МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 6-05-0612-01 Программная инженерия

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора LVK – 2024»

Выполнил студент Лужецкий Владислав Константинович

(Ф.И.О.)

Руководитель проекта асс. Гончар Е. А.

(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.)

Курсовой проект защищен с оценкой

Содержание

[Введение 4](#_Toc122337848)

[1 Спецификация языка программирования 5](#_Toc122337849)

[1.1 Характеристика языка программирования 5](#_Toc122337850)

[1.2 Определение алфавита языка программирования 5](#_Toc122337851)

[1.3 Применяемые сепараторы 5](#_Toc122337852)

[1.4 Применяемые кодировки 6](#_Toc122337853)

[1.5 Типы данных 6](#_Toc122337854)

[1.6 Преобразование типов данных 7](#_Toc122337855)

[1.7 Идентификаторы 7](#_Toc122337856)

[1.8 Литералы 8](#_Toc122337857)

[1.9 Объявление данных 8](#_Toc122337858)

[1.10 Инициализация данных 9](#_Toc122337859)

[1.11 Инструкции языка 9](#_Toc122337860)

[1.12 Операции языка 9](#_Toc122337861)

[1.13 Выражения и их вычисление 10](#_Toc122337862)

[1.14 Конструкции языка 10](#_Toc122337863)

[1.15 Области видимости идентификаторов 12](#_Toc122337864)

[1.16 Семантические проверки 12](#_Toc122337865)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc122337866)

[1.18 Стандартная библиотека и её состав 12](#_Toc122337867)

[1.19 Ввод и вывод данных 13](#_Toc122337868)

[1.20 Точка входа 13](#_Toc122337869)

[1.21 Препроцессор 14](#_Toc122337870)

[1.22 Соглашения о вызовах 14](#_Toc122337871)

[1.23 Объектный код 14](#_Toc122337872)

[1.24 Классификация сообщений транслятора 14](#_Toc122337873)

[1.25 Контрольный пример 14](#_Toc122337874)

[2 Структура транслятора 15](#_Toc122337875)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc122337876)

[2.2 Перечень входных параметров транслятора 16](#_Toc122337877)

[2.3 Протоколы, формируемые транслятором 16](#_Toc122337878)

[3 Разработка лексического анализатора 18](#_Toc122337879)

[3.1 Структура лексического анализатора 18](#_Toc122337880)

[3.2. Контроль входных символов 19](#_Toc122337881)

[3.3 Удаление избыточных символов 19](#_Toc122337882)

[3.4 Перечень ключевых слов 20](#_Toc122337883)

[3.5 Основные структуры данных 22](#_Toc122337884)

[3.6 Структура и перечень сообщений лексического анализатора 23](#_Toc122337885)

[3.7 Принцип обработки ошибок 24](#_Toc122337886)

[3.8 Параметры лексического анализатора 24](#_Toc122337887)

[3.9 Алгоритм лексического анализа 24](#_Toc122337888)

[3.10 Контрольный пример 24](#_Toc122337890)

[4. Разработка синтаксического анализатора 25](#_Toc122337891)

[4.1 Структура синтаксического анализатора 25](#_Toc122337892)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 25](#_Toc122337893)

[4.3 Построение конечного магазинного автомата 27](#_Toc122337894)

[4.4 Основные структуры данных 28](#_Toc122337895)

[4.5 Описание алгоритма синтаксического разбора 28](#_Toc122337896)

[4.6 Структура и перечень сообщений синтаксического анализатора 29](#_Toc122337897)

[4.7. Параметры синтаксического анализатора и режимы его работы 29](#_Toc122337898)

[4.8. Принцип обработки ошибок 29](#_Toc122337899)

[4.9. Контрольный пример 30](#_Toc122337900)

[5 Разработка семантического анализатора 31](#_Toc122337901)

[5.1 Структура семантического анализатора 31](#_Toc122337902)

[5.2 Функции семантического анализатора 31](#_Toc122337903)

[5.3 Структура и перечень сообщений семантического анализатора 31](#_Toc122337904)

[5.4 Принцип обработки ошибок 32](#_Toc122337905)

[5.5 Контрольный пример 32](#_Toc122337906)

[6. Вычисление выражений 33](#_Toc122337907)

[6.1 Выражения, допускаемые языком 33](#_Toc122337908)

[6.2 Польская запись и принцип её построения 33](#_Toc122337909)

[6.3 Программная реализация обработки выражений 33](#_Toc122337910)

[6.4 Контрольный пример 34](#_Toc122337911)

[7. Генерация кода 35](#_Toc122337912)

[7.1 Структура генератора кода 35](#_Toc122337913)

[7.2 Представление типов данных в оперативной памяти 35](#_Toc122337914)

[7.3 Статическая библиотека 36](#_Toc122337915)

[7.4 Особенности алгоритма генерации кода 36](#_Toc122337916)

[7.5 Входные параметры, управляющие генерацией кода 36](#_Toc122337917)

[7.6 Контрольный пример 37](#_Toc122337918)

[8. Тестирование транслятора 38](#_Toc122337919)

[8.1 Общие положения 38](#_Toc122337920)

[8.2 Результаты тестирования 38](#_Toc122337921)

[Список использованных источников 42](#_Toc122337924)

[Приложение А 43](#_Toc122337925)

[Приложение Б 51](#_Toc122337927)

[Приложение В 59](#_Toc122337926)

[Приложение Г 66](#_Toc122337927)

[Приложение Д 75](#_Toc122337928)

# Введение

В данном курсовом проекте поставлена задача разработки собственного языка программирования и транслятора для него. Название языка – LVK-2024. Написание транслятора будет осуществляться на языке C++, при этом код на языке LVK-2024 будет транслироваться в язык ассемблера.

В первом разделе представлена спецификация языка программирования LVK-2024. Описываются его ключевые особенности, синтаксис и семантика.

Во втором разделе рассматривается структура транслятора.

В третьем разделе подробно описан лексический анализатор, его структура и принцип работы. Рассматриваются правила, по которым исходный текст программы разбивается на токены.

Четвёртый раздел посвящён разработке синтаксического анализатора. Здесь разбираются принципы работы алгоритма, построение грамматик для языка LVK-2024.

В пятом разделе описан процесс разработки семантического анализатора. Рассматриваются проверки корректности типов, объявлений переменных, допустимости операций и других семантических правил языка.

В шестом разделе объясняется принцип работы польской (постфиксной) записи и её применение в трансляторе. Рассматриваются алгоритмы преобразования выражений из инфиксной в постфиксную форму.

Седьмой раздел описывает процесс генерации в код ассемблера.

В восьмом разделе описывается процесс тестирования транслятора.

Задание на курсовой проект можно разделить на следующие задачи:

1. Разработка спецификации языка LVK-2024;
2. разработка лексического анализатора;
3. разработка синтаксического анализатора;
4. разработка семантического анализатора;
5. разработка генератора кода;
6. тестирование транслятора.

# 1 Спецификация языка программирования

## 1.1 Характеристика языка программирования

Язык программирования LVK-2024 является процедурным, строго типизированным, не объектно-ориентированным, компилируемым.

## 1.2 Определение алфавита языка программирования

При написании программы на языке LVK-2024 используется таблица символов ASCII, представленная в рисунок.1.1.



Рисунок 1.1 Алфавит входных символов

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

## 1.3 Применяемые сепараторы

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

Таблица 1.1 Типы данных языка LVK-2024

|  |  |
| --- | --- |
| Символ(ы) | Назначение |
| ‘пробел’ | Разделитель цепочек. Допускается везде кроме названий идентификаторов и ключевых слов |

Продолжение таблицы 1.1

|  |  |
| --- | --- |
| Символ(ы) | Назначение |
| **[**,**]** | Блок функции или условной конструкции/цикла |
| **(**,**)** | Блок фактических или формальных параметров функции, а также приоритет арифметических операций |
| **,** | Разделитель параметров функций |
| **~** | Символ, отделяющий условные конструкции/циклы |
| **+,- ,\*,/,{,}** | Арифметические операции |
| **>,<,&,!,<=,>=** | Логические операции (операции сравнения: больше, меньше, проверка на равенство, на неравенство), используемые в условии цикла/условной конструкции. |
| **;** | Разделитель программных конструкций |
| **=** | Оператор присваивания |

## 1.4 Применяемые кодировки

Для написания программ язык LVK-2024 использует кодировку ASCII, содержащую английский алфавит, а также некоторые специальные символы, такие как [ ] ( ) , ; : ~ + - / \* > < & !.

## 1.5 Типы данных

В языке LVK-2024 реализованы три фундаментальных типа данных: беззнаковый целочисленный, символьный, строковый. Описание всех типов приведено в таблице 1.2.

Таблица 1.2 Типы данных языка LVK-2024

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| Беззнаковый целочисленный тип данных **num** | Фундаментальный тип данных. Используется для работы с числовыми значениями. В памяти занимает 4 байт. Максимальное значение: 4294967295.  Минимальное значение: 0.  Инициализация по умолчанию: значение 0.  Поддерживаемые операции:  **+** (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  **\*** (бинарный) – оператор умножения;  **/** (бинарный) – оператор деления;  **=** (бинарный) – оператор присваивания;  { (бинарный) – оператор сдвига влево;  } (бинарный) – оператор сдвига вправо.  В качестве условия цикла/условного оператора поддерживаются следующие логические операции: |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| Тип данных | Характеристика |
|  | **>** (бинарный) – оператор «больше»;  **<** (бинарный) – оператор «меньше»;  **!** (бинарный) – оператор проверки на неравенство;  >=( бинарный) – оператор «больше или равно»;  <=( бинарный) – оператор «меньше или равно»;  &(бинарный) – оператор проверки на равенство. |
| Символьный тип данных **symbol** | Фундаментальный тип данных. Используется для работы с символами, каждый из которых занимает 1 байт. Используется с одинарными кавычками ' '. Максимальное количество символов – 1. |
| Строковый тип данных **symbol** | Фундаментальный тип данных. Является расширением символьного типа данных. Используется с двойными кавычками “ ”. Используется для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 255. Операции над данными строкового типа: присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции. |

## 1.6 Преобразование типов данных

Преобразование типов данных в языке LVK-2024 не предусмотрено.

## 1.7 Идентификаторы

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Идентификаторы должны содержать только символы нижнего регистра латинского алфавита. Максимальная длина идентификатора равна восьми символам. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Префикс занимает 9 дополнительных символов. В случае превышения заданной длины, идентификаторы усекаются. Данные правила действуют для всех типов идентификаторов. Зарезервированные идентификаторы не предусмотрены. Идентификаторы не должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, имя функции, параметр функции. Имена идентификаторов-функций могут совпадать с именами команд ассемблера.

## 1.8 Литералы

С помощью литералов осуществляется инициализация переменных. Все литералы являются rvalue. Типы литералов языка LVK-2024 представлены в таблице 1.3.

Таблица 1.3 Литералы

|  |  |
| --- | --- |
| Литералы | Пояснение |
| Целочисленные литералы в десятичном представлении | Последовательность цифр 0…9 |
| Символьный литерал | Символ, заключенный в одинарные кавычки |
| Строковые литералы | Набор символов (от 1 до 255), заключённых в двойные кавычки |
| Целочисленные литералы в двоичном представлении | Последовательность цифр 0,1 с предшествующим символами ‘2x’ |
| Целочисленные литералы в восьмеричном представлении | Последовательность цифр 0…7 с предшествующим символами ‘8x’ |
| Целочисленные литералы в шестнадцатеричном представлении | Последовательность цифр 0…9 и символов a…f с предшествующим символами ‘16x’ |

Ограничения на строковые и строковые литералы языка LVK-2024: внутри литерала не допускается использование символов кириллицы, а также одинарных и двойных кавычек.

## 1.9 Объявление данных

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

Пример объявления числового типа с инициализацией:

# num primer = 12391; (десятичное представление)

# num primer = 2x110101; (двоичное представление)

# num primer = 8x237; (восьмеричное представление)

# num primer = 16xfa1f; (шестнадцатеричное представление)

Пример объявления переменной символьного типа с инициализацией:

# symbol test = ‘a’;

Пример объявления переменной строкового типа с инициализацией:

# symbol test= “Hello World”;

Для объявления функций используется ключевое слово function, перед которым указывается тип функции (если функция возвращает значение), или ключевое слово inane, если функция ничего не возвращает, а после – имя функции либо процедуры. Далее обязателен список параметров и тело функции.

## 1.10 Инициализация данных

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть только идентификаторы или литералы.

## 1.11 Инструкции языка

Инструкции языка LVK-2024 представлены в таблице 1.4.

Таблица 1.4 ИнструкцииязыкаLVK-2024

|  |  |
| --- | --- |
| Инструкция | Реализация |
| Объявление переменной | # <тип данных> <идентификатор>; |
| Объявление переменной с явной инициализацией | # <тип данных> <идентификатор> = <значение>;  Значение – инициализатор конкретного типа. Может быть только литералом или идентификатором |
| Возврат из функции или процедуры | Для функций, возвращающих значение:  finish <идентификатор/литерал>;  Для процедур:  finish; |
| Вывод данных | print <идентификатор/литерал>; |
| Вызов функции или процедуры | <идентификатор функции> (<список параметров>);  Список параметров может быть пустым. |
| Перевод строки | nl; |
| Присваивание | <идентификатор> = <выражение>;  Выражением может быть идентификатор, литерал, или вызов функции соответствующего типа. Для целочисленного типа выражение может быть дополнено арифметическими операциями с любым количеством операндов с использованием скобок. Для строкового типа выражение может быть только идентификатором, литералом или вызовом функции, возвращающей значение строкового типа. |

## 1.12 Операции языка

В языке LVK-2024 предусмотрены следующие операции с данными. Приоритетность операции умножения и деления выше приоритета операций сложения и вычитания. Для установки наивысшего приоритета используются круглые скобки. Операции языка представлены в таблице 1.5.

