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 ShDA-2019»

Выполнил студент Шестопалов Денис Александрович

(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 2019

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПЕУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий   
Кафедра программной инженерии

Утверждаю

Заведующая кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_ Н.В. Пацей

подпись инициалы и фамилия

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2018г.

**ЗАДАНИЕ**

**к курсовому проектированию**

**по дисциплине** "Языки программирования"

Специальность: 1-40 01 01 Программное обеспечение информационных технологий Группа: 6

Студент: Шестопалов Денис Александрович

(фамилия, имя, отчество)

**1. Тема проекта** Разработка компилятора ShDA-2019

утверждена приказом по университету от «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. №

**2. Срок сдачи студентом законченного проекта:** 16 декабря 2019 г.

**3. Исходные данные к проекту:**

Разработка программы осуществляется на языке C++ (стандартизации International Standard ISO/IEC 14882:2017(E) Programming Language C++ 17) в среде разработки Visual Studio 2017. Операционная система под которой происходит разработка Windows 10 (64-bit). Типы данных: беззнаковый целый, строковый . Программные конструкции: функции, условный оператор. Функции стандартной библиотеки: лексикографическое сравнение строк, длина строки. Арифметические операции: +, -, \*, /,:. Оператор вывода в стандартный поток: out, outn. Литералы: целые(восьмеричный | десятичный), строковый(произвольно).

**4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов):**

Введение

1) Спецификация языка программирования

2) Структура транслятора

3) Разработка лексического анализатора

4) Разработка синтаксического анализатора

5) Разработка семантического анализатора

6) Вычисление выражений

7) Генерация кода

8) Тестирование транслятора (Разработка и тестирование интерпретатора)

Приложения

Литература

**5. Перечень графического материала (с точным указанием обязательных чертежей)**

1) Граф

**6. Консультанты по проекту с указанием относящихся к ним разделов проекта**

|  |  |
| --- | --- |
| Раздел | Консультант |
| Разработка синтаксического и семантического анализатора. | Наркевич А. С. |
| Генерация кода. Разработка тестовых примеров. | Наркевич А. С. |
| Оформление пояснительной записки к курсовому проект. | Наркевич А. С. |
|  |  |

**7. Календарный план**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Наименование этапов курсового проекта | Срок выполнения этапов проекта | Примечание |
| 1 | Спецификация специализированного языка SDA-2019 | 07.10.2019 |  |
| 2 | Разработка лексического анализатора | 21.10.2019 |  |
| 3 | Разработка синтаксического анализатора | 04.11.2019 |  |
| 4 | Разработка семантического анализатора | 18.11.2019 |  |
| 5 | Генерация кода | 02.12.2019 |  |
| 6 | Тестирование компилятора | 02.12.2019 |  |
| 7 | Оформление пояснительной записки к курсовому проект | 16.12.2019 |  |
| 8 | Сдача проекта | 16.12.2019 |  |

**8. Дата выдачи задания 15.09.2018**

Руководитель Наркевич А.С.

(подпись)

Задание принял к исполнению Шестопалов Д.А.

(дата и подпись студента)

Оглавление

[Введение 7](#_Toc27384604)

[1. Спецификация языка программирования 8](#_Toc27384605)

[1.1. Характеристика языка программирования 8](#_Toc27384606)

[1.2. Алфавит языка 8](#_Toc27384607)

[1.3. Применяемые сепараторы 8](#_Toc27384608)

[1.4. Применяемые кодировки 9](#_Toc27384609)

[1.5. Типы данных 9](#_Toc27384610)

[1.6. Преобразование типов данных 10](#_Toc27384611)

[1.7. Идентификаторы 10](#_Toc27384612)

[1.8. Литералы 11](#_Toc27384613)

[1.9. Объявление данных 11](#_Toc27384614)

[1.10. Инициализация данных 11](#_Toc27384615)

[1.11. Инструкции языка 11](#_Toc27384616)

[1.12. Операции языка 12](#_Toc27384617)

[1.13. Выражения и их вычисления 13](#_Toc27384618)

[1.14. Конструкции языка 13](#_Toc27384619)

[1.15. Область видимости 14](#_Toc27384620)

[1.16. Семантические проверки 14](#_Toc27384621)

[1.17. Распределение оперативной памяти на этапе выполнения 14](#_Toc27384622)

[1.18. Стандартная библиотека и её вызов 14](#_Toc27384623)

[1.19. Ввод и вывод данных 15](#_Toc27384624)

[1.20. Точка входа 15](#_Toc27384625)

[1.21. Препроцессор 16](#_Toc27384626)

[1.22. Соглашения о вызовах 16](#_Toc27384627)

[1.23. Объектный код 16](#_Toc27384628)

[1.24. Классификация сообщений транслятора 16](#_Toc27384629)

[1.25. Контрольный пример 16](#_Toc27384630)

[2. Структура транслятора 17](#_Toc27384631)

[2.1. Компоненты транслятора, их назначение и принципы взаимодействия 17](#_Toc27384632)

[2.2. Перечень входных параметров транслятора 18](#_Toc27384633)

[2.3. Параметры, формируемые транслятором 18](#_Toc27384634)

[3. Разработка лексического анализатора 19](#_Toc27384635)

[3.1. Структура лексического анализатора 19](#_Toc27384636)

[3.2. Контроль входных символов 19](#_Toc27384637)

