o[i(7)];b;  0013 r[i(7)];  0014 }  0016 pi(9)[ti(10)]  0017 {  0018 cti(11)=l(4);  0019 o[l(12)];  0020 o[i(10)];b;  0021 u[i(11)gi(10)]  0022 {  0023 i(11)=i(11)l(13)+;  0024 o[i(11)];o[l(14)];  0025 }  0026 b;  0027 }  0029 m  0030 {  0031 cti(15)=kl(17)l(18)@2#;  0032 cti(19)=ki(15)l(21)@2#;  0033 o[i(19)];b;  0035 o[l(22)];b;  0036 cti(23)=l(24);  0037 o[i(23)];b;  0038 cti(25)=l(26);  0039 o[i(25)];b;  0040 o[l(27)];  0041 cti(28)=i(23)i(25)-;  0042 o[i(28)];b;  0044 cti(29)=l(30);  0045 o[l(31)];  0046 o[i(29)];b;  0048 cti(32)=l(33);  0049 o[i(32)];b;  0051 cti(34)=el(5)l(36)@2#;  0052 o[l(37)];  0053 o[i(34)];b;  0055 w[i(34)gl(38)]  0056 {  0057 o[l(39)];b;  0058 }  0059 !  0060 {  0061 o[l(40)];b;  0062 }  0064 cti(41)=dl(43)@1;  0065 o[l(44)];  0066 o[i(41)];b;  0068 cti(45)=zl(8)l(47)@2#;  0069 o[l(48)];  0070 o[i(45)];b;  0072 cti(49)=sl(51)l(52)@2#;  0073 o[l(53)];  0074 o[i(49)];b;  0076 cti(54)=i(0)l(13)l(55)@2#;  0077 o[i(54)];b;  0078 cti(56)=i(6)@0;  0079 o[i(56)];b;  0081 cti(57)=l(36);  0082 i(9)i(57)@1;  0084 w[i(57)gl(58)]  0085 {  0086 o[l(59)];b;  0087 }  0088 !  0089 {  0090 o[l(60)];b;  0091 }  0093 }  ------------------------------------------------------------------------------------------ |

## **Приложение З**

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib D:/БГТУ/GDD/GDD/Debug/Lib.lib  ExitProcess PROTO :DWORD  EXTRN COMBINE: proc  EXTRN LEXCOMP: proc  EXTRN MEASURE: proc  EXTRN BREAKL: proc  EXTRN RANDOMIZE: proc  EXTRN EXPONENT: proc  EXTRN read: proc  EXTRN OutputInt: proc  EXTRN OutputStr: proc  .stack 4096  .const  L1 SDWORD 0  L2 SDWORD 2  L3 SDWORD 3  L4 BYTE "Counting to length: ", 0  L5 SDWORD 1  L6 BYTE " ", 0  L7 BYTE "Test Course Project ", 0  L8 BYTE "GDD2023 ", 0  L9 BYTE "Start in 3 2 1", 0  L10 BYTE "Numbers: ", 0  L11 SDWORD -2  L12 SDWORD 8  L13 BYTE "bin - oct = ", 0  L14 SDWORD 1  L15 BYTE "Bool: ", 0  L16 BYTE "A", 0  L17 SDWORD 4  L18 BYTE "exp[2,4] = ", 0  L19 SDWORD 16  L20 BYTE "pow 2,4 really equal 16", 0  L21 BYTE "pow 2,4 not really equal 16", 0  L22 BYTE "Now", 0  L23 BYTE "length: ", 0  L24 SDWORD 40  L25 BYTE "Random: ", 0  L26 BYTE "Gddddd", 0  L27 BYTE "Gddd", 0  L28 BYTE "Lexcomp: ", 0  L29 SDWORD 5  L30 SDWORD 10  L31 BYTE "End of example if <= 10", 0  L32 BYTE "End of example if > 10", 0  .data  buffer BYTE 256 dup(0)  funcress SDWORD 0  funcctest SDWORD 0  cycleni SDWORD 0  mainstart DWORD ?  mainstartt DWORD ?  mainbin SDWORD 0  mainoct SDWORD 0  maindif SDWORD 0  mainbooly SDWORD 0  mainchar DWORD ?  