Таблица 1.5 Операции языка LVK-2024

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические | 1. **+ –** сложение 2. - – вычитание 3. \* – умножение 4. **/** – деление 5. = – присваивание 6. { – сдвиг влево 7. } – сдвиг вправо |
| Логические | 1. **>** – больше  2. **<** – меньше  3. ! – проверка на неравенство  4. <= – больше или равно  5. >= – меньше или равно  6. & – проверка на равенство |

## 1.13 Выражения и их вычисление

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

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

Перед генерацией кода каждое выражение приводится к записи в польской записи для удобства дальнейшего вычисления выражения на языке ассемблера.

## 1.14 Конструкции языка

Программа на языке LVK-2024 оформляется в виде функций пользователя и главной функции. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для лучшей читаемости кода.

Программные конструкции языка представлены в таблице 1.6.

Таблица 1.6 Программные конструкции языка LVK-2024

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция | root  [  …  ] |

Продолжение таблицы 1.6

|  |  |
| --- | --- |
| Внешняя функция | <тип данных> function <идентификатор> (<тип> <идентификатор>, ...)  […  finish <идентификатор/литерал>;  ] |
| Внешняя процедура | inane function <идентификатор> ( <тип> <идентификатор>, ...)  […  finish;  ] |
| Цикл | when: <идентификатор1> <оператор> <идентификатор2> ~  loop [ … ] ~  Цикл (операторы внутри блока loop) выполняется, пока истинно условие “<идентификатор1> <оператор> <идентификатор2>”, имеющее тот же смысл, что и в примере выше. |
| Условная конструкция | when: <идентификатор1> <оператор> <идентификатор2> ~  istrue [ … ]  isfalse [ … ] ~  <идентификатор1>, <идентификатор2> - идентификаторы или литералы целочисленного типа (но не два литерала одновременно). <оператор> - один из операторов сравнения ( > < & ! ), устанавливающий отношение между двумя операндами и организующий условие данной конструкции. При истинности условия выполняется код внутри блока istrue, иначе – код внутри блока isfalse. Любой из блоков istrue, isfalse может отсутствовать, но не оба блока одновременно. При отсутствии одного из блоков, в зависимости от истинности или ложности условия программа может как выполнить один из заявленных блоков, так и передать управление инструкции, следующей в коде за закрывающим условную конструкцию символом ‘~’. |
| Внешняя процедура | inane function <идентификатор> ( <тип> <идентификатор>, ...)  [ …  finish;  ] |

## 1.15 Области видимости идентификаторов

Область видимости: сверху вниз (как и в С++). Все объявления и операции с переменными происходят внутри какого-либо блока. Каждая переменная или параметр функции получают префикс – название функции, внутри которой они находятся.

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

## 1.16 Семантические проверки

В языке программирования LVK-2024 выполняются следующие семантические проверки:

1. Наличие функции root – точки входа в программу;
2. единственность точки входа;
3. переопределение идентификаторов;
4. использование идентификаторов без их объявления;
5. проверка соответствия типа функции и возвращаемого параметра;
6. правильность передаваемых в функцию параметров: количество, типы;
7. правильность строковых выражений;
8. превышение размера строковых и числовых литералов;
9. правильность составленного условия цикла/условного оператора;
10. проверка на совпадение названия переменной с названием функции из статической библиотеки.

## 1.17 Распределение оперативной памяти на этапе выполнения

Транслированный код использует две области памяти. В сегмент констант заносятся все литералы. В сегмент данных заносятся переменные и параметры функций. Локальная область видимости в исходном коде определяется за счет использования правил именования идентификаторов и регулируется их префиксами, что и обуславливает их локальность на уровне исходного кода, несмотря на то, что в оттранслированным в язык ассемблера коде переменные имеют глобальную область видимости.

## 1.18 Стандартная библиотека и её состав

В языке LVK-2024 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержимое библиотеки и описание функций представлено в таблице 1.8.

Таблица 1.8 Стандартная библиотека языка LVK-2024

|  |  |
| --- | --- |
| Функция | Описание |
| int atoii(“string”); | Беззнаковая целочисленная функция. Преобразует строку string в число и возвращает его. |

|  |  |
| --- | --- |
| int lenght(“string”); | Беззнаковая целочисленная функция.  Вычисляет и возвращает длину строки string. |

Продолжение таблицы 1.8

|  |  |
| --- | --- |
| int factorial(num); | Беззнаковая целочисленная функция. Вычисляет факториал беззнакового целочисленного числа int. |
| int power(int, int); | Беззнаковая целочисленная функция. Возводит беззнаковое целое число в степень. |
| int fibonachi(int); | Беззнаковая целочисленная функция. Вычисляет сумму чисел Фибоначчи. |

Стандартная библиотека написана на языке С++, подключается к транслированному коду на этапе генерации кода. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные конечному пользователю. Для вывода предусмотрен оператор print. Эти функции представлены в таблице 1.9.

Таблица 1.9 Дополнительные функции стандартной библиотеки

|  |  |
| --- | --- |
| Функция на языке С++ | Описание |
| void outnum(int value) | Функция для вывода в стандартный поток значения целочисленного идентификатора/литерала. |
| void outstr(char\* line) | Функция для вывода в стандартный поток значения строкового и символьного идентификатора/литерала. |

## 1.19 Ввод и вывод данных

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

Функции, управляющие выводом данных, реализованы на языке С++ и вызываются из транслированного кода, конечному пользователю недоступны. Пользовательская команда print в транслированном коде будут заменена вызовом нужных библиотечных функций. Библиотека, содержащая нужные процедуры, подключается на этапе генерации кода.

## 1.20 Точка входа

В языке LVK-2024 каждая программа должна содержать главную функцию (точку входа) root, с первой инструкции которой начнётся последовательное выполнение команд программы.

## 1.21 Препроцессор

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке LVK-2024 отсутствует.

## 1.22 Соглашения о вызовах

В языке вызов функций происходит по соглашению о вызовах stdcall. Осо-бенности stdcall:

– Все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## 1.23 Объектный код

Язык LVK-2024 транслируется в язык ассемблера, а затем - в объектный код.

## 1.24 Классификация сообщений транслятора

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы. Сообщения транслятора приведены в таблице 1.10, а также они приведены в приложении А.

Таблица 1.10 Классификация ошибок

|  |  |
| --- | --- |
| Номера ошибок | Характеристика |
| 0 – 199 | Системные ошибки |
| 200 – 299 | Ошибки лексического анализа |
| 300 – 399 | Ошибки семантического анализа |
| 600 – 699 | Ошибки синтаксического анализа |
| 400-499, 700-999 | Зарезервированные коды ошибок |

## 1.25 Контрольный пример

Контрольный пример демонстрирует главные особенности языка LVK-2024: его фундаментальные типы, основные структуры, функции, процедуры, использование функций статической библиотеки. Исходный код контрольного примера представлен в приложении А.

# 2 Структура транслятора

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

В языке LVK-2024 исходный код транслируется в язык Assembler. Транслятор языка разделён на отдельные части, которые взаимодействуют между собой и выполняют отведённые им функции, которые представлены в пункте 2.1. Для того чтобы получить ассемблерный код, используется выходные данные работы лексического анализатора, а именно таблица лексем и таблица идентификаторов. Для указания выходных файлов используются входные параметры транслятора, которые описаны в таблице 2.1. Структура транслятора языка LVK-2024 приведена на рисунке 2.1.



Рисунок 2.1 Структура транслятора языка программирования LVK-2024

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся последовательность символов входного языка. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. 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.

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 транслятора

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

Таблица 2.1 Входные параметры транслятора языка LVK-2024

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке LVK-2024, имеющий расширение .txt | Не предусмотрено |
| -log:<путь к log-файлу> | Файл журнала для вывода протоколов работы программы. | Значение по умолчанию:  <имя in-файла>.log |

## 2.3 Протоколы, формируемые транслятором

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

Таблица 2.2 Протоколы, формируемые транслятором языка LVK-2024

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования LVK-2024. 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. |

# 3 Разработка лексического анализатора

## 3.1 Структура лексического анализатора

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором. На вход лексического анализатора подаётся исходный код входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка. Лексический анализатор производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив токенов.

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.

Исходный код программы представлен в приложении А, структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 Структура лексического анализатора

Функции лексического анализатора:

− удаление «пустых» символов. Если «пустые» символы (пробелы, знаки табуляции и перехода на новую строку), синтаксический анализатор никогда не столкнется с ними (альтернативный способ, состоящий в модификации грамматики для включения «пустых» символов и комментариев в синтаксис, достаточно сложен для реализации);

− распознавание идентификаторов и ключевых слов;

− распознавание констант;

− распознавание разделителей и знаков операций.

## 3.2. Контроль входных символов

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

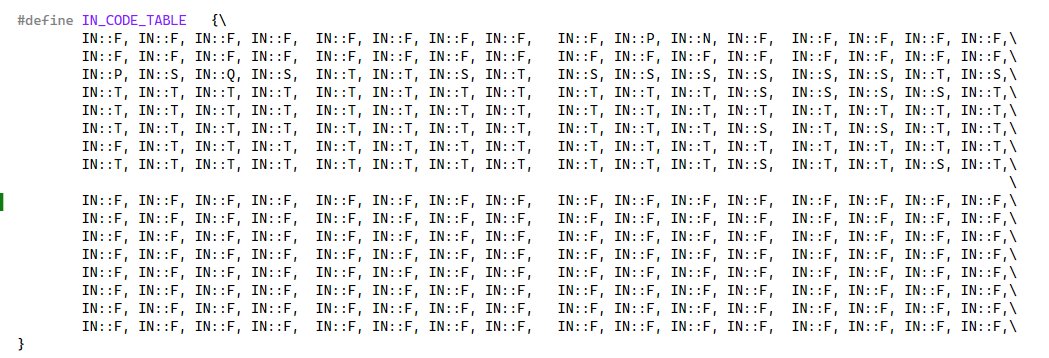


Рисунок 3.2. Таблица контроля входных символов

Таблица 3.1 Соответствие символов и их значений в таблице

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

## 3.3 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на токены.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом программы.
2. Встреча пробела или знака табуляции является своего рода встречей символа-сепаратора.
3. В отличие от других символов-сепараторов не записываем в очередь лексем эти символы, т.е. игнорируем.

## 3.4 Перечень ключевых слов

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

Таблица 3.2 Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| num, symbol | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 8 символов. |
| Литерал | l | Литерал любого доступного типа. |
| function | f | Объявление функции. |
| inane | p | Ключевое слово для процедур – функций, не возвращающих значения. Указывается перед словом function. |
| finish | e | Выход из функции/процедуры. |
| root | m | Главная функция. |
| # | n | Объявление переменной. |
| print | @ | Вывод данных. |
| when: | ? | Указывает начало цикла/условного оператора. |
| istrue | r | Истинная ветвь условного оператора. |
| isfalse | w | Ложная ветвь условного оператора. |
| loop | c | Указывает на начало тела цикла. |
| nl | ^ | Оператор вывода символа перевода строки. |
| ~ | ~ | Разделение конструкций в цикле/условном операторе. |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| [ | [ | Начало блока/тела функции. |
| ] | ] | Закрытие блока/тела функции. |
| ( | ( | Передача параметров в функцию, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | = | Знак присваивания. |
| +  -  \*  /  }  { | +  -  \*  /  }  { | Знаки операций. |

Продолжение таблицы 3.2

|  |  |  |
| --- | --- | --- |
| >  <  !  >=  <= | >  <  !  x  y | Знаки логических операторов |

Пример реализации таблицы лексем представлен в приложении Б.

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся токен и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Структура конечного автомата изображен на рисунке 3.3.

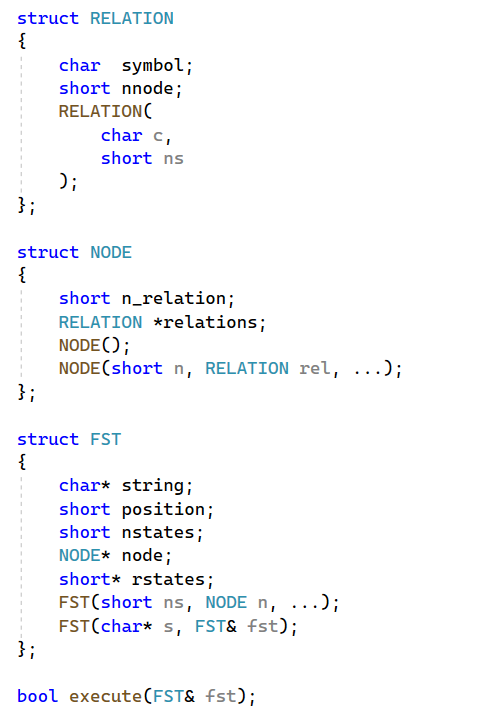
****

Рисунок 3.3 Структура конечного автомата

Пример графа перехода конечного автомата изображен на рисунке 3.4.

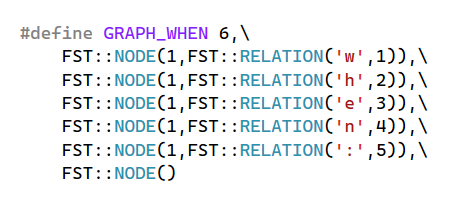


Рисунок 3.4 Пример реализации графа конечного автомата для токена when

Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов.

## 3.5 Основные структуры данных

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы, лексему (lexema), полученную при разборе, номер строки в исходном коде (sn), и номер в таблице идентификаторов, если лексема является идентификаторов (idxTI). Код C++ со структурой таблицы лексем представлен на рисунке 3.3. Код C++ со структурой таблицы идентификаторов представлен на рисунке 3.4.

