МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора DML-2021»

Выполнил студент Дащинский Максим Леонидович

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

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

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

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

Консультанты ст.пр. Наркевич Аделина Сергеевна

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

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

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2021

Оглавление

[**Введение** 4](#_Toc91015113)

[**Глава 1. Спецификация языка программирования** 5](#_Toc91015114)

[**1.1** **Характеристика языка программирования** 5](#_Toc91015115)

[**1.2** **Определение алфавита языка программирования** 5](#_Toc91015116)

[**1.3** **Применяемые сепараторы** 5](#_Toc91015117)

[**1.4** **Применяемые кодировки** 5](#_Toc91015118)

[**1.5** **Типы данных** 6](#_Toc91015119)

[**1.6** **Преобразование типов данных** 6](#_Toc91015120)

[**1.7** **Идентификаторы** 6](#_Toc91015121)

[**1.8** **Литералы** 7](#_Toc91015122)

[**1.9** **Объявление данных** 7](#_Toc91015123)

[**1.10** **Инициализация данных** 7](#_Toc91015124)

[**1.11** **Инструкции языка** 8](#_Toc91015125)

[**1.12** **Операции языка** 8](#_Toc91015126)

[**1.13** **Выражения и их вычисления** 9](#_Toc91015127)

[**1.14** **Программные конструкции языка** 9](#_Toc91015128)

[**1.15** **Область видимости идентификаторов** 9](#_Toc91015129)

[**1.16** **Семантические проверки** 10](#_Toc91015130)

[**1.17** **Распределение оперативной памяти на этапе выполнения** 10](#_Toc91015131)

[**1.18** **Стандартная библиотека и её состав** 10](#_Toc91015132)

[**1.19** **Ввод и вывод данных** 11](#_Toc91015133)

[**1.20** **Точка входа** 11](#_Toc91015134)

[**1.21** **Препроцессор** 11](#_Toc91015135)

[**1.22** **Соглашения о вызовах** 11](#_Toc91015136)

[**1.23** **Объектный код** 11](#_Toc91015137)

[**1.24** **Классификация сообщений транслятора** 12](#_Toc91015138)

[**1.25** **Контрольный пример** 12](#_Toc91015139)

[**Глава 2. Структура транслятора** 12](#_Toc91015140)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 12](#_Toc91015141)

[**2.2 Перечень входных параметров транслятора** 13](#_Toc91015142)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 14](#_Toc91015143)

[**Глава 3. Разработка лексического анализатора** 14](#_Toc91015144)

[**3.1 Структура лексического анализатора** 14](#_Toc91015145)

[**3.2. Контроль входных символов** 15](#_Toc91015146)

[**3.3 Удаление избыточных символов** 16](#_Toc91015147)

[**3.4 Перечень ключевых слов** 16](#_Toc91015148)

[**3.5 Основные структуры данных** 17](#_Toc91015149)

[**3.6 Принцип обработки ошибок** 19](#_Toc91015150)

[**3.7 Структура и перечень сообщений лексического анализа** 19](#_Toc91015151)

[**3.8 Параметры лексического анализатора** 19](#_Toc91015152)

[**3.9 Алгоритм лексического анализа** 19](#_Toc91015153)

[**3.10 Контрольный пример** 20](#_Toc91015154)

[**Глава 4. Разработка синтаксического анализатора** 20](#_Toc91015155)

[**4.1 Структура синтаксического анализатора** 20](#_Toc91015156)

[**4.2 Контекстно-свободная грамматика, описывающая синтаксис** 20](#_Toc91015157)

[**4.3 Построение конечного магазинного автомата** 23](#_Toc91015158)

[**4.4 Основные структуры данных** 23](#_Toc91015159)

[**4.5 Описание алгоритма синтаксического разбора** 24](#_Toc91015160)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 24](#_Toc91015161)

[**4.7. Параметры синтаксического анализатора и режимы его работы** 24](#_Toc91015162)

[**4.8. Принцип обработки ошибок** 25](#_Toc91015163)

[**4.9. Контрольный пример** 25](#_Toc91015164)

[**Глава 5. Разработка семантического анализатора** 25](#_Toc91015165)

[**5.1 Структура семантического анализатора** 25](#_Toc91015166)

[**5.2 Функции семантического анализатора** 26](#_Toc91015167)

[**5.3 Структура и перечень семантических ошибок** 26](#_Toc91015168)

[**5.4 Принцип обработки ошибок** 26](#_Toc91015169)

[**5.5 Контрольный пример** 26](#_Toc91015170)

[**Глава 6. Вычисление выражений** 27](#_Toc91015171)

[**6.1 Выражения, допускаемые языком** 27](#_Toc91015172)

[**6.2 Польская запись и принцип ее построения** 28](#_Toc91015173)

[**6.3 Программная реализация обработки выражений** 28](#_Toc91015174)

[**6.4 Контрольный пример** 29](#_Toc91015175)

[**Глава 7. Генерация кода** 29](#_Toc91015176)

[**7.1 Структура генератора кода** 29](#_Toc91015177)

[**7.2 Представление типов данных в оперативной памяти** 29](#_Toc91015178)

[**7.3 Статическая библиотека** 30](#_Toc91015179)

[**7.4 Особенности алгоритма генерации кода** 30](#_Toc91015180)

[**7.5 Входные параметры генератора кода** 31](#_Toc91015181)

[**7.6 Контрольный пример** 31](#_Toc91015182)

[**Глава 8. Тестирование транслятора** 31](#_Toc91015183)

[**8.1 Тестирование фазы проверки на допустимость символов** 31](#_Toc91015184)

[**8.2 Тестирование лексического анализатора** 31](#_Toc91015185)

[**8.3 Тестирование синтаксического анализатора** 32](#_Toc91015186)

[**Заключение** 33](#_Toc91015187)

[**Список использованных источников** 34](#_Toc91015188)

[**Приложение А** 35](#_Toc91015189)

[**Приложение Б** 36](#_Toc91015190)

[**Приложение В** 41](#_Toc91015191)

[**Приложение Г** 44](#_Toc91015194)

[**Приложение Д** 46](#_Toc91015196)

# **Введение**

В данном курсовом проекте поставлена задача разработки собственного языка программирования и транслятора для него. Название языка – DML-2021. Написание транслятора будет осуществляться на языке C++, при этом код на языке DML-2021 будет транслироваться в язык ассемблера. Язык ассемблера – это машинно-ориентированный язык, представляющий формат записи машинных команд, которые понятны для восприятия человеком.

Задание на курсовой проект можно разделить на следующие задачи:

1. разработка спецификации языка программирования;
2. разработка структуры транслятора;
3. разработка лексического анализатора;
4. разработка синтаксического анализатора;
5. разработка семантического анализатора;
6. обработка выражений с помощью польской инверсии;
7. генерация кода на язык ассемблера;
8. тестирование транслятора;

# **Глава 1. Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

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

## **1.2 Определение алфавита языка программирования**

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

## **1.3 Применяемые сепараторы**

Символы-сепараторы необходимы для разделения операция языка. Сепараторы, используемые в языке программирования DML-2021, приведены в таблице 1.1.