mainexp SDWORD 0  mainlength SDWORD 0  mainrandd SDWORD 0  mainlexc SDWORD 0  maintestx SDWORD 0  maintesty SDWORD 0  maint SDWORD 0  .code  func PROC funca : SDWORD, funcb : SDWORD  push L1  pop funcress  push funca  push funcb  pop eax  pop ebx  add eax, ebx  push eax  push L2  pop eax  pop ebx  mul ebx  push eax  push L2  pop ebx  pop eax  cdq  idiv ebx  push eax  pop funcress  push funcress  push L2  pop eax  pop ebx  mul ebx  push eax  pop funcress  push funcress  jmp local0  local0:  pop eax  ret  func ENDP  funcc PROC  push L3  pop funcctest  push funcctest  call OutputInt  call BREAKL  pop eax  push funcctest  jmp local1  local1:  pop eax  ret  funcc ENDP  cyclen PROC cyclenn : SDWORD  push L1  pop cycleni  push offset L4  call OutputStr  push cyclenn  call OutputInt  call BREAKL  mov eax, cycleni  cmp eax, cyclenn  jl cycle0  jmp cyclenext0  cycle0:  push cycleni  push L5  pop eax  pop ebx  add eax, ebx  push eax  pop cycleni  push cycleni  call OutputInt  push offset L6  call OutputStr  mov eax, cycleni  cmp eax, cyclenn  jl cycle0  cyclenext0:  call BREAKL  ret  cyclen ENDP  main PROC  push offset L7  push offset L8  pop edx  pop edx  push offset L8  push offset L7  push offset buffer  call COMBINE  push eax  pop mainstart  push mainstart  push offset L9  pop edx  pop edx  push offset L9  push mainstart  push offset buffer  call COMBINE  push eax  pop mainstartt  push mainstartt  call OutputStr  call BREAKL  push offset L10  call OutputStr  call BREAKL  push L11  pop mainbin  push mainbin  call OutputInt  call BREAKL  push L12  pop mainoct  push mainoct  call OutputInt  call BREAKL  push offset L13  call OutputStr  push mainbin  push mainoct  pop ebx  pop eax  sub eax, ebx  push eax  pop maindif  push maindif  call OutputInt  call BREAKL  push L14  pop mainbooly  push offset L15  call OutputStr  push mainbooly  call OutputInt  call BREAKL  push offset L16  pop mainchar  push mainchar  call OutputStr  call BREAKL  push L2  push L17  pop edx  pop edx  push L17  push L2  call EXPONENT;  push eax  pop mainexp  push offset L18  call OutputStr  push mainexp  call OutputInt  call BREAKL  mov eax, mainexp  cmp eax, L19  jz m0  jnz m1  je m1  m0:  push offset L20  call OutputStr  call BREAKL  jmp e0  m1:  push offset L21  call OutputStr  call BREAKL  e0:  push offset L22  pop edx  push offset L22  call MEASURE;  push eax  pop mainlength  push offset L23  call OutputStr  push mainlength  call OutputInt  call BREAKL  push L3  push L24  pop edx  pop edx  push L24  push L3  call RANDOMIZE;  push eax  pop mainrandd  push offset L25  call OutputStr  push mainrandd  call OutputInt  call BREAKL  push offset L26  push offset L27  pop edx  pop edx  push offset L27  push offset L26  call LEXCOMP;  push eax  pop mainlexc  push offset L28  call OutputStr  push mainlexc  call OutputInt  call BREAKL  push func  push L5  push L29  pop edx  pop edx  push L29  push L5  call func  push eax  pop maintestx  push maintestx  call OutputInt  call BREAKL  push funcc  call funcc  push eax  pop maintesty  push maintesty  call OutputInt  call BREAKL  push L17  pop maint  push maint  call cyclen  mov eax, maint  cmp eax, L30  jl m2  jz m2  jg m3  je m3  m2:  push offset L31  call OutputStr  call BREAKL  jmp e1  m3:  push offset L32  call OutputStr  call BREAKL  e1:  push 0  call ExitProcess  main ENDP  end main |