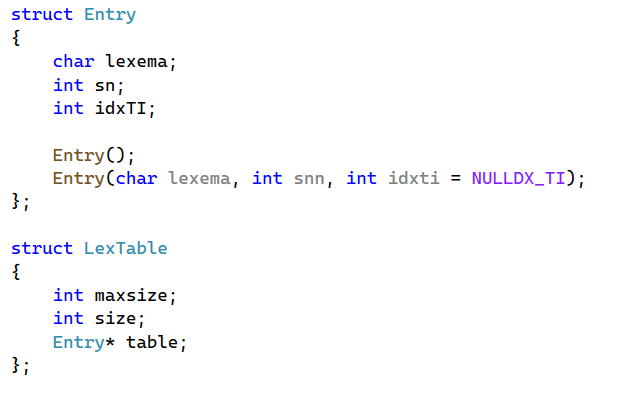


Рисунок 3.3 Структура таблицы лексем

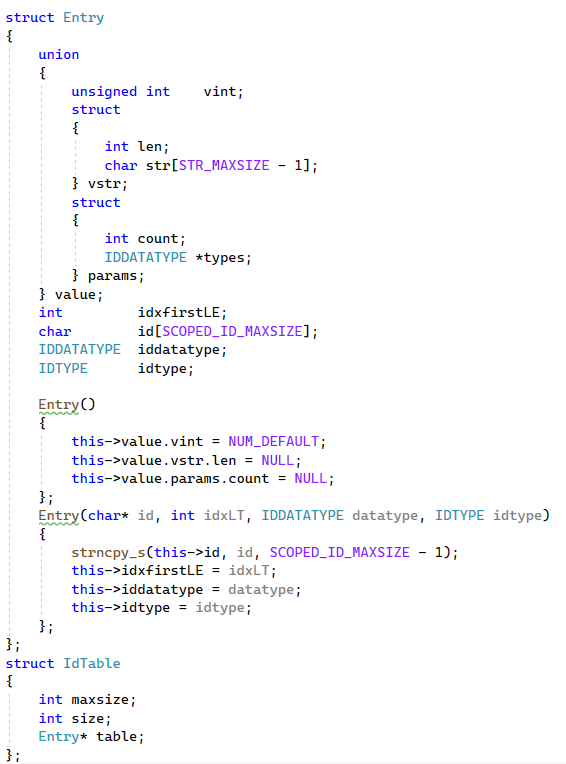


Рисунок 3.4 Структура таблицы идентификаторов

Таблица идентификаторов содержит имя идентификатора (id), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и значение (или параметры функций) (value).

## 3.6 Структура и перечень сообщений лексического анализатора

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. Перечень сообщений представлен на рисунке 3.5.

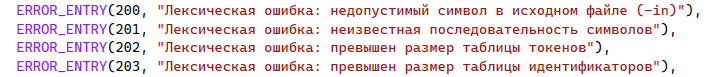


Рисунок 3.5 - Сообщения лексического анализатора

При возникновении сообщения, лексический анализатор игнорирует найденную ошибку и продолжает работу с исходным кодом.

## 3.7 Принцип обработки ошибок

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением.

## 3.8 Параметры лексического анализатора

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся как в файл журнала, так и в консоль.

## 3.9 Алгоритм лексического анализа

1. проверяет входной поток символов программы на исходном языке на допустимость, удаляет лишние пробелы и добавляет сепаратор для вычисления номера строки для каждой лексемы;
2. для выделенной части входного потока выполняется функция распознавания лексемы;
3. при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;
4. формирует протокол работы;
5. при неуспешном распознавании выдается сообщение об ошибке.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «symbol» представлен на рисунке 3.6, где S0 – начальное, а S6 – конечное состояние автомата.

s y m b o l

Рисунок 3.6 Пример графа переходов для цепочки symbol

## 3.10 Контрольный пример

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

# 4. Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора

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

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

Рисунок 4.1 Структура синтаксического анализатора.

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка LVK-2024 используется контекстно-свободная грамматика , где:

1. T – множество терминальных символов.

2. N – множество нетерминальных символов (первый столбец таблицы 4.1).

3. P – множество правил языка (второй столбец таблицы 4.1).

4. 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.

Описание нетерминальных символов содержится в таблице 4.1.

Таблица 4.1 Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| S | S->tfiPTS  S->pfiPGS  S->m[K] | Стартовые правила, описывающее общую структуру программы |
| P | P->(E)  P->() | Правила для параметров объявляемых функций |
| T | T->[eV;]  T->[KeV;] | Правила для тела функций |
| G | G->[e;]  G->[Ke;] | Правила для тела процедур |
| E | E->ti,E  E->ti | Правила для списка параметров функции |
| F | F->(N)  F->() | Правила для вывозов функций(в т.ч. и в выражениях) |
| N | N->i  N>l  N->l,N  N->I,N | Правила для параметров вызываемых функций |
| R | R->rY~  R>wY~  R>cY~  R->rYwY~  R->wYrY~ | Правила составления цикла/условного оператора |
| Z | Z->iLi  Z->iLl  Z->lli | Правила для условия цикла/условного оператора |
| L | L-><  L->>  L->!  L->>=  L-><= | Правила для логических операторов |
| A | A->+  A->-  A->\*  A->/  A->{  A->} | Правила для арифметических операторов |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| V | V->l  V->i  V->q | Правила для простых выражений |
| Y | Y->[X] | Правила для тела цикла/условного выражения |
| W | W->l  W->i  W->(W)  W->(W)AW  W->iF  W->iAW  W->lAW  W->iFAW | Правила для сложных выражений |
| K | K->nti=V;K  K->nti;K  K->i=W;K  K->oV;K  K->^;K  K->&Z~RK  K->iF;K  K->nti=V;  K->nti;  K->i=W;  K->oV;  K->^;  K->&Z~R  K->iF; | Программные конструкции |
| X | X->i=W;X  X->oV;X  X->^;X  X->iF;X  X->i=W;  X->oV;  X->^;  X->iF; | Программные конструкции внутри цикла/условного оператора |

## 4.3 Построение конечного магазинного автомата

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

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки, стек автомата |
|  | Алфавит входных символов | Алфавит представляет из себя множества терминальных и нетерминальных символов. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ ]) |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ] |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

## 4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка LVK-2024. Данные структуры в приложении В.

## 4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата следующий:

1. В магазин записывается стартовый символ;
2. на основе полученных ранее таблиц формируется входная лента;
3. напускается 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 4.3.

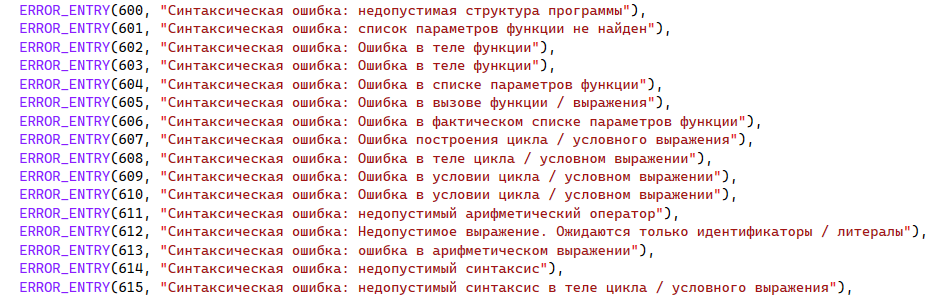


Рисунок 4.3 - Сообщения синтаксического анализатора

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

## 4.7. Параметры синтаксического анализатора и режимы его работы

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

## 4.8. Принцип обработки ошибок

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку. Тогда анализ останавливается и выводится сообщение об ошибке (если она найдена). Если в процессе анализа более трёх ошибок, то анализ останавливается.

## 4.9. Контрольный пример

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

# 5 Разработка семантического анализатора

## 5.1 Структура семантического анализатора

Семантический анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.

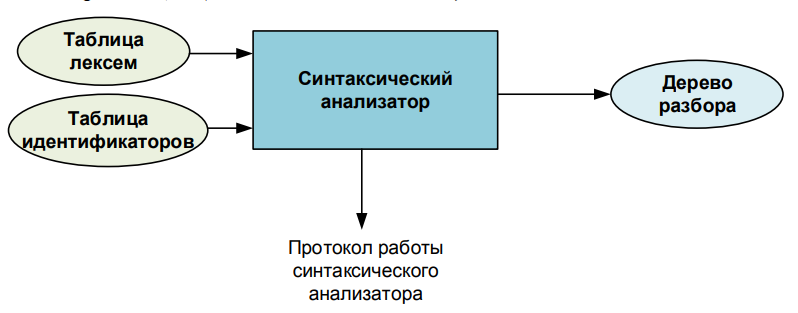


Рисунок 5.1. Структура семантического анализатора

Некоторые проверки (такие как проверка на единственность точки входа, проверка на предварительное объявление переменной) осуществляется лексическим анализом.

## 5.2 Функции семантического анализатора

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

## 5.3 Структура и перечень сообщений семантического анализатора

Сообщения, формируемые семантическим анализатором, представлены на рис 5.1.

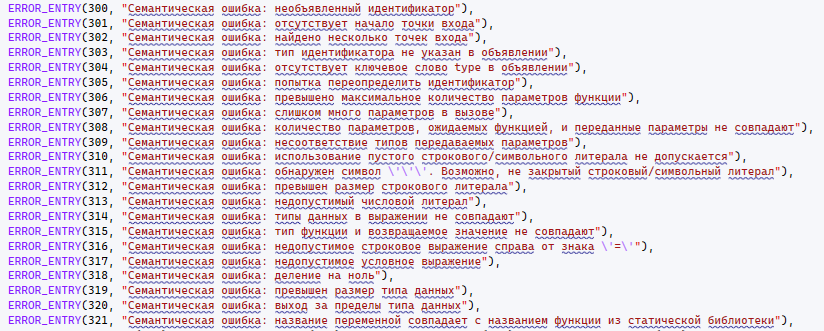


Рисунок 5.2 – Перечень сообщений семантического анализатора

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

## 5.4 Принцип обработки ошибок

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит протоколирование данных ошибок с номером ошибки и диагностическим сообщением.

## 5.5 Контрольный пример

Соответствие примеров ошибок в исходном коде диагностических сообщений об ошибках приведено в таблице 5.1.

Таблица 5.1. Примеры диагностики ошибок

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| root[  num x = 9;  print x;  ] | Ошибка N304: Семантическая ошибка: В объявлении отсутствует ключевое слово # Строка: 2 |
| root[  # num x = 9;  # symbol y =x;  ] | Ошибка N314: Семантическая ошибка: Типы данных в выражении не совпадают Строка: 3 |
| root [  # num x = 9;  ]  root [  # symbol y = ‘a’;  ] | Ошибка N302: Семантическая ошибка: Обнаружено несколько точек входа root Строка: 0 |

# 6. Вычисление выражений

## 6.1 Выражения, допускаемые языком

В языке LVK-2024 допускаются вычисления выражений целочисленного типа данных с поддержкой вызова функций внутри выражений. Приоритет операций представлен на таблице 6.1.

Таблица 6.1. Приоритеты операций

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

## 6.2 Польская запись и принцип её построения

Все выражения языка LVK-2024 преобразовываются к обратной польской записи.

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

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.

## 6.3 Программная реализация обработки выражений

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

## 6.4 Контрольный пример

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

Таблица 6.2. Преобразование выражений к ПОЛИЗ

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

# 7. Генерация кода

## 7.1 Структура генератора кода

В языке LVK-2024 генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В соответствии с таблицей лексем строится файл на языке ассемблера, который будет являться результатом работы транслятора. Структура генератора кода LVK-2024 представлена на рисунке 7.1.



Рисунок 7.1 – Структура генератора кода

В случае возникновения ошибок генерация кода не будет осуществляться.

## 7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке LVK-2024 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка LVK-2024 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке LVK-2024 | Тип идентификатора на языке ассемблера | Пояснение |
| num | dword | Хранит целочисленный тип данных. |
| symbol(символьный) | byte | Хранит указатель на первый символ. |
| symbol(строковый) | byte | Хранит указатель на первый символ. |

## 7.3 Статическая библиотека

В языке LVK-2024 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера. Объявление функций статической библиотеки генерируется автоматически. Функции статической библиотеки представлены в таблице 7.2.

Таблица 7.2 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| unsigned int outlstr(char\* sym) | Вывод на консоль символа/строки |
| unsigned int outnum(int num) | Вывод на консоль беззнаковой целочисленной переменной num |
| unsigned int lenght(char\* str) | Вычисление длины строки |
| unsigned int atoii(char\* buffer, char\* ptr) | Преобразование строки в число |
| unsigned int factorial(unsigned int number) | Вычисление факториала числа |
| unsigned int power(unsigned int number, unsigned int pow) | Возведение беззнакового целого числа в степень |
| unsigned int fibonachi(unsigned int number) | Вычисляет число последовательности Фибоначчи |

## 7.4 Особенности алгоритма генерации кода

В языке LVK-2024 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2.

Рисунок 7.2 – Структура генератора кода

Особенность алгоритма – это оптимизация инструкций с учётом типов данных и поддержка вызовов функций из статической библиотеки.

## 7.5 Входные параметры, управляющие генерацией кода

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке LVK-2024. Результаты работы генератора кода выводятся в файл с расширением .asm.

## 7.6 Контрольный пример

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Д. Результат работы контрольного примера приведён на рисунке 7.3.

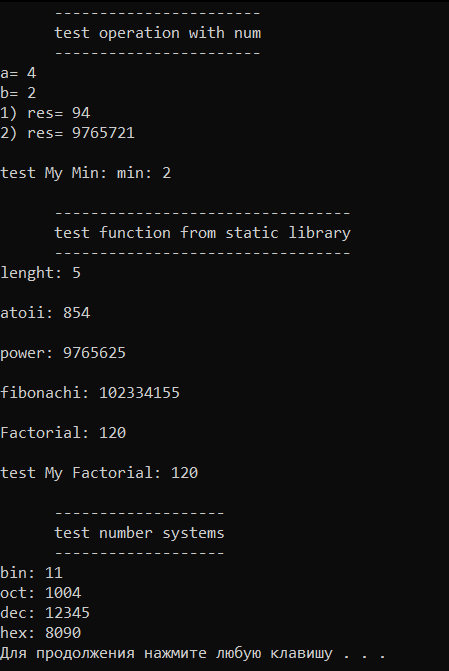


Рисунок 7.3 Результат работы программы на языке LVK-2024

# 8. Тестирование транслятора

## 8.1 Общие положения

Для проверки транслятора языка LVK-2024 на всех стадиях трансляции: лексический, синтаксический и семантический анализ, а также генерация машинного кода был сделан подбор тестов. Основной задачей тестов является выявление всех возможных ошибок, указанных в спецификации, и проверка корректной работы транслятора в условиях как корректного, так и ошибочного исходного кода.