Таблица 1.1 Символы-сепараторы

|  |  |
| --- | --- |
| Символ(ы) | Назначение |
| ‘пробел’ | Разделитель цепочек. Допускается везде кроме названий идентификаторов и ключевых слов |
| ( ) | Параметры операций и функций |
| { } | Программный блок инструкций |
| , | Разделитель параметров функций |
| + - \*/ | Арифметические операции |
| > < ^ \_ & ! | Логические операции (операции сравнения: больше, меньше, больше или равно, меньше или равно, проверка на равенство, неравенство), используемые в условии цикла/условной конструкции. |
| ; | Разделитель программных конструкций |
| = | Оператор присваивания |

## **1.4 Применяемые кодировки**

Для написания программ язык DML-2021 использует кодировку Windows-1251, содержащую английский алфавит, а также некоторые специальные символы, такие как [ ] ( ) , ; : + - / \* > < = !{}.

## **1.5 Типы данных**

В языке программирования DML-2021 используются два основных типа данных, которые описываются в таблице 1.2. Пользовательские типы данных не поддерживаются.

Таблица 1.2 Типы данных

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Целочисленный тип данных uint | Фундаментальный тип данных, используемый для объявления целочисленных данных. Этот тип данных занимает 4 байта. Без явно указанной инициализации переменной, присваивается нулевое значение. Представляет только положительное целое число.  Максимальное значение: 4294967295.  Минимальное значение: 0. |
| Строковый тип данных str | Фундаментальный тип данных, используемый для объявления строк. Без явно указанной инициализации переменной, присваивается нулевое значение (пустая строка). Используется для работы с символами, каждый из которых занимает 1 байт.  Максимальное количество символов – 255. |

## **1.6 Преобразование типов данных**

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

## **1.7 Идентификаторы**

Общее количество идентификаторов ограниченно максимальным размером таблицы идентификаторов (4096). Идентификаторы могут содержать символы как нижнего регистра, так и верхнего. Максимальная длина идентификатора равна 10 символам. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Данные правила действуют для всех идентификаторов. Зарезервированные идентификаторы не предусмотрены. Идентификаторы не должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, имя функции, параметр функции. Имя идентификатора составляется по следующим образом:

* может состоять из цифр, но не может начинаться с цифры;
* состоит из символов латинского алфавита;
* максимальная длина идентификатора равна 10 и не должна превышать это значение. При превышении максимально значения длина идентификатора усекается до 10.

## **1.8 Литералы**

В языке программирования DML-2021 существует только 2 типа литералов: целые и символьные. Все литералы являются **rvalue**. Целочисленные литералы могут быть представлены в виде десятичного и восьмеричного представления, а строковые литералы – произвольно. Их краткое описание представлено в таблице 1.3.

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| Литералы | Пояснение |
| Целочисленные литералы в десятичном представлении | Последовательность цифр 0…9 без знака минус |
| Целочисленные литералы в восьмеричном представлении | Последовательность цифр 0…7 с предшествующим двумя символом “8x” |
| Строковые литералы | Набор символов алфавита языка, заключенных в одинарные кавычки |

Ограничения на целочисленные литералы: не могут начинаться с 0, если их значение не 0.

## **1.9 Объявление данных**

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

Пример объявления числового типа с инициализацией:

**var uint** num10 = 100

**var** **uint** num8 = 8x15

Пример объявления переменной символьного типа с инициализацией:

**var str** strhello = “Привет”

**var str** string = “car - vehicle”

Для объявления функций используется ключевое слово **func**, перед которым указывается тип функции. Далее обязателен список параметров и тело функции. Все функции должны возвращать значение.

## **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 и литералы. При объявлении переменные инициализируются значением по умолчанию: для строк это пустая строка, а для беззнакового целого это нуль. Способы инициализации переменных языка программирования DML-2021 представлены в таблице 1.4.

Таблица 1.4 Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| var <тип данных> <идентификатор>; | Автоматическая инициализация переменной. uint – инициализируется нулем, str – пустой строкой. |
| var <тип данных> <идентификатор> = <значение>; | Инициализация переменной с присваиванием значения. |

## **1.11 Инструкции языка**

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

Таблица 1.5 Инструкции языка программирования DML-2021

|  |  |
| --- | --- |
| Инструкция | Запись |
| Объявление переменной | var <тип данных> <идентификатор>;  var <тип данных> <идентификатор> = <значение>; |
| Присваивание | <идентификатор> = <значение>|<идентификатор>; |
| Объявление функции | func <тип данных> <идентификатор> ([<тип данных> <идентификатор>][, <тип данных> <идентификатор>]\*) {…} |
| Блок инструкций | {  …  } |
| Возврат из подпрограммы | ret <выражение>; |
| Вывод данных | print <идентификатор>|<литерал>; |
| Условный оператор с блоком else | if [<условие>]  { … }  else  { … } |
| Условный оператор | if [<условие>]  { … } |

## **1.12 Операции языка**

Приоритетность операции умножения выше приоритета операций вычитания и сложения. Для установки наивысшего приоритета используются круглые скобки. К строкам нельзя применять арифметические операции. Язык программирования DML-2021 может выполнять арифметические и операции сравнения, представленные в таблице 1.6.

Таблица 1.6 Операции и их приоритеты

|  |  |
| --- | --- |
| Операция | Приоритет операции |
| (  )  >  <  &  ^  \_  ! | 0 |
| , | 1 |
| +  - | 2 |
| \*  / | 3 |

## **1.13 Выражения и их вычисления**

Круглые скобки в выражении используются для изменения приоритетов операций. Не допускается запись двух подряд арифметических операций. Также круглые скобки могут использоваться для передачи параметров функций. Фигурные скобки содержат блоки кода функций и циклов.

## **1.14 Программные конструкции языка**

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

Таблица 1.7 Программные конструкции

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

## **1.15 Область видимости идентификаторов**

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

## **1.16 Семантические проверки**

Семантические проверки конструкции языка программирования DML-2021:

1. Наличие функции main
2. Единственность точки входа
3. Переопределение идентификаторов
4. Соответствие сигнатуры функции
5. Дублирование идентификатора
6. Использование необъявленного идентификатора
7. Присваивание значения переменной
8. Соответствие параметров функции
9. Совместимость типов
10. Правильность составленного условного оператора

## **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 DML-2021 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Все стандартные библиотеки реализованы на языке C++. Стандартные библиотеки подключены по умолчанию в программу. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные конечному пользователю. Содержимое библиотеки и описание функций представлено в таблице 1.8.

Таблица 1.8 Стандартная библиотека языка DML-2021

|  |  |
| --- | --- |
| Функция | Описание |
| **str** Date(); | Входной параметр: ничего.  Выходной параметр: текущее время системы.  Строковая функция, возвращает текущее дату системы. |
| **str** Time(); | Входной параметр: ничего.  Выходной параметр: текущее время системы.  Строковая функция, возвращает текущее время системы. |

## **1.19 Ввод и вывод данных**

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

В языке программирования DML-2021 ввод данных не поддерживается.

Функции, управляющие вводом/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 автоматически.

## **1.20 Точка входа**

Точкой входа является функция main.

## **1.21 Препроцессор**

В языке программирования DML-2021 препроцессор не предусматривается.

## **1.22 Соглашения о вызовах**

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

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## **1.23 Объектный код**

Язык программирования DML-2021 транслируется в язык ассемблера.

## **1.24 Классификация сообщений транслятора**

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

Таблица 1.9 Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-99 | Системные ошибки |
| 100-119 | Ошибки при работе с файлами |
| 200-219 | Ошибки лексического анализа |
| 300-319 | Ошибки синтаксического анализа |
| 600-319 | Ошибки семантического анализа |

## **1.25 Контрольный пример**

Контрольный пример представлен в приложении А.