Основные принципы тестирования:

1. Проверка системных ошибок
2. Проверка лексического анализа.
3. Проверка синтаксического анализа.
4. Проверка семантического анализа.
5. Проверка генерации кода.

## 8.2 Результаты тестирования

В языке LVK-2024 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| root [п] | Ошибка N200: Лексическая ошибка: Недопустимый символ в исходном файле(-in) Строка: 2 Позиция в строке: 2 |

На этапе лексического анализа в языке LVK-2024 могут возникнуть ошибки. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 - Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| root  [# num a19;] | Ошибка N201: Лексическая ошибка: Неизвестная последовательность символов Строка: 2 |

На этапе синтаксического анализа в языке LVK-2024 могут возникнуть ошибки. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 - Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| root # num x; ] | Ошибка 600: строка 1, Синтаксическая ошибка: недопустимая структура программы |

Продолжение таблицы 8.3

|  |  |
| --- | --- |
| symbol function fi([]  root[] | Ошибка 601: строка 1, Синтаксическая ошибка: список  параметров функции не найден |
| symbol function fi()  [nl; print]  root[] | Ошибка 602: строка 1, Синтаксическая ошибка: Ошибка в теле функции |
| inane function fi()  [nl print;]  root[] | Ошибка 603: строка 1, Синтаксическая ошибка: Ошибка в теле процедуры |

|  |  |
| --- | --- |
| inane function fi(num num)[]  root[] | Ошибка 604: строка 1, Синтаксическая ошибка: Ошибка в списке параметров функции |
| symbol function fi(num x)[finish 3;]  root [nl;fi(5,5; nl;] | Ошибка 605: строка 2, Синтаксическая ошибка: Ошибка в вызове функции/выражении |
| symbol function fi(num x)[finish 3;]  root [nl;fi(5,5,5 5);] | Ошибка 606: строка 2, Синтаксическая ошибка: Ошибка в списке фактических параметров функции |
| root [# num x; when: x > 2 ~ loop ~] | Ошибка 607: строка 1, Синтаксическая ошибка: Ошибка при конструировании цикла/условного выражения |
| root [# num x; when: x > 2 4 loop ~] | Ошибка 608: строка 1, Синтаксическая ошибка: Ошибка в теле цикла/условного выражения |
| root [when: 1 = 2 ~] | Ошибка 609: строка 1, Синтаксическая ошибка: Ошибка в условии цикла/условного выражения |
| root [when: 1 = 2 ~] | Ошибка 610: строка 1, Синтаксическая ошибка: Неверный условный оператор |
| root [# num x; x = x ! x;] | Ошибка 611: строка 1, Синтаксическая ошибка: Недопустимый арифметический оператор |
| root [# num x; print new;] | Ошибка 612: строка 1, Синтаксическая ошибка: Недопустимое выражение. Ожидаются только идентификаторы/литералы |
| root [# num x; x = 1 ++;] | Ошибка 613: строка 1, Синтаксическая ошибка: Ошибка в арифметическом выражении |
| root [nl; 4;] | Ошибка 614: строка 1, Синтаксическая ошибка: Недопустимый синтаксис |
| root [# num a; when: a < 3 ~ istrue [nl; 3;] ~] | Ошибка 615: строка 1, Синтаксическая ошибка: Недопустимый синтаксис в теле цикла/условного выражения |

Семантический анализ в языке LVK-2024 содержит множество проверок по семантическим правилам. Итоги тестирования семантического анализатора на корректное обнаружение семантических ошибок приведены в таблице 8.4.

Таблица 8.4 - Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| root [ a = 1 ] | Ошибка 300: Семантическая ошибка: Необъявленный идентификатор Строка: 1 |
| symbol function fi()[] | Ошибка 301: Семантическая ошибка: Отсутствует точка входа root Строка: 0 |
| root []  root [] | Ошибка 302: Семантическая ошибка: найдено несколько точек входа Строка: 0 |
| root [a = 1;] | Ошибка 304: Семантическая ошибка: В объявлении отсутствует ключевое слово # Строка: 1 |
| root [# num t;  # symbol t;] | Ошибка 305: Семантическая ошибка: Попытка переопределения идентификатора Строка: 3 |
| inane function fi()[]  root [fi(“a”,”b”,”c”,”d”] | Ошибка 307: Семантическая ошибка: Слишком много параметров в вызове Строка: 1 |
| symbol function fi(symbol x, symbol y, symbol z, symbol s)  root [] | Ошибка 306: Семантическая ошибка: Превышено максимальное количество параметров функции Строка: 1 |
| symbol function fi(symbol x)[finish "a";]  root [fi("a", "b");] | Ошибка 308: Семантическая ошибка: Количество ожидаемых функцией и передаваемых параметров не совпадают Строка: 2 |
| symbol function fi(symbol x)[finish "a";]  root [fi("a", "b");] | Ошибка 309: Семантическая ошибка: несоответствие типов передаваемых параметров Строка: 2 |
| root [# symbol x="";] | Ошибка 310: Семантическая ошибка: Использование пустого строкового литерала недопустимо Строка: 1 |
| root [# symbol x=";] | Ошибка 311: Семантическая ошибка: Обнаружен символ '"'. Возможно, не закрыт строковый литерал Строка: 1 |
| root [# num x;  x=123421554645645432;] | Ошибка 313: Семантическая ошибка: Недопустимый целочисленный литерал Строка: 1 |
| root [# num x;  x = 5 + "abc";] | Ошибка 314: Семантическая ошибка: Типы данных в выражении не совпадают Строка: 2 |
| symbol function fi()[finish 5;]  root [nl;] | Ошибка 315: Семантическая ошибка: Тип функции и возвращаемого значения не совпадают Строка: 1 |
| root [# symbol x; x = "abc" + "d";] | Ошибка 316: Семантическая ошибка: Недопустимое строковое выражение справа от знака '=' Строка: 1 |
| root [when: "symbol"& 6~  istrue[print "symbol";]] | Ошибка 317: Семантическая ошибка: Неверное условное выражение Строка: 1 |
| root [# num a =5;  a = a/0; print a;] | Ошибка 318: Семантическая ошибка: Деление на ноль Строка: 4 |

Продолжение таблицы 8.4

|  |  |
| --- | --- |
| root [# num x;  x=16xfffffffff;] | Ошибка 319: Семантическая ошибка: превышен размер типа данных Строка: 2 |
| root [# num x = 16xffffffff;  x=x+1;] | Ошибка 320: Семантическая ошибка: выход за пределы типа данных Строка: 2 |
| # num power = 1; | Ошибка 321: Семантическая ошибка: название переменной совпадает с названием функции из статической библиотеки Строка: 1 |

# Заключение

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования LVK-2024 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка LVK-2024;
2. разработаны конечные автоматы и алгоритмы на их основе для эффективной работы лексического анализатора;
3. осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
5. осуществлена программная реализация синтаксического анализатора;
6. разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
7. разработан транслятор кода на язык ассемблера;
8. проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка LVK-2024 включает:

1. Три типа данных;
2. поддержка операторов вывода и перевода строки;
3. возможность вызова функций стандартной библиотеки;
4. наличие арифметических операторов для вычисления выражений;
5. поддержка функций, процедур, операторов цикла и условия;
6. структурированная и классифицированная система для обработки ошибок пользователя.

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

# Список использованных источников

1. Курс лекций по КПО Наркевич А.С.
2. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
3. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.
4. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.
5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с

# Приложение А

Листинг 1 – Исходный код программы на языка LVK-2024

num function fact(num number)[

# num i = 1;

# num res = 1;

when: number > i

~

loop[

res = res \* i;

i = i+1;

]

~

res = res \* number;

finish res;

]

inane function myprint(symbol a)[

nl;

print "myprint: ";

print a;

finish;

]

num function mymin(num a, num b)[

# num result;

when: a > b

~

istrue[result = b;]

isfalse[result = a;]

~

finish result;

]

root[

print " -----------------------";

nl;

print " test operation with num";

nl;

print " -----------------------";

nl;

# num a = 4;

# num b = 2;

# num res;

print "a= ";

print a;

nl;

print "b= ";

print b;

nl;

res = a + b \* (100 - 10) / 2;

print "1) res= ";

print res;

res = res + b + power(5,10);

nl;

print "2) res= ";

print res;

nl;

nl;

# num mm;

print "test My Min: ";

mm = mymin(a, b);

print "min: ";

print mm;

nl;

nl;

print " ---------------------------------";

nl;

print " test function from static library";

nl;

print " ---------------------------------";

nl;

# num le;

le = lenght("hello");

print "lenght: ";

print le;

nl;

nl;

# num e;

e = atoii("5125");

print "atoii: ";

e = e / 6;

print e;

nl;

nl;

e = power(5,10);

print "power: ";

print e;

nl;

nl;

e = fibonachi(40);

print "fibonachi: ";

print e;

nl;

nl;

e = factorial(5);

print "Factorial: ";

print e;

nl;

nl;

print "test My Factorial: ";

e = fact(5);

print e;

nl;

nl;

print " -------------------";

nl;

print " test number systems";

nl;

print " -------------------";

nl;

# num bin = 2x1011;

# num oct = 8x1754;

# num dec = 12345;

# num hex = 16x1f9a;

print "bin: ";

print bin;

nl;

print "oct: ";

print oct;

nl;

print "dec: ";

print dec;

nl;

print "hex: ";

print hex;

nl;

]

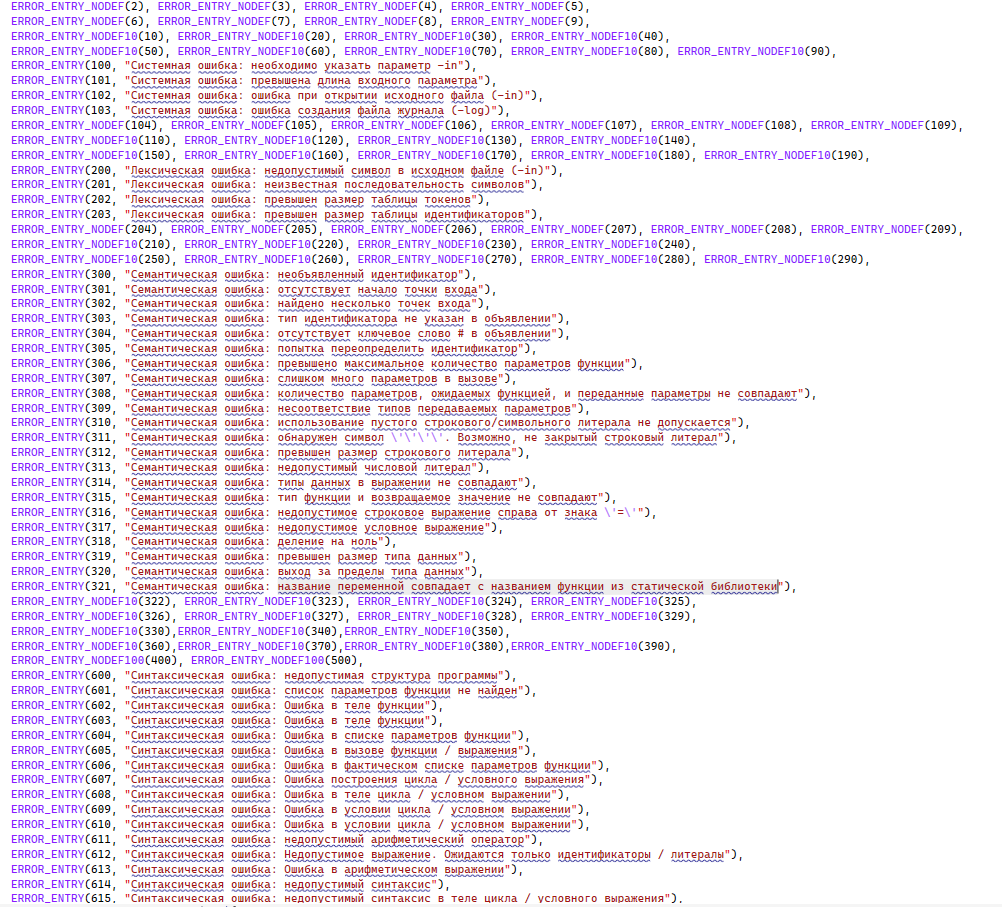


Рисунок 1 – Таблица ошибок языка LVK-2024

**Приложение Б**

Листинг 1 Таблица идентификаторов контрольного примера

~~~~~~~~~~~~~~~~ТАБЛИЦА ИДЕНТИФИКАТОРОВ ~~~~~~~~~~~~~~~~~~~~~

| № |Линия в LT | ID Тип | Название | Значение(аргументы)

| 0 | 2 | number function | fact | P0:NUMBER |

| 1 | 5 | number parameter | factnumber |

| 2 | 10 | number variable | facti |0

| 3 | 12 | number literal | LTRL1 |1

| 4 | 16 | number variable | factres |0

| 5 | 53 | proc function | myprint | P0:SYMBOL |

| 6 | 56 | symbol parameter | myprinta |

| 7 | 62 | symbol literal | LTRL2 |[9]myprint:

| 8 | 72 | number function | mymin | P0:NUMBER | P1:NUMBER |

| 9 | 75 | number parameter | mymina |

| 10 | 78 | number parameter | myminb |

| 11 | 83 | number variable | myminresult |0

| 12 | 112 | symbol literal | LTRL3 |[29] -----------------------

| 13 | 117 | symbol literal | LTRL4 |[29] test operation with num

| 14 | 128 | number variable | roota |0

| 15 | 130 | number literal | LTRL5 |4

| 16 | 134 | number variable | rootb |0

| 17 | 136 | number literal | LTRL6 |2

| 18 | 140 | number variable | rootres |0

| 19 | 143 | symbol literal | LTRL7 |[3]a=

| 20 | 151 | symbol literal | LTRL8 |[3]b=

| 21 | 165 | number literal | LTRL9 |100

| 22 | 167 | number literal | LTRL10 |10

| 23 | 173 | symbol literal | LTRL11 |[8]1) res=

| 24 | 184 | number PIE LIBf | power | P0:NUMBER | P1:NUMBER |

| 25 | 186 | number literal | LTRL12 |5

| 26 | 194 | symbol literal | LTRL13 |[8]2) res=

| 27 | 205 | number variable | rootmm |0

| 28 | 208 | symbol literal | LTRL14 |[13]test My Min:

| 29 | 220 | symbol literal | LTRL15 |[5]min:

| 30 | 230 | symbol literal | LTRL16 |[39] ---------------------------------

| 31 | 235 | symbol literal | LTRL17 |[39] test function from static library

| 32 | 246 | number variable | rootle |0

| 33 | 250 | number PIE LIBf | lenght | P0:SYMBOL |

| 34 | 252 | symbol literal | LTRL18 |[5]hello

| 35 | 256 | symbol literal | LTRL19 |[8]lenght:

| 36 | 267 | number variable | roote |0

| 37 | 271 | number PIE LIBf | atoii | P0:SYMBOL |

| 38 | 273 | symbol literal | LTRL20 |[4]5125

| 39 | 277 | symbol literal | LTRL21 |[7]atoii:

| 40 | 283 | number literal | LTRL22 |6

| 41 | 302 | symbol literal | LTRL23 |[7]power:

| 42 | 313 | number PIE LIBf | fibonachi | P0:NUMBER |

| 43 | 315 | number literal | LTRL24 |40

| 44 | 319 | symbol literal | LTRL25 |[11]fibonachi:

| 45 | 330 | number PIE LIBf | factorial | P0:NUMBER |

| 46 | 336 | symbol literal | LTRL26 |[11]Factorial:

| 47 | 346 | symbol literal | LTRL27 |[19]test My Factorial:

| 48 | 363 | symbol literal | LTRL28 |[25] -------------------

| 49 | 368 | symbol literal | LTRL29 |[25] test number systems

| 50 | 379 | number variable | rootbin |0

| 51 | 381 | number literal | LTRL30 |11

| 52 | 385 | number variable | rootoct |0

| 53 | 387 | number literal | LTRL31 |1004

| 54 | 391 | number variable | rootdec |0

| 55 | 393 | number literal | LTRL32 |12345

| 56 | 397 | number variable | roothex |0

| 57 | 399 | number literal | LTRL33 |8090

| 58 | 402 | symbol literal | LTRL34 |[5]bin:

| 59 | 410 | symbol literal | LTRL35 |[5]oct:

| 60 | 418 | symbol literal | LTRL36 |[5]dec:

| 61 | 426 | symbol literal | LTRL37 |[5]hex:

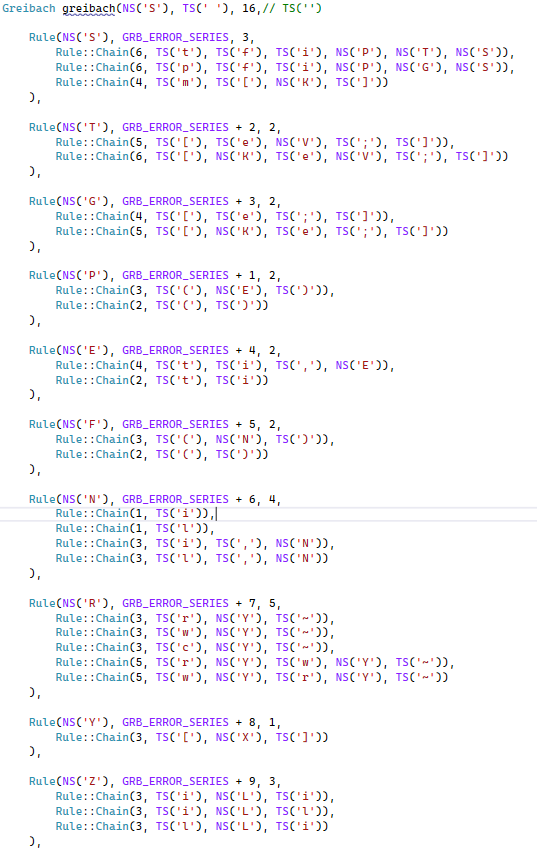
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Листинг 2 Таблица лексем после контрольного примера

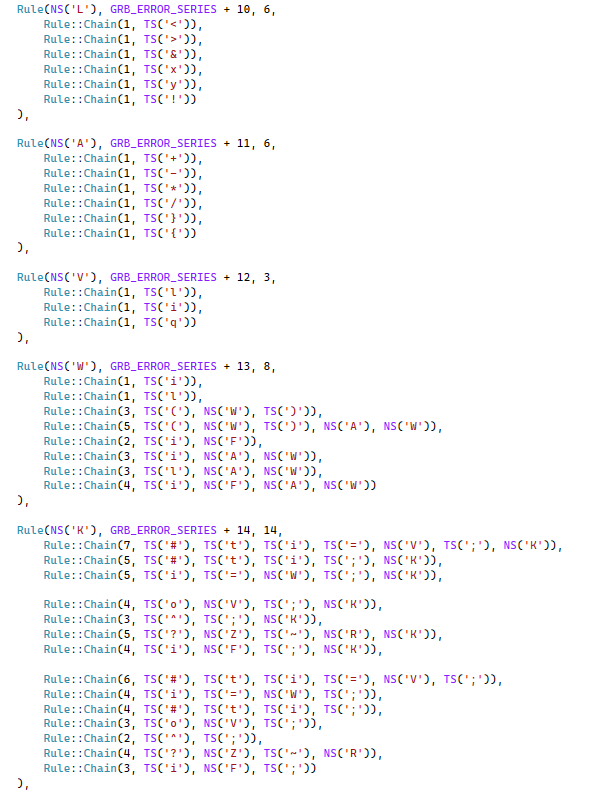
|  |
| --- |
| ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ТАБЛИЦА ЛЕКСЕМ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  | № | Лексема | ЛИНИЯ | ИНДЕКС В TI |  | 0 | t | 1 | |  | 1 | f | 1 | |  | 2 | i | 1 | 0 |  | 3 | ( | 1 | |  | 4 | t | 1 | |  | 5 | i | 1 | 1 |  | 6 | ) | 1 | |  | 7 | [ | 1 | |  | 8 | # | 2 | |  | 9 | t | 2 | |  | 10 | i | 2 | 2 |  | 11 | = | 2 | |  | 12 | l | 2 | 3 |  | 13 | ; | 2 | |  | 14 | # | 3 | |  | 15 | t | 3 | |  | 16 | i | 3 | 4 |  | 17 | = | 3 | |  | 18 | l | 3 | 3 |  | 19 | ; | 3 | |  | 20 | ? | 4 | |  | 21 | i | 4 | 1 |  | 22 | > | 4 | |  | 23 | i | 4 | 2 |  | 24 | ~ | 5 | |  | 25 | c | 6 | |  | 26 | [ | 6 | |  | 27 | i | 7 | 4 |  | 28 | = | 7 | |  | 29 | i | 7 | 4 |  | 30 | \* | 7 | |  | 31 | i | 7 | 2 |  | 32 | ; | 7 | |  | 33 | i | 8 | 2 |  | 34 | = | 8 | |  | 35 | i | 8 | 2 |  | 36 | + | 8 | |  | 37 | l | 8 | 3 |  | 38 | ; | 8 | |  | 39 | ] | 9 | |  | 40 | ~ | 10 | |  | 41 | i | 11 | 4 |  | 42 | = | 11 | |  | 43 | i | 11 | 4 |  | 44 | \* | 11 | |  | 45 | i | 11 | 1 |  | 46 | ; | 11 | |  | 47 | e | 12 | |  | 48 | i | 12 | 4 |  | 49 | ; | 12 | |  | 50 | ] | 13 | |  | 51 | p | 15 | |  | 52 | f | 15 | |  | 53 | i | 15 | 5 |  | 54 | ( | 15 | |  | 55 | t | 15 | |  | 56 | i | 15 | 6 |  | 57 | ) | 15 | |  | 58 | [ | 15 | |  | 59 | ^ | 16 | |  | 60 | ; | 16 | |  | 61 | o | 17 | |  | 62 | l | 17 | 7 |  | 63 | ; | 17 | |  | 64 | o | 18 | |  | 65 | i | 18 | 6 |  | 66 | ; | 18 | |  | 67 | e | 19 | |  | 68 | ; | 19 | |  | 69 | ] | 20 | |  | 70 | t | 22 | |  | 71 | f | 22 | |  | 72 | i | 22 | 8 |  | 73 | ( | 22 | |  | 74 | t | 22 | |  | 75 | i | 22 | 9 |  | 76 | , | 22 | |  | 77 | t | 22 | |  | 78 | i | 22 | 10 |  | 79 | ) | 22 | |  | 80 | [ | 22 | |  | 81 | # | 23 | |  | 82 | t | 23 | |  | 83 | i | 23 | 11 |  | 84 | ; | 23 | |  | 85 | ? | 24 | |  | 86 | i | 24 | 9 |  | 87 | > | 24 | |  | 88 | i | 24 | 10 |  | 89 | ~ | 25 | |  | 90 | w | 26 | |  | 91 | [ | 26 | |  | 92 | i | 26 | 11 |  | 93 | = | 26 | |  | 94 | i | 26 | 10 |  | 95 | ; | 26 | |  | 96 | ] | 26 | |  | 97 | r | 27 | |  | 98 | [ | 27 | |  | 99 | i | 27 | 11 |  |100 | = | 27 | |  |101 | i | 27 | 9 |  |102 | ; | 27 | |  |103 | ] | 27 | |  |104 | ~ | 28 | |  |105 | e | 29 | |  |106 | i | 29 | 11 |  |107 | ; | 29 | |  |108 | ] | 30 | |  |109 | m | 32 | |  |110 | [ | 32 | |  |111 | o | 33 | |  |112 | l | 33 | 12 |  |113 | ; | 33 | |  |114 | ^ | 34 | |  |115 | ; | 34 | |  |116 | o | 35 | |  |117 | l | 35 | 13 |  |118 | ; | 35 | |  |119 | ^ | 36 | |  |120 | ; | 36 | |  |121 | o | 37 | |  |122 | l | 37 | 12 |  |123 | ; | 37 | |  |124 | ^ | 38 | |  |125 | ; | 38 | |  |126 | # | 39 | |  |127 | t | 39 | |  |128 | i | 39 | 14 |  |129 | = | 39 | |  |130 | l | 39 | 15 |  |131 | ; | 39 | |  |132 | # | 40 | |  |133 | t | 40 | |  |134 | i | 40 | 16 |  |135 | = | 40 | |  |136 | l | 40 | 17 |  |137 | ; | 40 | |  |138 | # | 41 | |  |139 | t | 41 | |  |140 | i | 41 | 18 |  |141 | ; | 41 | |  |142 | o | 43 | |  |143 | l | 43 | 19 |  |144 | ; | 43 | |  |145 | o | 44 | |  |146 | i | 44 | 14 |  |147 | ; | 44 | |  |148 | ^ | 45 | |  |149 | ; | 45 | |  |150 | o | 46 | |  |151 | l | 46 | 20 |  |152 | ; | 46 | |  |153 | o | 47 | |  |154 | i | 47 | 16 |  |155 | ; | 47 | |  |156 | ^ | 48 | |  |157 | ; | 48 | |  |158 | i | 50 | 18 |  |159 | = | 50 | |  |160 | i | 50 | 14 |  |161 | + | 50 | |  |162 | i | 50 | 16 |  |163 | \* | 50 | |  |164 | ( | 50 | |  |165 | l | 50 | 21 |  |166 | - | 50 | |  |167 | l | 50 | 22 |  |168 | ) | 50 | |  |169 | / | 50 | |  |170 | l | 50 | 17 |  |171 | ; | 50 | |  |172 | o | 51 | |  |173 | l | 51 | 23 |  |174 | ; | 51 | |  |175 | o | 52 | |  |176 | i | 52 | 18 |  |177 | ; | 52 | |  |178 | i | 54 | 18 |  |179 | = | 54 | |  |180 | i | 54 | 18 |  |181 | + | 54 | |  |182 | i | 54 | 16 |  |183 | + | 54 | |  |184 | i | 54 | 24 |  |185 | ( | 54 | |  |186 | l | 54 | 25 |  |187 | , | 54 | |  |188 | l | 54 | 22 |  |189 | ) | 54 | |  |190 | ; | 54 | |  |191 | ^ | 55 | |  |192 | ; | 55 | |  |193 | o | 56 | |  |194 | l | 56 | 26 |  |195 | ; | 56 | |  |196 | o | 57 | |  |197 | i | 57 | 18 |  |198 | ; | 57 | |  |199 | ^ | 58 | |  |200 | ; | 58 | |  |201 | ^ | 59 | |  |202 | ; | 59 | |  |203 | # | 61 | |  |204 | t | 61 | |  |205 | i | 61 | 27 |  |206 | ; | 61 | |  |207 | o | 62 | |  |208 | l | 62 | 28 |  |209 | ; | 62 | |  |210 | i | 63 | 27 |  |211 | = | 63 | |  |212 | i | 63 | 8 |  |213 | ( | 63 | |  |214 | i | 63 | 14 |  |215 | , | 63 | |  |216 | i | 63 | 16 |  |217 | ) | 63 | |  |218 | ; | 63 | |  |219 | o | 64 | |  |220 | l | 64 | 29 |  |221 | ; | 64 | |  |222 | o | 65 | |  |223 | i | 65 | 27 |  |224 | ; | 65 | |  |225 | ^ | 66 | |  |226 | ; | 66 | |  |227 | ^ | 67 | |  |228 | ; | 67 | |  |229 | o | 69 | |  |230 | l | 69 | 30 |  |231 | ; | 69 | |  |232 | ^ | 70 | |  |233 | ; | 70 | |  |234 | o | 71 | |  |235 | l | 71 | 31 |  |236 | ; | 71 | |  |237 | ^ | 72 | |  |238 | ; | 72 | |  |239 | o | 73 | |  |240 | l | 73 | 30 |  |241 | ; | 73 | |  |242 | ^ | 74 | |  |243 | ; | 74 | |  |244 | # | 75 | |  |245 | t | 75 | |  |246 | i | 75 | 32 |  |247 | ; | 75 | |  |248 | i | 76 | 32 |  |249 | = | 76 | |  |250 | i | 76 | 33 |  |251 | ( | 76 | |  |252 | l | 76 | 34 |  |253 | ) | 76 | |  |254 | ; | 76 | |  |255 | o | 77 | |  |256 | l | 77 | 35 |  |257 | ; | 77 | |  |258 | o | 78 | |  |259 | i | 78 | 32 |  |260 | ; | 78 | |  |261 | ^ | 79 | |  |262 | ; | 79 | |  |263 | ^ | 80 | |  |264 | ; | 80 | |  |265 | # | 82 | |  |266 | t | 82 | |  |267 | i | 82 | 36 |  |268 | ; | 82 | |  |269 | i | 83 | 36 |  |270 | = | 83 | |  |271 | i | 83 | 37 |  |272 | ( | 83 | |  |273 | l | 83 | 38 |  |274 | ) | 83 | |  |275 | ; | 83 | |  |276 | o | 84 | |  |277 | l | 84 | 39 |  |278 | ; | 84 | |  |279 | i | 85 | 36 |  |280 | = | 85 | |  |281 | i | 85 | 36 |  |282 | / | 85 | |  |283 | l | 85 | 40 |  |284 | ; | 85 | |  |285 | o | 86 | |  |286 | i | 86 | 36 |  |287 | ; | 86 | |  |288 | ^ | 87 | |  |289 | ; | 87 | |  |290 | ^ | 88 | |  |291 | ; | 88 | |  |292 | i | 90 | 36 |  |293 | = | 90 | |  |294 | i | 90 | 24 |  |295 | ( | 90 | |  |296 | l | 90 | 25 |  |297 | , | 90 | |  |298 | l | 90 | 22 |  |299 | ) | 90 | |  |300 | ; | 90 | |  |301 | o | 91 | |  |302 | l | 91 | 41 |  |303 | ; | 91 | |  |304 | o | 92 | |  |305 | i | 92 | 36 |  |306 | ; | 92 | |  |307 | ^ | 93 | |  |308 | ; | 93 | |  |309 | ^ | 94 | |  |310 | ; | 94 | |  |311 | i | 96 | 36 |  |312 | = | 96 | |  |313 | i | 96 | 42 |  |314 | ( | 96 | |  |315 | l | 96 | 43 |  |316 | ) | 96 | |  |317 | ; | 96 | |  |318 | o | 97 | |  |319 | l | 97 | 44 |  |320 | ; | 97 | |  |321 | o | 98 | |  |322 | i | 98 | 36 |  |323 | ; | 98 | |  |324 | ^ | 99 | |  |325 | ; | 99 | |  |326 | ^ | 100 | |  |327 | ; | 100 | |  |328 | i | 102 | 36 |  |329 | = | 102 | |  |330 | i | 102 | 45 |  |331 | ( | 102 | |  |332 | l | 102 | 25 |  |333 | ) | 102 | |  |334 | ; | 102 | |  |335 | o | 103 | |  |336 | l | 103 | 46 |  |337 | ; | 103 | |  |338 | o | 104 | |  |339 | i | 104 | 36 |  |340 | ; | 104 | |  |341 | ^ | 105 | |  |342 | ; | 105 | |  |343 | ^ | 106 | |  |344 | ; | 106 | |  |345 | o | 108 | |  |346 | l | 108 | 47 |  |347 | ; | 108 | |  |348 | i | 109 | 36 |  |349 | = | 109 | |  |350 | i | 109 | 0 |  |351 | ( | 109 | |  |352 | l | 109 | 25 |  |353 | ) | 109 | |  |354 | ; | 109 | |  |355 | o | 110 | |  |356 | i | 110 | 36 |  |357 | ; | 110 | |  |358 | ^ | 111 | |  |359 | ; | 111 | |  |360 | ^ | 112 | |  |361 | ; | 112 | |  |362 | o | 114 | |  |363 | l | 114 | 48 |  |364 | ; | 114 | |  |365 | ^ | 115 | |  |366 | ; | 115 | |  |367 | o | 116 | |  |368 | l | 116 | 49 |  |369 | ; | 116 | |  |370 | ^ | 117 | |  |371 | ; | 117 | |  |372 | o | 118 | |  |373 | l | 118 | 48 |  |374 | ; | 118 | |  |375 | ^ | 119 | |  |376 | ; | 119 | |  |377 | # | 120 | |  |378 | t | 120 | |  |379 | i | 120 | 50 |  |380 | = | 120 | |  |381 | l | 120 | 51 |  |382 | ; | 120 | |  |383 | # | 121 | |  |384 | t | 121 | |  |385 | i | 121 | 52 |  |386 | = | 121 | |  |387 | l | 121 | 53 |  |388 | ; | 121 | |  |389 | # | 122 | |  |390 | t | 122 | |  |391 | i | 122 | 54 |  |392 | = | 122 | |  |393 | l | 122 | 55 |  |394 | ; | 122 | |  |395 | # | 123 | |  |396 | t | 123 | |  |397 | i | 123 | 56 |  |398 | = | 123 | |  |399 | l | 123 | 57 |  |400 | ; | 123 | |  |401 | o | 125 | |  |402 | l | 125 | 58 |  |403 | ; | 125 | |  |404 | o | 126 | |  |405 | i | 126 | 50 |  |406 | ; | 126 | |  |407 | ^ | 127 | |  |408 | ; | 127 | |  |409 | o | 128 | |  |410 | l | 128 | 59 |  |411 | ; | 128 | |  |412 | o | 129 | |  |413 | i | 129 | 52 |  |414 | ; | 129 | |  |415 | ^ | 130 | |  |416 | ; | 130 | |  |417 | o | 131 | |  |418 | l | 131 | 60 |  |419 | ; | 131 | |  |420 | o | 132 | |  |421 | i | 132 | 54 |  |422 | ; | 132 | |  |423 | ^ | 133 | |  |424 | ; | 133 | |  |425 | o | 134 | |  |426 | l | 134 | 61 |  |427 | ; | 134 | |  |428 | o | 135 | |  |429 | i | 135 | 56 |  |430 | ; | 135 | |  |431 | ^ | 136 | |  |432 | ; | 136 | |  |433 | ] | 137 | | |

# Приложение В

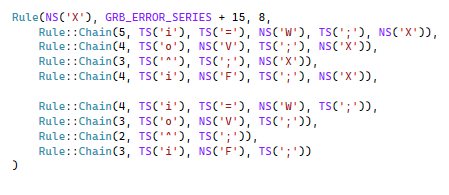
Листинг 1 Грамматика языка LVK-2024



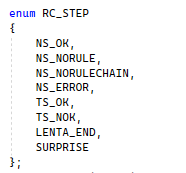
Листинг 1(продолжение) Грамматика языка LVK-2024

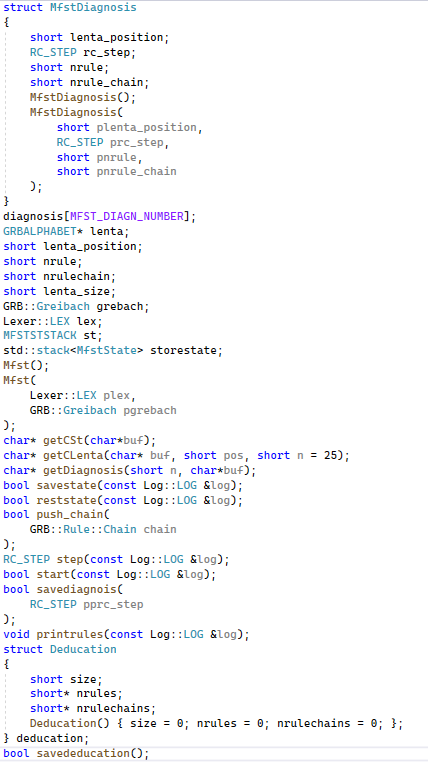


Листинг 1(продолжение) Грамматика языка LVK-2024

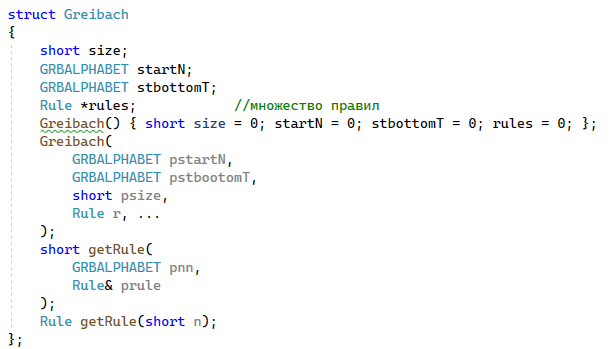


Листинг 2 Структура магазинного автомата





Листинг 3 Структура грамматики Грейбах



Листинг 4 Разбор исходного кода синтаксическим анализатором

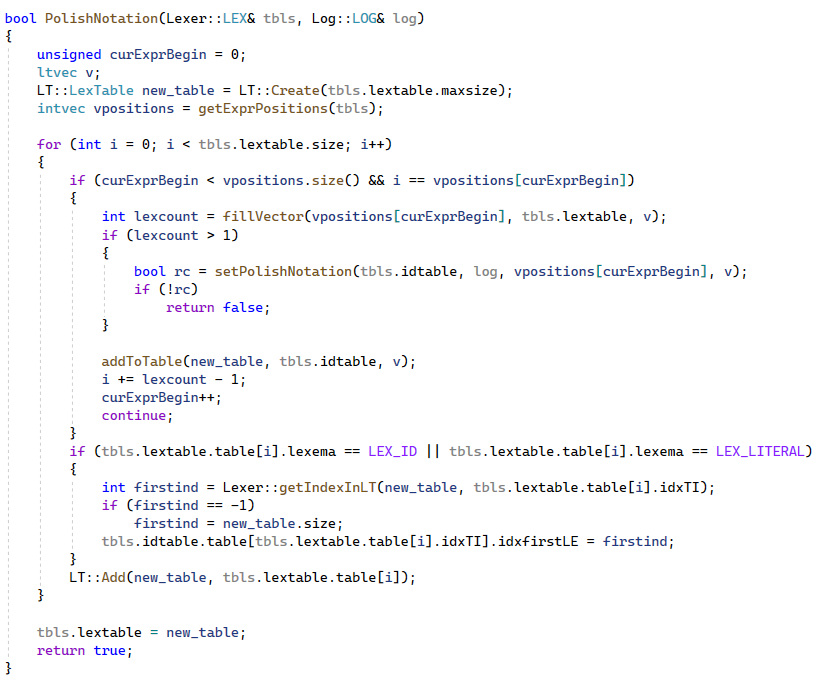
|  |
| --- |
| Шаг :Rule Entarance tape Stack  0 :S->tfiPTS tfi(ti)[#ti=l;#ti=l;?i>i~ S  1 : SAVESTATE: 1  1 : tfi(ti)[#ti=l;#ti=l;?i>i~ tfiPTS  2 : fi(ti)[#ti=l;#ti=l;?i>i~c fiPTS  3 : i(ti)[#ti=l;#ti=l;?i>i~c[ iPTS  4 : (ti)[#ti=l;#ti=l;?i>i~c[i PTS  5 :P->(E) (ti)[#ti=l;#ti=l;?i>i~c[i PTS  6 : SAVESTATE: 2  6 : (ti)[#ti=l;#ti=l;?i>i~c[i (E)TS  7 : ti)[#ti=l;#ti=l;?i>i~c[i= E)TS  8 :E->ti,E ti)[#ti=l;#ti=l;?i>i~c[i= E)TS  9 : SAVESTATE: 3  9 : ti)[#ti=l;#ti=l;?i>i~c[i= ti,E)TS  10 : i)[#ti=l;#ti=l;?i>i~c[i=i i,E)TS  11 : )[#ti=l;#ti=l;?i>i~c[i=i\* ,E)TS  12 : TS\_NOK/NS\_NORULECHАIN  12 : RESSTATE  12 : ti)[#ti=l;#ti=l;?i>i~c[i= E)TS  13 :E->ti ti)[#ti=l;#ti=l;?i>i~c[i= E)TS  14 : SAVESTATE: 3  14 : ti)[#ti=l;#ti=l;?i>i~c[i= ti)TS  15 : i)[#ti=l;#ti=l;?i>i~c[i=i i)TS  16 : )[#ti=l;#ti=l;?i>i~c[i=i\* )TS  17 : [#ti=l;#ti=l;?i>i~c[i=i\*i TS  18 :T->[eV;] [#ti=l;#ti=l;?i>i~c[i=i\*i TS  19 : SAVESTATE: 4  19 : [#ti=l;#ti=l;?i>i~c[i=i\*i [eV;]S  20 : #ti=l;#ti=l;?i>i~c[i=i\*i; eV;]S  21 : TS\_NOK/NS\_NORULECHАIN  21 : RESSTATE  21 : [#ti=l;#ti=l;?i>i~c[i=i\*i TS  22 :T->[KeV;] [#ti=l;#ti=l;?i>i~c[i=i\*i TS  23 : SAVESTATE: 4  23 : [#ti=l;#ti=l;?i>i~c[i=i\*i [KeV;]S  24 : #ti=l;#ti=l;?i>i~c[i=i\*i; KeV;]S  25 :K->#ti=V;K #ti=l;#ti=l;?i>i~c[i=i\*i; KeV;]S  26 : SAVESTATE: 5  26 : #ti=l;#ti=l;?i>i~c[i=i\*i; #ti=V;KeV;]S  27 : ti=l;#ti=l;?i>i~c[i=i\*i;i ti=V;KeV;]S  28 : i=l;#ti=l;?i>i~c[i=i\*i;i= i=V;KeV;]S  29 : =l;#ti=l;?i>i~c[i=i\*i;i=i =V;KeV;]S  30 : l;#ti=l;?i>i~c[i=i\*i;i=i+ V;KeV;]S |

Листинг 4 (прод.) Разбор исходного кода синтаксическим анализатором

|  |
| --- |
| 1485: SAVESTATE: 230  1485: ^;ol;oi;^;] ^;K]  1486: ;ol;oi;^;] ;K]  1487: ol;oi;^;] K]  1488:K->oV;K ol;oi;^;] K]  1489: SAVESTATE: 231  1489: ol;oi;^;] oV;K]  1490: l;oi;^;] V;K]  1491:V->l l;oi;^;] V;K]  1492: SAVESTATE: 232  1492: l;oi;^;] l;K]  1493: ;oi;^;] ;K]  1494: oi;^;] K]  1495:K->oV;K oi;^;] K]  1496: SAVESTATE: 233  1496: oi;^;] oV;K]  1497: i;^;] V;K]  1498:V->i i;^;] V;K]  1499: SAVESTATE: 234  1499: i;^;] i;K]  1500: ;^;] ;K]  1501: ^;] K]  1502:K->^;K ^;] K]  1503: SAVESTATE: 235  1503: ^;] ^;K]  1504: ;] ;K]  1505: ] K]  1506: TNS\_NORULECHAIN/NS\_NORULE  1506: RESSTATE  1506: ^;] K]  1507:K->^; ^;] K]  1508: SAVESTATE: 235  1508: ^;] ^;]  1509: ;] ;]  1510: ] ]  1511:  1512: LENTA\_END  1513: ------>LENTA\_END |

# Приложение Г

Листинг 1 Программная реализация механизма преобразования в ПОЛИЗ



Листинг 2 Таблица идентификаторов после преобразования выражений в ПОЛИЗ

|  |
| --- |
| | | № |Линия в LT | ID Тип | Название | Значение(аргументы)  |0 | 2 | number function | fact | P0:NUMBER |  |1 | 5 | number parameter | factnumber |  |2 | 10 | number variable | facti |0  |3 | 12 | number literal | LTRL1 |1  |4 | 16 | number variable | factres |0  |5 | 53 | proc function | myprint | P0:SYMBOL |  |6 | 56 | symbol parameter | myprinta |  |7 | 62 | symbol literal | LTRL2 |[9]myprint:  |8 | 72 | number function | mymin | P0:NUMBER | P1:NUMBER |  |9 | 75 | number parameter | mymina |  |10 | 78 | number parameter | myminb |  |11 | 83 | number variable | myminresult |0  |12 | 112 | symbol literal | LTRL3 |[29] -----------------------  |13 | 117 | symbol literal | LTRL4 |[29] test operation with num  |14 | 128 | number variable | roota |0  |15 | 130 | number literal | LTRL5 |4  |16 | 134 | number variable | rootb |0  |17 | 136 | number literal | LTRL6 |2  |18 | 140 | number variable | rootres |0  |19 | 143 | symbol literal | LTRL7 |[3]a=  |20 | 151 | symbol literal | LTRL8 |[3]b=  |21 | 162 | number literal | LTRL9 |100  |22 | 163 | number literal | LTRL10 |10  |23 | 171 | symbol literal | LTRL11 |[8]1) res=  |24 | 181 | number PIE LIBf | power | P0:NUMBER | P1:NUMBER |  |25 | 183 | number literal | LTRL12 |5  |26 | 192 | symbol literal | LTRL13 |[8]2) res=  |27 | 203 | number variable | rootmm |0  |28 | 206 | symbol literal | LTRL14 |[13]test My Min:  |29 | 218 | symbol literal | LTRL15 |[5]min:  |30 | 228 | symbol literal | LTRL16 |[39] ---------------------------------  |31 | 233 | symbol literal | LTRL17 |[39] test function from static library  |32 | 244 | number variable | rootle |0  |33 | 248 | number PIE LIBf | lenght | P0:SYMBOL |  |34 | 250 | symbol literal | LTRL18 |[5]hello  |35 | 254 | symbol literal | LTRL19 |[8]lenght:  |36 | 265 | number variable | roote |0  |37 | 269 | number PIE LIBf | atoii | P0:SYMBOL |  |38 | 271 | symbol literal | LTRL20 |[4]5125  |39 | 275 | symbol literal | LTRL21 |[7]atoii:  |40 | 280 | number literal | LTRL22 |6  |41 | 300 | symbol literal | LTRL23 |[7]power:  |42 | 311 | number PIE LIBf | fibonachi | P0:NUMBER |  |43 | 313 | number literal | LTRL24 |40  |44 | 317 | symbol literal | LTRL25 |[11]fibonachi:  |45 | 328 | number PIE LIBf | factorial | P0:NUMBER |  |46 | 334 | symbol literal | LTRL26 |[11]Factorial:  |47 | 344 | symbol literal | LTRL27 |[19]test My Factorial:  |48 | 361 | symbol literal | LTRL28 |[25] -------------------  |49 | 366 | symbol literal | LTRL29 |[25] test number systems  |50 | 377 | number variable | rootbin |0  |51 | 379 | number literal | LTRL30 |11  |52 | 383 | number variable | rootoct |0  |53 | 385 | number literal | LTRL31 |1004  |54 | 389 | number variable | rootdec |0  |55 | 391 | number literal | LTRL32 |12345  |56 | 395 | number variable | roothex |0  |57 | 397 | number literal | LTRL33 |8090  |58 | 400 | symbol literal | LTRL34 |[5]bin:  |59 | 408 | symbol literal | LTRL35 |[5]oct:  |60 | 416 | symbol literal | LTRL36 |[5]dec:  |61 | 424 | symbol literal | LTRL37 |[5]hex: |