# **Глава 2. Структура транслятора**

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

В языке DML-2021 исходный код транслируется в язык Assembler. 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 DML-2021 приведена на рисунке 1.



Рисунок 1 Структура транслятора языка программирования DML-2021

Транслятор разделен на несколько частей: лексический анализатор, синтаксический анализатор, семантический анализатор и генератор кода. Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором.

Цели лексического анализатора:

* убрать все лишние пробелы;
* выполнить распознавание лексем;
* построить таблицу лексем и таблицу идентификаторов;
* при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

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

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

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

## **2.2 Перечень входных параметров 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 DML-2021

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке DML-2021 , имеющий расширение .txt | Не предусмотрено |
| -log:<путь к log-файлу> | Файл журнала для вывода протоколов работы программы | Значение по умолчанию:  <имя in-файла>.log |
| -out:<путь к out-файлу> | Код программы сгенерированный на языке ассемблера | Значение по умолчанию:  <имя in-файла>.asm |

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

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

Таблица 2.2 Протоколы, формируемые транслятором языка DML-2021

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования DML-2021 . Содержит таблицу лексем и таблицу идентификаторов, протокол работы синтаксического анализатора и дерево разбора, полученные на этапе лексического и синтаксического анализа, а также результат работы алгоритма преобразования выражений к польской записи. |
| Выходной файл, c расширением "-asm:" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

# **Глава 3. Разработка лексического анализатора**

# **3.1 Структура лексического анализатора**

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 конструкции языка и 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.

Функции лексического анализатора:

− удаление «пустых» символов и комментариев. Для облегчения работы синтаксического анализатора

− распознавание идентификаторов и ключевых слов;

− распознавание разделителей и знаков операций.

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



Рисунок 3.1 Структура лексического анализатора

# **3.2. Контроль входных символов**

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

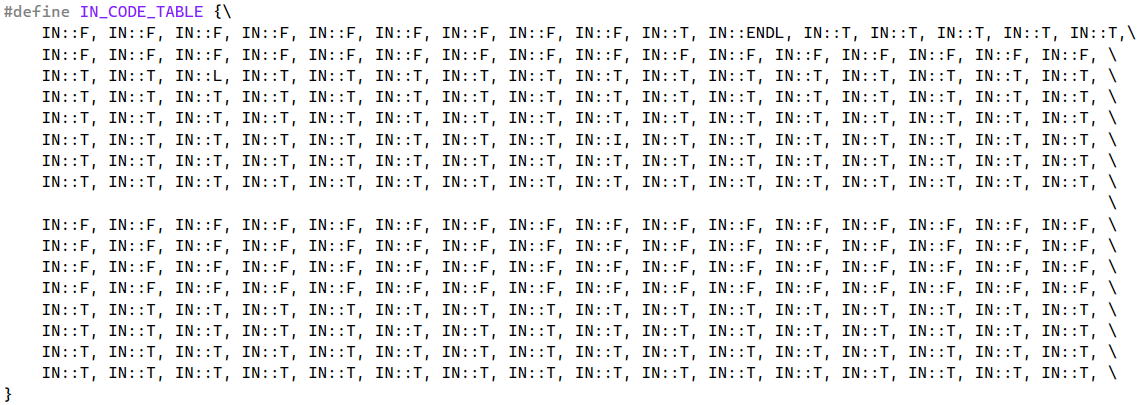


Рисунок 3.2. Таблица контроля входных символов

Таблица 3.1 Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенный | T |
| Запрещенный | F |
| Игнорируемый | I |
| Одинарная кавычка | L |

## **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 пробел и знак табуляции.

## **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 |
| uint, str | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 10 символов. |
| Литерал | l | Литерал любого доступного типа. |
| func | f | Объявление функции. |
| ret | r | Выход из функции. |
| main | m | Главная функция. |
| var | d | Объявление переменной. |
| if | ? | Разделение конструкций в /условном операторе. |
| else | e | Разделение конструкций в /условном операторе. |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| ( | ( | Передача параметров в функцию, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | = | Знак присваивания. |
| +, -, \*, / | v | Знаки операций |
| >, <, &, !, ^, \_ | v | 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. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Структура конечного автомата и пример графа перехода конечного автомата изображены на рисунках 3.3 и 3.4 соответственно.

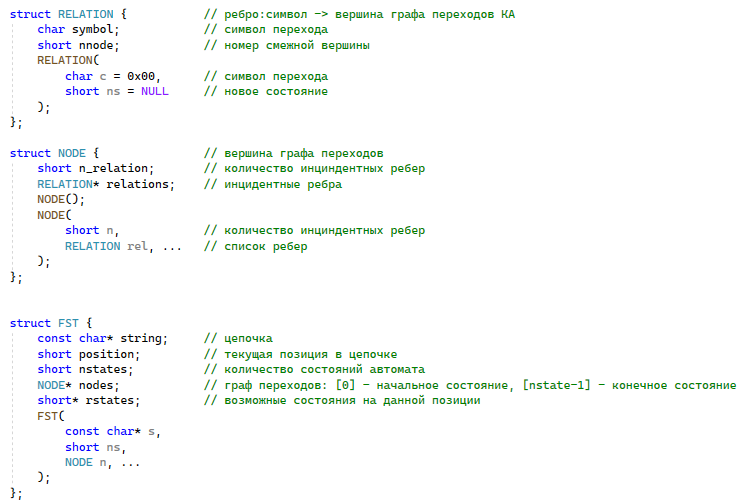


Рисунок 3.3 Структура конечного автомата

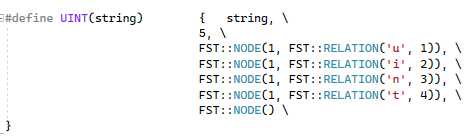


Рисунок 3.4 Пример реализации графа конечного автомата для токена uint

## **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 (line), номер в таблице идентификаторов, если лексема является идентификатором (idxTI) и приоритет, если лексема является операцией. Таблица идентификаторов содержит имя идентификатора (id), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и значение (или параметры функций) (value). Код со структурой таблицы лексем представлен на рисунке 3.4. Код со структурой таблицы идентификаторов представлен на рисунке 3.5.

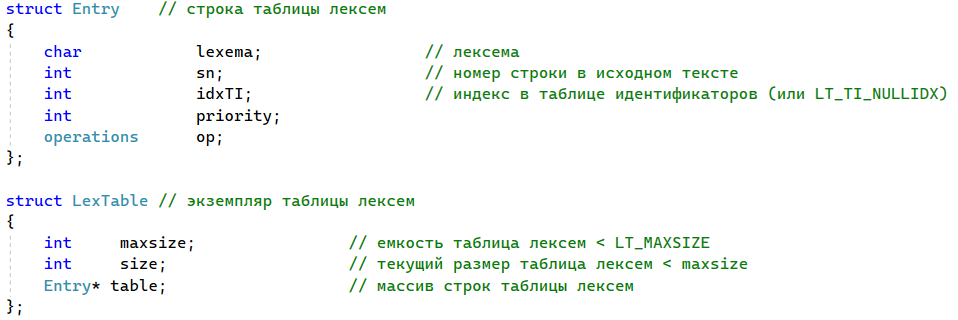


Рисунок 3.5 Структура таблицы лексем

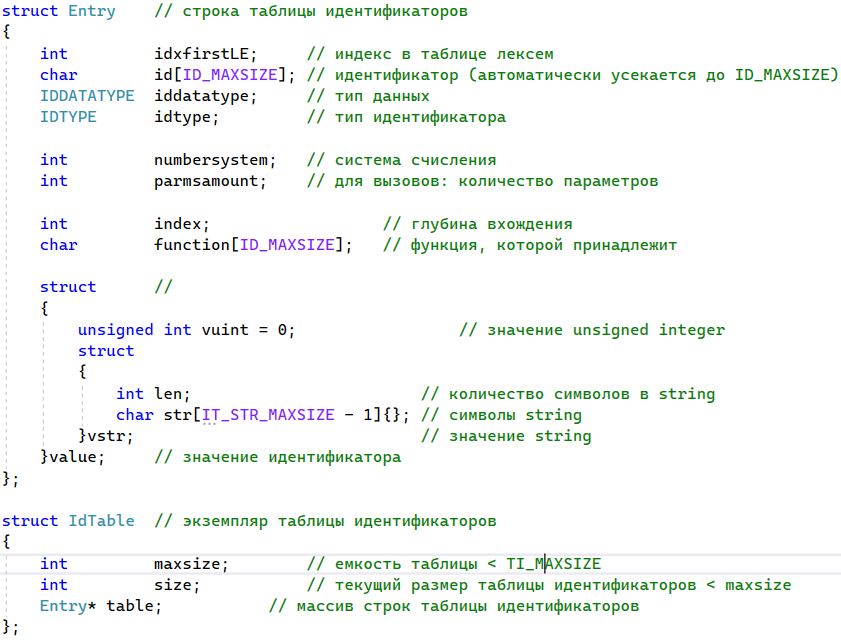


Рисунок 3.6 Структура таблицы идентификаторов

## **3.6 Принцип обработки ошибок**

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. При возникновении сообщения, лексический анализатор игнорирует найденную ошибку и продолжает работу с исходным кодом, при условии, что найденных ошибок не больше трех. Перечень сообщений представлен на рисунке 3.6.

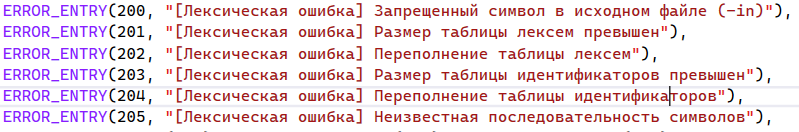


Рисунок 3.7 - Сообщения лексического анализатора

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

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. Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

## **3.8 Параметры лексического анализатора**

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

## **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 «**print**» представлен на рисунке 3.2, где S0 – начальное, а S5 – конечное состояние автомата.

p r i n t

Рисунок 3.8 Пример графа переходов для цепочки string

## **3.10 Контрольный пример**

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

# **Глава 4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализатор: часть компилятора, выполняющая синтаксический анализ, то есть исходный код проверяется на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией– дерево разбора. Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.



Рисунок 4.1 Структура синтаксического анализатора.

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

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

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

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

Правила языка DML-2021 представлены в приложении Г.

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

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

Описание нетерминальных символов содержится в таблице 4.1.

Таблица 4.1 Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | tfiFBS  piFUS  m{N}  tfiFB  piFU | Проверка правильности структуры программы |
| F | (P)  () | Проверка наличия параметров функции |
| P | ti  ti,P | Проверка на правильность параметров функции при её объявлении |
| B | {NrI;}  {rI;}  {N} | Проверка наличия тела функции |
| I | i  l | Проверка на недопустимое выражение |
| N | dti;N  dti=E;N  i=E;N  ?(R){X}N  ?(R){X}e{X}N  rE;N  iK;N  dti; | Проверка на правильность конструкции в теле функции |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| N | dti=E;  i=E;  ?(R){X}  ?(R){X}e{X}  pI;  rE;  iK; | Проверка на правильность конструкции в теле функции |
| R | i  ivi  ivl  lvi | Проверка на правильность в условном выражении |
| K | (W)  () | Проверка на правильность вызова функции |
| E | i  l  (E)  iK  iM  lM  (E)M  iKM | Проверка на правильность арифметического выражения |
| W | i  l  i,W  l,W | Проверка на правильность параметров вызываемой функции |
| M | vE  vEM | Проверка на правильность арифметических действий |
| X | dti;N  dti=E;N  i=E;N  pI;N  rE;N  iK;X  dti;  dti=E;  i=E;  pI;  rE;  iK; | Проверка на правильность конструкции условном выражение |

## 

## **4.3 Построение конечного магазинного автомата**

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

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит представляет из себя множества терминальных и нетерминальных символов, описание которых содержится в таблица 1.2 и 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $) |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ А) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

## **4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка DML-2021 . Данные структуры в приложении В.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

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

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

Перечень сообщений синтаксического анализатора представлен в листинге 4.1.

|  |
| --- |
| ERROR\_ENTRY(600, "[ Синтаксическая ошибка ] Неверная структура программы"),  ERROR\_ENTRY(601, "[ Синтаксическая ошибка ] Отсутствует список параметров функции"),  ERROR\_ENTRY(602, "[ Синтаксическая ошибка ] Ошибка в параметрах функции"),  ERROR\_ENTRY(603, "[ Синтаксическая ошибка ] Отсутствует тело функции"),  ERROR\_ENTRY(604, "[ Синтаксическая ошибка ] Недопустимое выражение"),  ERROR\_ENTRY(605, "[ Синтаксическая ошибка ] Отсутствует тело условия"),  ERROR\_ENTRY(606, "[ Синтаксическая ошибка ] Неверная конструкция в теле функции"),  ERROR\_ENTRY(607, "[ Синтаксическая ошибка ] Ошибка в условном выражении"),  ERROR\_ENTRY(608, "[ Синтаксическая ошибка ] Ошибка в вызове функции"),  ERROR\_ENTRY(609, "[ Синтаксическая ошибка ] Ошибка в арифметическом выражении"),  ERROR\_ENTRY(610, "[ Синтаксическая ошибка ] Ошибка в списке параметров при вызове функции"),  ERROR\_ENTRY(611, "[ Синтаксическая ошибка ] Ошибка в арифметическом выражении"),  ERROR\_ENTRY(612, "[ Синтаксическая ошибка ] Неверная конструкция в теле условия"), |

Листинг 4.1 – Перечень сообщений синтаксического анализатора

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

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

## **4.8. Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

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

## **4.9. Контрольный пример**

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

## **Глава 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. Некоторые проверки (такие как проверка на единственность точки входа, проверка на предварительное объявление переменной) осуществляются в процессе лексического анализа. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1. Структура семантического анализатора

## **5.2 Функции семантического анализатора**

Семантический анализатор проверяет правильность составления программных конструкций. При невозможности подобрать правило перехода будет выведен код ошибки, а так же код этой ошибки. Информация об ошибках выводится в консоль, а так же в протокол работы.