Листинг 3 Таблица лексем после преобразования выражений в ПОЛИЗ

| № | Лексема | ЛИНИЯ | ИНДЕКС В TI |

|0 | t | 1 | |

|1 | f | 1 | |

|2 | i | 1 |0 |

|3 | ( | 1 | |

|4 | t | 1 | |

|5 | i | 1 |1 |

|6 | ) | 1 | |

|7 | [ | 1 | |

|8 | # | 2 | |

|9 | t | 2 | |

|10 | i | 2 |2 |

|11 | = | 2 | |

|12 | l | 2 |3 |

|13 | ; | 2 | |

|14 | # | 3 | |

|15 | t | 3 | |

|16 | i | 3 |4 |

|17 | = | 3 | |

|18 | l | 3 |3 |

|19 | ; | 3 | |

|20 | ? | 4 | |

|21 | i | 4 |1 |

|22 | > | 4 | |

|23 | i | 4 |2 |

|24 | ~ | 5 | |

|25 | c | 6 | |

|26 | [ | 6 | |

|27 | i | 7 |4 |

|28 | = | 7 | |

|29 | i | 7 |4 |

|30 | i | 7 |2 |

|31 | \* | 7 | |

|32 | ; | 7 | |

|33 | i | 8 |2 |

|34 | = | 8 | |

|35 | i | 8 |2 |

|36 | l | 8 |3 |

|37 | + | 8 | |

|38 | ; | 8 | |

|39 | ] | 9 | |

|40 | ~ | 10 | |

|41 | i | 11 |4 |

|42 | = | 11 | |

|43 | i | 11 |4 |

|44 | i | 11 |1 |

|45 | \* | 11 | |

|46 | ; | 11 | |

|47 | e | 12 | |

|48 | i | 12 |4 |

|49 | ; | 12 | |

|50 | ] | 13 | |

|51 | p | 15 | |

|52 | f | 15 | |

|53 | i | 15 |5 |

|54 | ( | 15 | |

|55 | t | 15 | |

|56 | i | 15 |6 |

|57 | ) | 15 | |

|58 | [ | 15 | |

|59 | ^ | 16 | |

|60 | ; | 16 | |

|61 | o | 17 | |

|62 | l | 17 |7 |

|63 | ; | 17 | |

|64 | o | 18 | |

|65 | i | 18 |6 |

|66 | ; | 18 | |

|67 | e | 19 | |

|68 | ; | 19 | |

|69 | ] | 20 | |

|70 | t | 22 | |

|71 | f | 22 | |

|72 | i | 22 |8 |

|73 | ( | 22 | |

|74 | t | 22 | |

|75 | i | 22 |9 |

|76 | , | 22 | |

|77 | t | 22 | |

|78 | i | 22 |10 |

|79 | ) | 22 | |

|80 | [ | 22 | |

|81 | # | 23 | |

|82 | t | 23 | |

|83 | i | 23 |11 |

|84 | ; | 23 | |

|85 | ? | 24 | |

|86 | i | 24 |9 |

|87 | > | 24 | |

|88 | i | 24 |10 |

|89 | ~ | 25 | |

|90 | w | 26 | |

|91 | [ | 26 | |

|92 | i | 26 |11 |

|93 | = | 26 | |

|94 | i | 26 |10 |

|95 | ; | 26 | |

|96 | ] | 26 | |

|97 | r | 27 | |

|98 | [ | 27 | |

|99 | i | 27 |11 |

|100 | = | 27 | |

|101 | i | 27 |9 |

|102 | ; | 27 | |

|103 | ] | 27 | |

|104 | ~ | 28 | |

|105 | e | 29 | |

|106 | i | 29 |11 |

|107 | ; | 29 | |

|108 | ] | 30 | |

|109 | m | 32 | |

|110 | [ | 32 | |

|111 | o | 33 | |

|112 | l | 33 |12 |

|113 | ; | 33 | |

|114 | ^ | 34 | |

|115 | ; | 34 | |

|116 | o | 35 | |

|117 | l | 35 |13 |

|118 | ; | 35 | |

|119 | ^ | 36 | |

|120 | ; | 36 | |

|121 | o | 37 | |

|122 | l | 37 |12 |

|123 | ; | 37 | |

|124 | ^ | 38 | |

|125 | ; | 38 | |

|126 | # | 39 | |

|127 | t | 39 | |

|128 | i | 39 |14 |

|129 | = | 39 | |

|130 | l | 39 |15 |

|131 | ; | 39 | |

|132 | # | 40 | |

|133 | t | 40 | |

|134 | i | 40 |16 |

|135 | = | 40 | |

|136 | l | 40 |17 |

|137 | ; | 40 | |

|138 | # | 41 | |

|139 | t | 41 | |

|140 | i | 41 |18 |

|141 | ; | 41 | |

|142 | o | 43 | |

|143 | l | 43 |19 |

|144 | ; | 43 | |

|145 | o | 44 | |

|146 | i | 44 |14 |

|147 | ; | 44 | |

|148 | ^ | 45 | |

|149 | ; | 45 | |

|150 | o | 46 | |

|151 | l | 46 |20 |

|152 | ; | 46 | |

|153 | o | 47 | |

|154 | i | 47 |16 |

|155 | ; | 47 | |

|156 | ^ | 48 | |

|157 | ; | 48 | |

|158 | i | 50 |18 |

|159 | = | 50 | |

|160 | i | 50 |14 |

|161 | i | 50 |16 |

|162 | l | 50 |21 |

|163 | l | 50 |22 |

|164 | - | 50 | |

|165 | \* | 50 | |

|166 | l | 50 |17 |

|167 | / | 50 | |

|168 | + | 50 | |

|169 | ; | 50 | |

|170 | o | 51 | |

|171 | l | 51 |23 |

|172 | ; | 51 | |

|173 | o | 52 | |

|174 | i | 52 |18 |

|175 | ; | 52 | |

|176 | i | 54 |18 |

|177 | = | 54 | |

|178 | i | 54 |18 |

|179 | i | 54 |16 |

|180 | + | 54 | |

|181 | i | 54 |24 |

|182 | ( | 54 | |

|183 | l | 54 |25 |

|184 | , | 54 | |

|185 | l | 54 |22 |

|186 | ) | 54 | |

|187 | + | 54 | |

|188 | ; | 54 | |

|189 | ^ | 55 | |

|190 | ; | 55 | |

|191 | o | 56 | |

|192 | l | 56 |26 |

|193 | ; | 56 | |

|194 | o | 57 | |

|195 | i | 57 |18 |

|196 | ; | 57 | |

|197 | ^ | 58 | |

|198 | ; | 58 | |

|199 | ^ | 59 | |

|200 | ; | 59 | |

|201 | # | 61 | |

|202 | t | 61 | |

|203 | i | 61 |27 |

|204 | ; | 61 | |

|205 | o | 62 | |

|206 | l | 62 |28 |

|207 | ; | 62 | |

|208 | i | 63 |27 |

|209 | = | 63 | |

|210 | i | 63 |8 |

|211 | ( | 63 | |

|212 | i | 63 |14 |

|213 | , | 63 | |

|214 | i | 63 |16 |

|215 | ) | 63 | |

|216 | ; | 63 | |

|217 | o | 64 | |

|218 | l | 64 |29 |

|219 | ; | 64 | |

|220 | o | 65 | |

|221 | i | 65 |27 |

|222 | ; | 65 | |

|223 | ^ | 66 | |

|224 | ; | 66 | |

|225 | ^ | 67 | |

|226 | ; | 67 | |

|227 | o | 69 | |

|228 | l | 69 |30 |

|229 | ; | 69 | |

|230 | ^ | 70 | |

|231 | ; | 70 | |

|232 | o | 71 | |

|233 | l | 71 |31 |

|234 | ; | 71 | |

|235 | ^ | 72 | |

|236 | ; | 72 | |

|237 | o | 73 | |

|238 | l | 73 |30 |

|239 | ; | 73 | |

|240 | ^ | 74 | |

|241 | ; | 74 | |

|242 | # | 75 | |

|243 | t | 75 | |

|244 | i | 75 |32 |

|245 | ; | 75 | |

|246 | i | 76 |32 |

|247 | = | 76 | |

|248 | i | 76 |33 |

|249 | ( | 76 | |

|250 | l | 76 |34 |

|251 | ) | 76 | |

|252 | ; | 76 | |

|253 | o | 77 | |

|254 | l | 77 |35 |

|255 | ; | 77 | |

|256 | o | 78 | |

|257 | i | 78 |32 |

|258 | ; | 78 | |

|259 | ^ | 79 | |

|260 | ; | 79 | |

|261 | ^ | 80 | |

|262 | ; | 80 | |

|263 | # | 82 | |

|264 | t | 82 | |

|265 | i | 82 |36 |

|266 | ; | 82 | |

|267 | i | 83 |36 |

|268 | = | 83 | |

|269 | i | 83 |37 |

|270 | ( | 83 | |

|271 | l | 83 |38 |

|272 | ) | 83 | |

|273 | ; | 83 | |

|274 | o | 84 | |

|275 | l | 84 |39 |

|276 | ; | 84 | |

|277 | i | 85 |36 |

|278 | = | 85 | |

|279 | i | 85 |36 |

|280 | l | 85 |40 |

|281 | / | 85 | |

|282 | ; | 85 | |

|283 | o | 86 | |

|284 | i | 86 |36 |

|285 | ; | 86 | |

|286 | ^ | 87 | |

|287 | ; | 87 | |

|288 | ^ | 88 | |

|289 | ; | 88 | |

|290 | i | 90 |36 |

|291 | = | 90 | |

|292 | i | 90 |24 |

|293 | ( | 90 | |

|294 | l | 90 |25 |

|295 | , | 90 | |

|296 | l | 90 |22 |

|297 | ) | 90 | |

|298 | ; | 90 | |

|299 | o | 91 | |

|300 | l | 91 |41 |

|301 | ; | 91 | |

|302 | o | 92 | |

|303 | i | 92 |36 |

|304 | ; | 92 | |

|305 | ^ | 93 | |

|306 | ; | 93 | |

|307 | ^ | 94 | |

|308 | ; | 94 | |

|309 | i | 96 |36 |

|310 | = | 96 | |

|311 | i | 96 |42 |

|312 | ( | 96 | |

|313 | l | 96 |43 |

|314 | ) | 96 | |

|315 | ; | 96 | |

|316 | o | 97 | |

|317 | l | 97 |44 |

|318 | ; | 97 | |

|319 | o | 98 | |

|320 | i | 98 |36 |

|321 | ; | 98 | |

|322 | ^ | 99 | |

|323 | ; | 99 | |

|324 | ^ | 100 | |

|325 | ; | 100 | |

|326 | i | 102 |36 |

|327 | = | 102 | |

|328 | i | 102 |45 |

|329 | ( | 102 | |

|330 | l | 102 |25 |

|331 | ) | 102 | |

|332 | ; | 102 | |

|333 | o | 103 | |

|334 | l | 103 |46 |

|335 | ; | 103 | |

|336 | o | 104 | |

|337 | i | 104 |36 |

|338 | ; | 104 | |

|339 | ^ | 105 | |

|340 | ; | 105 | |

|341 | ^ | 106 | |

|342 | ; | 106 | |

|343 | o | 108 | |

|344 | l | 108 |47 |

|345 | ; | 108 | |

|346 | i | 109 |36 |

|347 | = | 109 | |

|348 | i | 109 |0 |

|349 | ( | 109 | |

|350 | l | 109 |25 |

|351 | ) | 109 | |

|352 | ; | 109 | |

|353 | o | 110 | |

|354 | i | 110 |36 |

|355 | ; | 110 | |

|356 | ^ | 111 | |

|357 | ; | 111 | |

|358 | ^ | 112 | |

|359 | ; | 112 | |

|360 | o | 114 | |

|361 | l | 114 |48 |

|362 | ; | 114 | |

|363 | ^ | 115 | |

|364 | ; | 115 | |

|365 | o | 116 | |

|366 | l | 116 |49 |

|367 | ; | 116 | |

|368 | ^ | 117 | |

|369 | ; | 117 | |

|370 | o | 118 | |

|371 | l | 118 |48 |

|372 | ; | 118 | |

|373 | ^ | 119 | |

|374 | ; | 119 | |

|375 | # | 120 | |

|376 | t | 120 | |

|377 | i | 120 |50 |