## **5.3 Структура и перечень семантических ошибок**

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

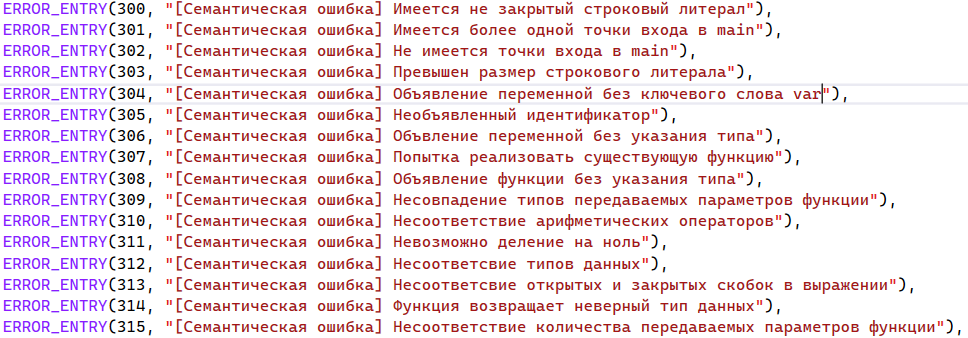


Рисунок 5.2 – Перечень сообщений семантического анализатора

## **5.4 Принцип обработки ошибок**

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

## **5.5 Контрольный пример**

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

Таблица 5.1 – Примеры диагностики ошибок

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| {  var uint x;  var uint y;  y = 10;  x = y - 50;  } | Ошибка 302: [Семантическая ошибка] Не имеется точки входа в main |

Продолжение таблицы 5.1

|  |  |
| --- | --- |
| main  {  var uint x;  var str string;  string = x;  } | Ошибка 312: [Семантическая ошибка] Несоответсвие типов данных  Строка 5 позиция -1 |
| uint func same()  {  ret 0;  }  uint func same()  {  ret 1;  }  main  {  var uint result = same();  } | Ошибка 307: [Семантическая ошибка] Попытка реализовать существующую функцию  Строка 6 позиция -1 |

## **Глава 6. Вычисление выражений**

## **6.1 Выражения, допускаемые языком**

В языке DML-2021 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, /, логические операции, как >, <, !, ^, &, \_, ~ и (), а также вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке DML-2021

|  |  |
| --- | --- |
| Приоритет | Операция |
| 0 | ( > < ~ ! ^ & \_ |
| 1 | , |
| 2 | + - |
| 3 | \* / |
| 4 | ) |

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

Все выражения языка DML-2021 преобразовываются к обратной польской записи.

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

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

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| y+pow(y) |  |  |
| +pow(y) | y |  |
| pow(y) | y | + |
| (y) | y | + |
| y) | y | + |
| ) | yy | + |
|  | yy@1+ |  |

## **6.3 Программная реализация обработки выражений**

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

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

## **6.4 Контрольный пример**

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

## **Глава 7. Генерация кода**

## **7.1 Структура генератора кода**

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



Рисунок 7.1 – Структура генератора кода

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

Элементы таблицы идентификаторов расположены сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке DML-2021 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке DML-2021 | Тип идентификатора на языке ассемблера | Пояснение |
| uint | dword | Хранит целочисленный тип данных без знака. |
| str | byte | Каждый символ строки типа str хранится в поле размером 1 байт. |

## **7.3 Статическая библиотека**

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

В языке программирования DML-2021 библиотеки подключаются по умолчанию. Подключение библиотеки в языке ассемблера происходит с помощью директивы includelib на этапе генерации кода. Далее с помощью оператора EXTRN объявляются имена функций из библиотеки. Оператор EXTRN выполняет две функции. Во-первых, он сообщает ассемблеру, что указанное символическое имя является внешним для текущего ассемблирования. Вторая функция оператора EXTRN указывает ассемблеру тип соответствующего символического имени. Ассемблирование является очень формальной процедурой, то ассемблер должен знать, что представляет из себя каждый символ. Это позволяет ему генерировать правильные команды. Вышеописанное проиллюстрировано на листинге 7.1.

|  |
| --- |
| out << ".586\n";  out << ".model flat, stdcall\n";  out << "includelib libucrt.lib\n";  out << "includelib kernel32.lib\n";  out << "includelib ../Debug/StaticLib.lib\n";  out << "ExitProcess PROTO :DWORD\n\n";  out << "EXTRN Date: proc\n";  out << "EXTRN Time: proc\n";  out << "EXTRN OutputInt: proc\n";  out << "EXTRN OutputStr: proc\n";  out << "\n.stack 4096\n\n"; |

Листинг 7.1 Фрагмент функции генерации кода

## **7.4 Особенности алгоритма генерации кода**

В языке DML-2021 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2.

Таблица идентификаторов

Таблица лексем

Генератор кода

asm-файл

Assembler

Рисунок 7.1 – Структура генератора кода

## **7.5 Входные параметры генератора кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке DML-2021 . Результаты работы генератора кода выводятся в файл с расширением .out.

## **7.6 Контрольный пример**

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

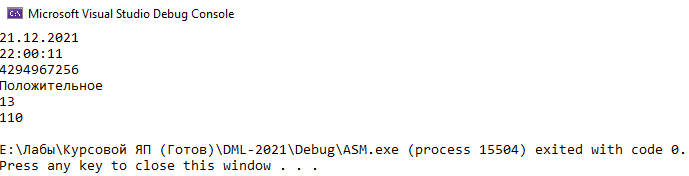


Рисунок 7.2 Результат работы программы на языке DML-2021

## **Глава 8. Тестирование транслятора**

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

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

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| uint func ёfelev{ret 11;} | Ошибка 200: [Лексическая ошибка] Запрещенный символ в исходном файле (-in)  Строка 1 позиция 9 |

## **8.2 Тестирование лексического анализатора**

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

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  var uint $lol$;  } | Ошибка 205: [ LEXICAL ] Неизвестная последовательность символов  Строка 3 позиция 12 |

## **8.3 Тестирование синтаксического анализатора**

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

Таблица 8.3 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| uint func wr(str s, uint x)  {  ret x;  }  main  {  var uint res = wr(9, );  } | Cтрока 8, [ SYNTAX ] Ошибка в списке параметров при вызове функции |

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования DML-2021 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка DML-2021;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
5. Осуществлена программная реализация синтаксического анализатора;
6. Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
7. Разработан транслятор кода на язык ассемблера;
8. Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка DML-2021 включает:

1. 2 типа данных;
2. Поддержка операторов ввода и перевода строки;
3. Наличие 4 арифметических операторов для вычисления выражений
4. Наличие 6 логических операторов для использования в условной конструкции
5. Поддержка функций и условий;
6. Наличие библиотеки стандартных функций языка
7. 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 ЯП 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 языке DML-2021

|  |
| --- |
| str func fposneg(uint n, str pos, str neg)  {  if (n ^ 0)  {  ret pos;  }  else  {  ret neg;  }  }  uint func pow(uint x)  {  x = x \* x;  ret x;  }  main  {  var str date = Date();  print date;  var str time = Time();  print time;  var uint x;  var uint y;  y = 10;  x = y - 50;  print x;  var str pos = 'Положительное';  var str neg = 'Отрицательное';  var str res = fposneg(x, pos, neg);  print res;  var uint i = 8x15;  print i;  var uint polish;  polish = y + pow(y);  print polish;  } |

**Приложение Б**

Листинг 1 Таблица идентификаторов контрольного примера

|  |
| --- |
| ----------------------------------------------------------------------------  # | Идентификатор | Тип данных | Тип идентификатора | Индекс в ТЛ | Значение  ----------------------------------------------------------------------------  0000 | fposneg | str | функция | 2 | -  0001 | fposnegn | uint | параметр | 5 | -  0002 | fposnegpos | str | параметр | 8 | -  0003 | fposnegneg | str | параметр | 11 | -  0004 | ^ | - | оператор | 17 | -  0005 | L1 | uint | литерал | 18 | 0  0006 | pow | uint | функция | 34 | -  0007 | powx | uint | параметр | 37 | -  0008 | \* | - | оператор | 44 | -  0009 | main | uint | функция | 50 | -  0010 | maindate | str | переменная | 54 | [0]""  0011 | Date | str | функция | 56 | -  0012 | maintime | str | переменная | 65 | [0]""  0013 | Time | str | функция | 67 | -  0014 | mainx | uint | переменная | 76 | 0  0015 | mainy | uint | переменная | 80 | 0  0016 | L2 | uint | литерал | 84 | 10  0017 | - | - | оператор | 90 | -  0018 | L3 | uint | литерал | 89 | 50  0019 | mainpos | str | переменная | 97 | [0]""  0020 | L4 | str | литерал | 99 | [13]"Положительное"  0021 | mainneg | str | переменная | 103 | [0]""  0022 | L5 | str | литерал | 105 | [13]"Отрицательное"  0023 | mainres | str | переменная | 109 | [0]""  0024 | maini | uint | переменная | 125 | 0  0025 | L6 | uint | литерал | 127 | 13  0026 | mainpolish | uint | переменная | 134 | 0  0027 | + | - | оператор | 142 | -  ----------------------------------------------------------------------------  Количество идентификаторов: 28 |

Листинг 2 Таблица лексем контрольного примера

|  |
| --- |
| ----------------------------------------------------------------------------  # | Лексема | Строка | Индекс в ТИ  ----------------------------------------------------------------------------  0000 | t | 1 | -  0001 | f | 1 | -  0002 | i | 1 | 0  0003 | ( | 1 | -  0004 | t | 1 | -  0005 | i | 1 | 1  0006 | , | 1 | -  0007 | t | 1 | -  0008 | i | 1 | 2  0009 | , | 1 | -  0010 | t | 1 | -  0011 | i | 1 | 3  0012 | ) | 1 | -  0013 | { | 2 | -  0014 | ? | 3 | -  0015 | ( | 3 | -  0016 | i | 3 | 1  0017 | v | 3 | 4  0018 | l | 3 | 5  0019 | ) | 3 | -  0020 | { | 4 | -  0021 | r | 5 | -  0022 | i | 5 | 2  0023 | ; | 5 | -  0024 | } | 6 | -  0025 | e | 7 | -  0026 | { | 8 | -  0027 | r | 9 | -  0028 | i | 9 | 3  0029 | ; | 9 | -  0030 | } | 10 | -  0031 | } | 11 | -  0032 | t | 13 | -  0033 | f | 13 | -  0034 | i | 13 | 6  0035 | ( | 13 | -  0036 | t | 13 | -  0037 | i | 13 | 7  0038 | ) | 13 | -  0039 | { | 14 | -  0040 | i | 15 | 7  0041 | = | 15 | -  0042 | i | 15 | 7  0043 | v \* | 15 | 8 |

Листинг 2 (продолжение) Таблица лексем контрольного примера

|  |
| --- |
| 0044 | i | 15 | 7  0045 | ; | 15 | -  0046 | r | 16 | -  0047 | i | 16 | 7  0048 | ; | 16 | -  0049 | } | 17 | -  0050 | m | 19 | 9  0051 | { | 20 | -  0052 | d | 21 | -  0053 | t | 21 | -  0054 | i | 21 | 10  0055 | = | 21 | -  0056 | i | 21 | 11  0057 | ( | 21 | -  0058 | ) | 21 | -  0059 | ; | 21 | -  0060 | p | 22 | -  0061 | i | 22 | 10  0062 | ; | 22 | -  0063 | d | 23 | -  0064 | t | 23 | -  0065 | i | 23 | 12  0066 | = | 23 | -  0067 | i | 23 | 13  0068 | ( | 23 | -  0069 | ) | 23 | -  0070 | ; | 23 | -  0071 | p | 24 | -  0072 | i | 24 | 12  0073 | ; | 24 | -  0074 | d | 25 | -  0075 | t | 25 | -  0076 | i | 25 | 14  0077 | ; | 25 | -  0078 | d | 26 | -  0079 | t | 26 | -  0080 | i | 26 | 15  0081 | ; | 26 | -  0082 | i | 27 | 15  0083 | = | 27 | -  0084 | l | 27 | 16  0085 | ; | 27 | -  0086 | i | 28 | 14  0087 | = | 28 | -  0088 | i | 28 | 15  0089 | v - | 28 | 17  0090 | l | 28 | 18  0091 | ; | 28 | -  0092 | p | 29 | -  0093 | i | 29 | 14 |