|378 | = | 120 | |

|379 | l | 120 |51 |

|380 | ; | 120 | |

|381 | # | 121 | |

|382 | t | 121 | |

|383 | i | 121 |52 |

|384 | = | 121 | |

|385 | l | 121 |53 |

|386 | ; | 121 | |

|387 | # | 122 | |

|388 | t | 122 | |

|389 | i | 122 |54 |

|390 | = | 122 | |

|391 | l | 122 |55 |

|392 | ; | 122 | |

|393 | # | 123 | |

|394 | t | 123 | |

|395 | i | 123 |56 |

|396 | = | 123 | |

|397 | l | 123 |57 |

|398 | ; | 123 | |

|399 | o | 125 | |

|400 | l | 125 |58 |

|401 | ; | 125 | |

|402 | o | 126 | |

|403 | i | 126 |50 |

|404 | ; | 126 | |

|405 | ^ | 127 | |

|406 | ; | 127 | |

|407 | o | 128 | |

|408 | l | 128 |59 |

|409 | ; | 128 | |

|410 | o | 129 | |

|411 | i | 129 |52 |

|412 | ; | 129 | |

|413 | ^ | 130 | |

|414 | ; | 130 | |

|415 | o | 131 | |

|416 | l | 131 |60 |

|417 | ; | 131 | |

|418 | o | 132 | |

|419 | i | 132 |54 |

|420 | ; | 132 | |

|421 | ^ | 133 | |

|422 | ; | 133 | |

|423 | o | 134 | |

|424 | l | 134 |61 |

|425 | ; | 134 | |

|426 | o | 135 | |

|427 | i | 135 |56 |

|428 | ; | 135 | |

|429 | ^ | 136 | |

|430 | ; | 136 | |

|431 | ] | 137 | |

# Приложение Д

Листинг 1 Результат генерации кода контрольного примера в Ассемблере

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ..\Debug\StaticLibrary.lib  ExitProcess PROTO:DWORD  .stack 4096  outnum PROTO : DWORD  outstr PROTO : DWORD  lenght PROTO : DWORD  atoii PROTO : DWORD, : DWORD  fibonachi PROTO : DWORD  factorial PROTO : DWORD  power PROTO : DWORD, : DWORD  .const  newline byte 13, 10, 0  LTRL1 dword 1  LTRL2 byte 'myprint: ', 0  LTRL3 byte ' -----------------------', 0  LTRL4 byte ' test operation with num', 0  LTRL5 dword 4  LTRL6 dword 2  LTRL7 byte 'a= ', 0  LTRL8 byte 'b= ', 0  LTRL9 dword 100  LTRL10 dword 10  LTRL11 byte '1) res= ', 0  LTRL12 dword 5  LTRL13 byte '2) res= ', 0  LTRL14 byte 'test My Min: ', 0  LTRL15 byte 'min: ', 0  LTRL16 byte ' ---------------------------------', 0  LTRL17 byte ' test function from static library', 0  LTRL18 byte 'hello', 0  LTRL19 byte 'lenght: ', 0  LTRL20 byte '5125', 0  LTRL21 byte 'atoii: ', 0  LTRL22 dword 6  LTRL23 byte 'power: ', 0  LTRL24 dword 40  LTRL25 byte 'fibonachi: ', 0  LTRL26 byte 'Factorial: ', 0  LTRL27 byte 'test My Factorial: ', 0  LTRL28 byte ' -------------------', 0  LTRL29 byte ' test number systems', 0  LTRL30 dword 11  LTRL31 dword 1004  LTRL32 dword 12345  LTRL33 dword 8090  LTRL34 byte 'bin: ', 0  LTRL35 byte 'oct: ', 0  LTRL36 byte 'dec: ', 0  LTRL37 byte 'hex: ', 0  .data  temp sdword ?  buffer byte 256 dup(0)  ERROR\_MINUS byte 'ERROR 404: LEAVE THE RANGE',0  ERROR\_DIVIDE byte 'ERROR 303: DIVIDE BY ZERO',0  facti dword 0  factres dword 0  myminresult dword 0  roota dword 0  rootb dword 0  rootres dword 0  rootmm dword 0  rootle dword 0  roote dword 0  rootbin dword 0  rootoct dword 0  rootdec dword 0  roothex dword 0  .code  ;----------- fact ------------  \_fact PROC,  factnumber : dword  ; ~~~ сохранение регистров ~~~  push ebx  push edx  ; ~~~~~~~~~~~~~~~~~~~~~~  push LTRL1  pop ebx  mov facti, ebx  push LTRL1  pop ebx  mov factres, ebx  mov edx, factnumber  cmp edx, facti  jg cycle1  jmp cyclenext1  cycle1:  push factres  push facti  pop ebx  pop eax  imul eax, ebx  push eax  pop ebx  mov factres, ebx  push facti  push LTRL1  pop ebx  pop eax  add eax, ebx  push eax  pop ebx  mov facti, ebx  mov edx, factnumber  cmp edx, facti  jg cycle1  cyclenext1:  push factres  push factnumber  pop ebx  pop eax  imul eax, ebx  push eax  pop ebx  mov factres, ebx  ; ~~~ восстановление регистров ~~~  pop edx  pop ebx  ; ~~~~~~~~~~~~~~~~~~~~~~~~  mov eax, factres  ret  \_fact ENDP  ;------------------------------  ;----------- myprint ------------  \_myprint PROC,  myprinta : dword  ; ~~~ сохранение регистров ~~~  push ebx  push edx  ; ~~~~~~~~~~~~~~~~~~~~~~  push offset newline  call outstr  push offset LTRL2  call outstr  push myprinta  call outstr  ; ~~~ восстановление регистров ~~~  pop edx  pop ebx  ; ~~~~~~~~~~~~~~~~~~~~~~~~  ret  \_myprint ENDP  ;------------------------------  ;----------- mymin ------------  \_mymin PROC,  mymina : dword, myminb : dword  ; ~~~ сохранение регистров ~~~  push ebx  push edx  ; ~~~~~~~~~~~~~~~~~~~~~~  mov edx, mymina  cmp edx, myminb  jg right2  jl wrong2  right2:  push myminb  pop ebx  mov myminresult, ebx  jmp next2  wrong2:  push mymina  pop ebx  mov myminresult, ebx  next2:  ; ~~~ восстановление регистров ~~~  pop edx  pop ebx  ; ~~~~~~~~~~~~~~~~~~~~~~~~  mov eax, myminresult  ret  \_mymin ENDP  ;------------------------------  ;----------- MAIN ------------  main PROC  push offset LTRL3  call outstr  push offset newline  call outstr  push offset LTRL4  call outstr  push offset newline  call outstr  push offset LTRL3  call outstr  push offset newline  call outstr  push LTRL5  pop ebx  mov roota, ebx  push LTRL6  pop ebx  mov rootb, ebx  push offset LTRL7  call outstr  push roota  call outnum  push offset newline  call outstr  push offset LTRL8  call outstr  push rootb  call outnum  push offset newline  call outstr  push roota  push rootb  push LTRL9  push LTRL10  pop ebx  pop eax  cmp ebx, eax  jle positive\_value\_164  push offset ERROR\_MINUS  call outstr  INVOKE ExitProcess,0  positive\_value\_164:  sub eax, ebx  push eax  pop ebx  pop eax  imul eax, ebx  push eax  push LTRL6  pop ebx  pop eax  cmp ebx, 0  jne division\_by\_zero\_error\_167  push offset ERROR\_DIVIDE  call outstr  INVOKE ExitProcess,0  division\_by\_zero\_error\_167:  cdq  idiv ebx  push eax  pop ebx  pop eax  add eax, ebx  push eax  pop ebx  mov rootres, ebx  push offset LTRL11  call outstr  push rootres  call outnum  push rootres  push rootb  pop ebx  pop eax  add eax, ebx  push eax  push LTRL10  push LTRL12  call power  push eax  pop ebx  pop eax  add eax, ebx  push eax  pop ebx  mov rootres, ebx  push offset newline  call outstr  push offset LTRL13  call outstr  push rootres  call outnum  push offset newline  call outstr  push offset newline  call outstr  push offset LTRL14  call outstr  push rootb  push roota  call \_mymin  push eax  pop ebx  mov rootmm, ebx  push offset LTRL15  call outstr  push rootmm  call outnum  push offset newline  call outstr  push offset newline  call outstr  push offset LTRL16  call outstr  push offset newline  call outstr  push offset LTRL17  call outstr  push offset newline  call outstr  push offset LTRL16  call outstr  push offset newline  call outstr  push offset LTRL18  call lenght  push eax  pop ebx  mov rootle, ebx  push offset LTRL19  call outstr  push rootle  call outnum  push offset newline  call outstr  push offset newline  call outstr  push offset LTRL20  push offset buffer  call atoii  push eax  pop ebx  mov roote, ebx  push offset LTRL21  call outstr  push roote  push LTRL22  pop ebx  pop eax  cmp ebx, 0  jne division\_by\_zero\_error\_281  push offset ERROR\_DIVIDE  call outstr  INVOKE ExitProcess,0  division\_by\_zero\_error\_281:  cdq  idiv ebx  push eax  pop ebx  mov roote, ebx  push roote  call outnum  push offset newline  call outstr  push offset newline  call outstr  push LTRL10  push LTRL12  call power  push eax  pop ebx  mov roote, ebx  push offset LTRL23  call outstr  push roote  call outnum  push offset newline  call outstr  push offset newline  call outstr  push LTRL24  call fibonachi  push eax  pop ebx  mov roote, ebx  push offset LTRL25  call outstr  push roote  call outnum  push offset newline  call outstr  push offset newline  call outstr  push LTRL12  call factorial  push eax  pop ebx  mov roote, ebx  push offset LTRL26  call outstr  push roote  call outnum  push offset newline  call outstr  push offset newline  call outstr  push offset LTRL27  call outstr  push LTRL12  call \_fact  push eax  pop ebx  mov roote, ebx  push roote  call outnum  push offset newline  call outstr  push offset newline  call outstr  push offset LTRL28  call outstr  push offset newline  call outstr  push offset LTRL29  call outstr  push offset newline  call outstr  push offset LTRL28  call outstr  push offset newline  call outstr  push LTRL30  pop ebx  mov rootbin, ebx  push LTRL31  pop ebx  mov rootoct, ebx  push LTRL32  pop ebx  mov rootdec, ebx  push LTRL33  pop ebx  mov roothex, ebx  push offset LTRL34  call outstr  push rootbin  call outnum  push offset newline  call outstr  push offset LTRL35  call outstr  push rootoct  call outnum  push offset newline  call outstr  push offset LTRL36  call outstr  push rootdec  call outnum  push offset newline  call outstr  push offset LTRL37  call outstr  push roothex  call outnum  push offset newline  call outstr  push 0  call ExitProcess  main ENDP  end main |