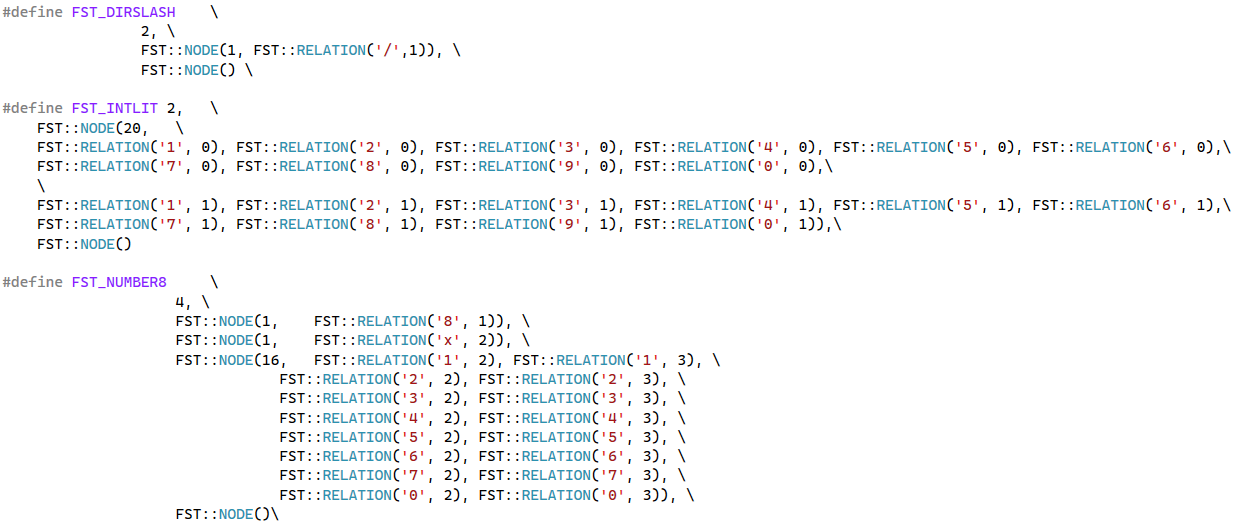
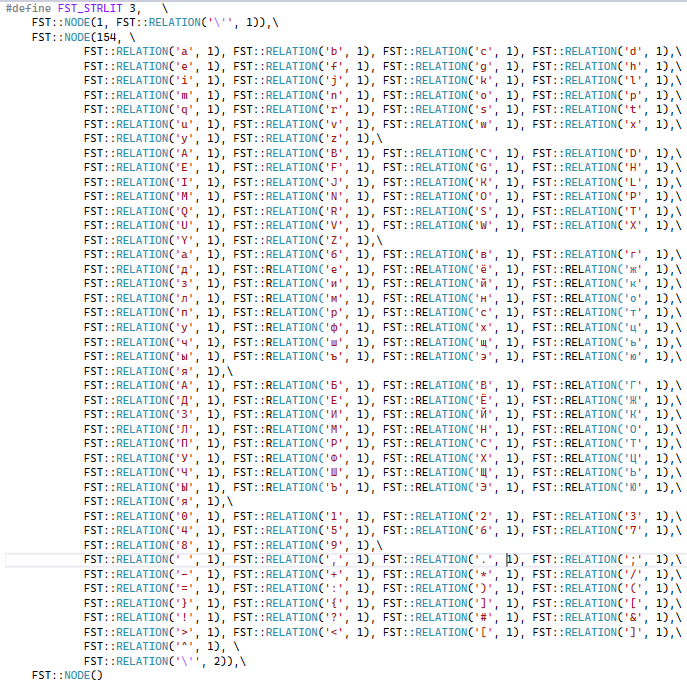
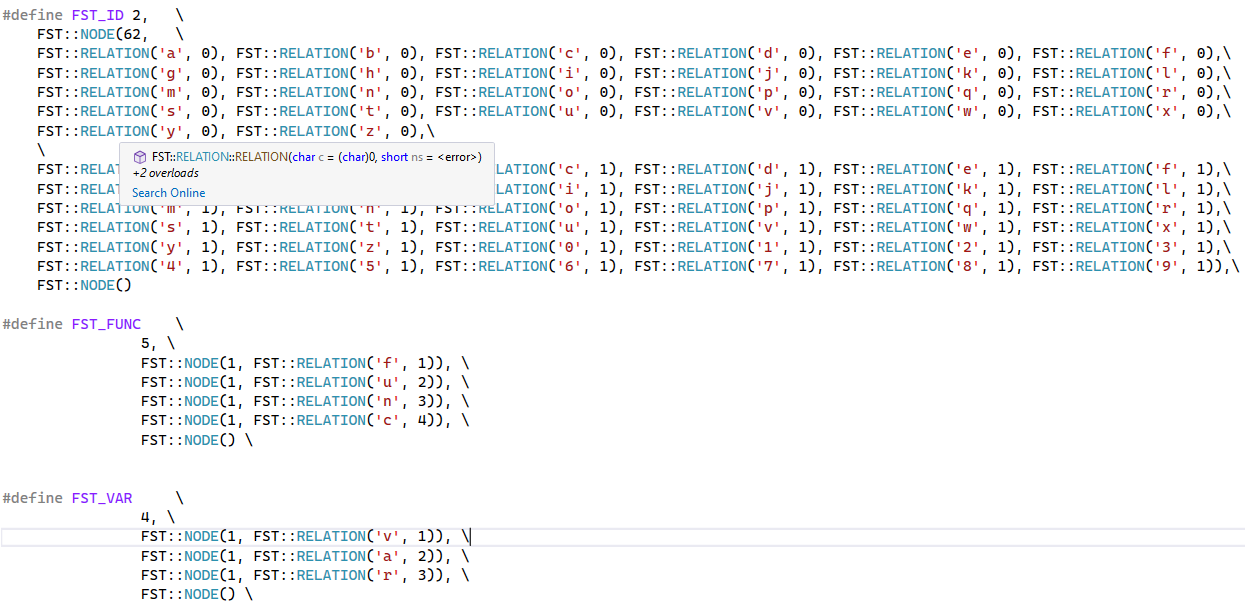
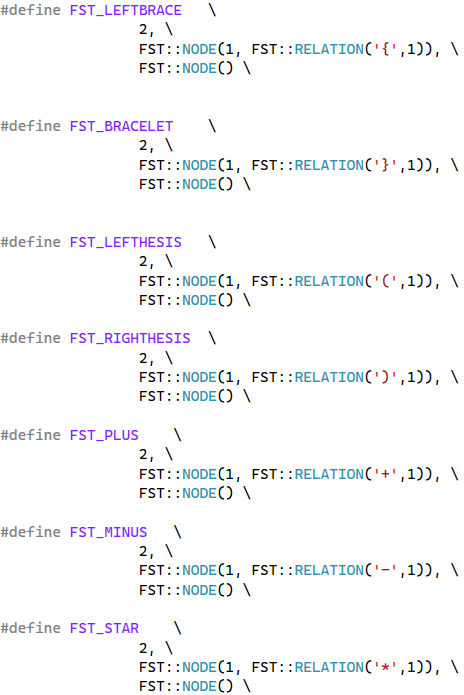
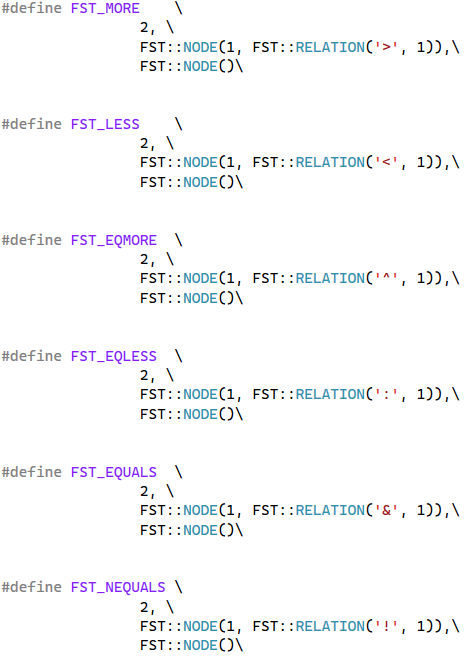
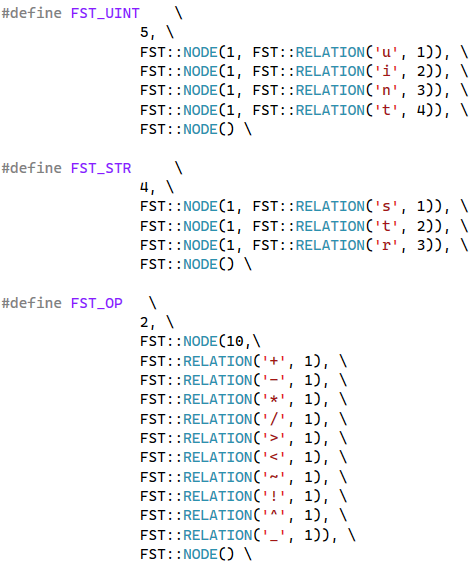
Листинг 2 (продолжение) Таблица лексем контрольного примера

|  |
| --- |
| 0094 | ; | 29 | -  0095 | d | 30 | -  0096 | t | 30 | -  0097 | i | 30 | 19  0098 | = | 30 | -  0099 | l | 30 | 20  0100 | ; | 30 | -  0101 | d | 31 | -  0102 | t | 31 | -  0103 | i | 31 | 21  0104 | = | 31 | -  0105 | l | 31 | 22  0106 | ; | 31 | -  0107 | d | 32 | -  0108 | t | 32 | -  0109 | i | 32 | 23  0110 | = | 32 | -  0111 | i | 32 | 0  0112 | ( | 32 | -  0113 | i | 32 | 14  0114 | , | 32 | -  0115 | i | 32 | 19  0116 | , | 32 | -  0117 | i | 32 | 21  0118 | ) | 32 | -  0119 | ; | 32 | -  0120 | p | 33 | -  0121 | i | 33 | 23  0122 | ; | 33 | -  0123 | d | 34 | -  0124 | t | 34 | -  0125 | i | 34 | 24  0126 | = | 34 | -  0127 | l | 34 | 25  0128 | ; | 34 | -  0129 | p | 35 | -  0130 | i | 35 | 24  0131 | ; | 35 | -  0132 | d | 36 | -  0133 | t | 36 | -  0134 | i | 36 | 26  0135 | ; | 36 | -  0136 | i | 37 | 26  0137 | = | 37 | -  0138 | i | 37 | 15  0139 | v + | 37 | 27  0140 | i | 37 | 6  0141 | ( | 37 | -  0142 | i | 37 | 15  0143 | ) | 37 | - |

Листинг 2 (продолжение) Таблица лексем после контрольного примера

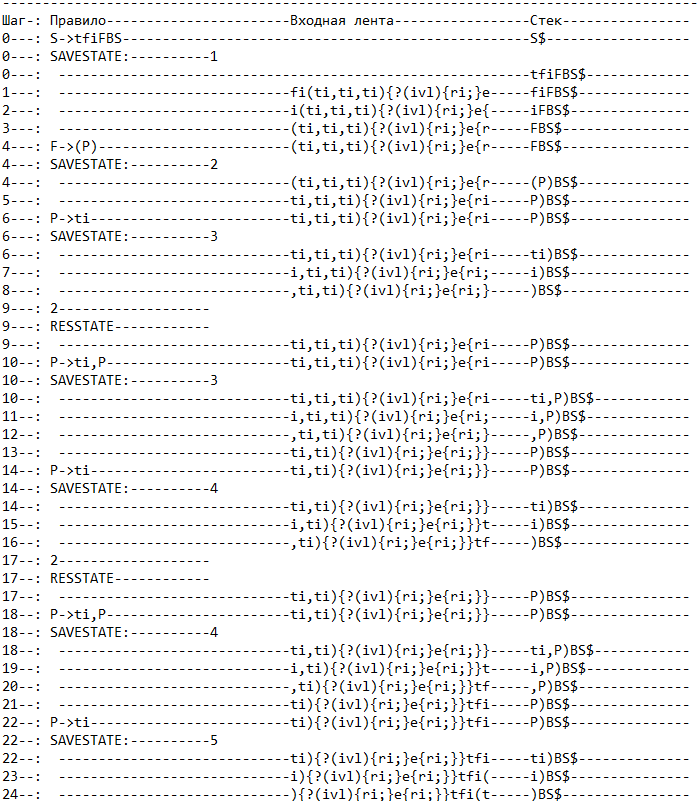
|  |
| --- |
| 0143 | ) | 37 | -  0144 | ; | 37 | -  0145 | p | 38 | -  0146 | i | 38 | 26  0147 | ; | 38 | -  0148 | } | 39 | -  ----------------------------------------------------------------------------  Всего лексем: 149 |

**Приложение В**

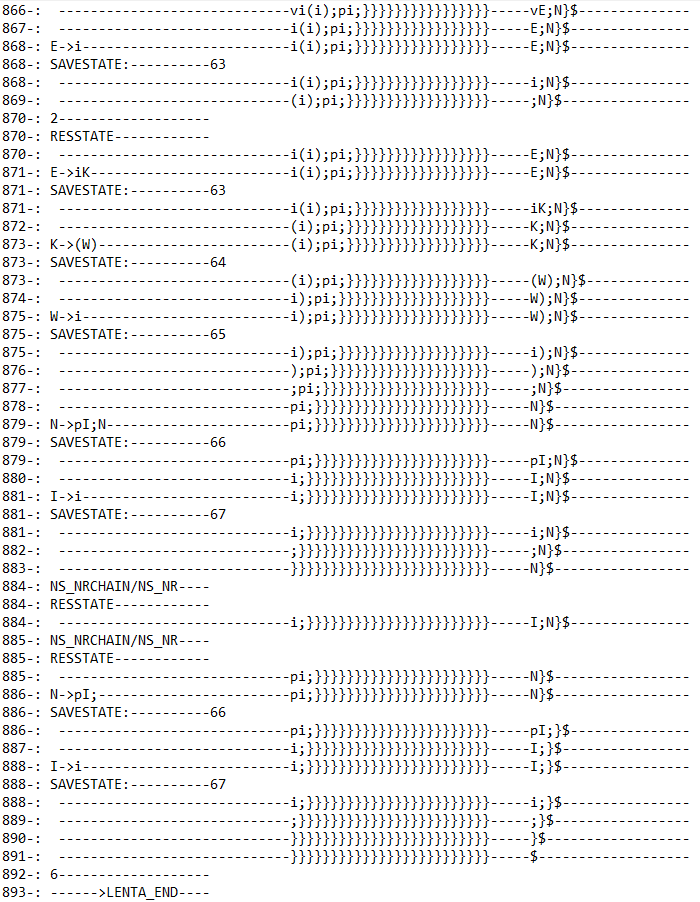


**Приложение Г**

Начало трассировки синтаксического анализатора



Конец трассировки синтаксического анализатора



**Приложение Д**

Листинг 1 Код на языке Assembler с контрольного примера

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ../Debug/StaticLib.lib  ExitProcess PROTO :DWORD  EXTRN Date: proc  EXTRN Time: proc  EXTRN OutputInt: proc  EXTRN OutputStr: proc  .stack 4096  .CONST  L1 DWORD 0  L2 DWORD 10  L3 DWORD 50  L4 BYTE "Положительное", 0  L5 BYTE "Отрицательное", 0  L6 DWORD 13  .data  buffer BYTE 256 dup(0)  maindate DWORD ?  maintime DWORD ?  mainx DWORD 0  mainy DWORD 0  mainpos DWORD ?  mainneg DWORD ?  mainres DWORD ?  maini DWORD 0  mainpolish DWORD 0  .code  fposneg PROC fposnegn : DWORD, fposnegpos : DWORD, fposnegneg : DWORD  mov eax, fposnegn  cmp eax, L1  jae m0  jb m1  je m1  m0:  push fposnegpos  jmp local0  jmp e0  m1:  push fposnegneg |

Листинг 1 (продолжение) Код на языке Assembler с контрольного примера

|  |
| --- |
| jmp local0  e0:  local0:  pop eax  ret  fposneg ENDP  pow PROC powx : DWORD  push powx  push powx  pop eax  pop ebx  mul ebx  push eax  pop powx  push powx  jmp local1  local1:  pop eax  ret  pow ENDP  main PROC  call Date  push eax  pop maindate  push maindate  call OutputStr  call Time  push eax  pop maintime  push maintime  call OutputStr  push L2  pop mainy  push mainy  push L3  pop ebx  pop eax  sub eax, ebx  push eax  pop mainx  push mainx  call OutputInt  push offset L4  pop mainpos  push offset L5  pop mainneg  push mainx  push mainpos |

Листинг 1 (продолжение) Код на языке Assembler с контрольного примера

|  |
| --- |
| push mainx  call fposneg  push eax  pop mainres  push mainres  call OutputStr  push L6  pop maini  push maini  call OutputInt  push mainy  push mainy  pop edx  push mainy  call pow  push eax  pop eax  pop ebx  add eax, ebx  push eax  pop mainpolish  push mainpolish  call OutputInt  call ExitProcess  main ENDP  end main |