[3.3. Удаление избыточных символов 20](#_Toc27384638)

[3.4. Перечень ключевых слов 20](#_Toc27384639)

[3.5. Основные структуры данных 21](#_Toc27384640)

[3.6. Структура и перечень сообщений лексического анализатора 21](#_Toc27384641)

[3.7. Принцип обработки ошибок 22](#_Toc27384642)

[3.8. Параметры лексического анализатора 22](#_Toc27384643)

[3.9. Алгоритм лексического анализа 22](#_Toc27384644)

[3.10. Контрольный пример 22](#_Toc27384645)

[4. Разработка синтаксического анализатора 23](#_Toc27384646)

[4.1. Структура синтаксического анализатора 23](#_Toc27384647)

[4.2. Контекстно свободная грамматика, описывающая синтаксис языка 23](#_Toc27384648)

[4.3. Построение конечного магазинного автомата 24](#_Toc27384649)

[4.4. Основные структуры данных 25](#_Toc27384650)

[4.5. Описание алгоритма синтаксического разбора 25](#_Toc27384651)

[4.6. Структура и перечень сообщений синтаксического анализатора 26](#_Toc27384652)

[4.7. Параметры синтаксического анализатора и режимы его работы 26](#_Toc27384653)

[4.8. Принцип обработки ошибок 26](#_Toc27384654)

[4.9. Контрольный пример 26](#_Toc27384655)

[5. Разработка семантического анализатора 27](#_Toc27384656)

[5.1. Структура семантического анализатора 27](#_Toc27384657)

[5.2. Функции семантического анализатора 27](#_Toc27384658)

[5.3. Принцип обработки ошибок 28](#_Toc27384659)

[6. Вычисление выражений 29](#_Toc27384660)

[6.1. Выражения, допускаемые языком 29](#_Toc27384661)

[6.2. Польская запись и принцип её простроения 29](#_Toc27384662)

[6.3. Программная реализация обработки выражений 30](#_Toc27384663)

[6.4. Контрольный пример 30](#_Toc27384664)

[7. Генерация кода 31](#_Toc27384665)

[7.1. Структура генератора кода 31](#_Toc27384666)

[7.2. Представление типов данных в оперативной памяти 31](#_Toc27384667)

[7.3. Статическая библиотека 32](#_Toc27384668)

[7.4. Алгоритм работы генератора кода 32](#_Toc27384669)

[7.5. Контрольный пример 32](#_Toc27384670)

[8. Тестирование транслятора 33](#_Toc27384671)

[8.1. Общие положения 33](#_Toc27384672)

[Заключение 34](#_Toc27384673)

[Приложение А. Контрольный пример 35](#_Toc27384674)

[Приложение Б. Таблица символов 36](#_Toc27384675)

[Приложение В. Результат работы лексического анализатора 37](#_Toc27384676)

[Приложение Г. Правила синтаксического анализатора 43](#_Toc27384677)

[Приложение Д. Структура синтаксического анализатора 44](#_Toc27384678)

[Приложение Е. Результат работы синтаксического анализатора 45](#_Toc27384679)

[Приложение Ж. Код программы, приводящей выражения к обратной польской записи 46](#_Toc27384680)

[Приложение 3. Генерация кода 48](#_Toc27384681)

[Литература 50](#_Toc27384682)

## Введение

Транслятор – это комплекс отдельных программ, позволяющих преобразовывать исходный код на одном языке программирования в исходный код на другом языке программирования. Задача транслятора – сделать программу, написанную на некотором языке программирования, понятной компьютеру. Этого можно добиться одним из двух способов: компиляцией или интерпретацией. Язык ShDA-2019 является компилируемым. Компилятор переводит исходную программу в эквивалентную ей на язык ассемблера.

Процесс создания компилятора можно свести к решению нескольких задач, которые принято называть фазами компиляции. Разрабатываемый компилятор состоит из следующих фаз:

лексический анализ;

синтаксический анализ;

семантический анализ;

генерация кода;

Цель моего курсового проекта – разработка всех частей транслятора под собственный язык программирования. В процессе работы проведены многие исследования на предмет эффективности тех или иных алгоритмов.

Курсовой проект разбит на 8 глав, в каждой из которых подробно описан процесс работы анализаторов и генератора, а также тестирование самого транслятора.

## Спецификация языка программирования

## Характеристика языка программирования

Язык программирования ShDA-2019 – это функциональный, строго типизированный, компилируемый язык. Не является объектно-ориентированным.

## Алфавит языка

Исходный код языка программирования ShDA–2019 может содержать символы латинского алфавита малого и верхнего регистра, цифры десятеричной системы счисления от 0 до 9, в строковых литералах разрешены символы латинского алфавита, а также знаки препинания.

## Применяемые сепараторы

Символы-сепараторы – символы, используемые для разделения отдельных лексических единиц или функциональных элементов в исходном коде программы.

Символы–сепараторы представлены в таблице 1.1.

Таблица 1.1. Сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ;  +  -  \*  /  : | Разделение инструкций |
| “ ”(пробел) | Разделение цепочек |
| , | Разделение параметров функций |
| Продолжение таблицы 1.1. | |
| {  } | Программный блок |
| (  ) | Параметры и приоритетность операций(в выражениях) |
| [  ] | Блок условной конструкции |

## Применяемые кодировки

Для написания исходного кода на языке программирования ShDA-2019 используется кодировка Windows-1251, представленная на рисунке ниже

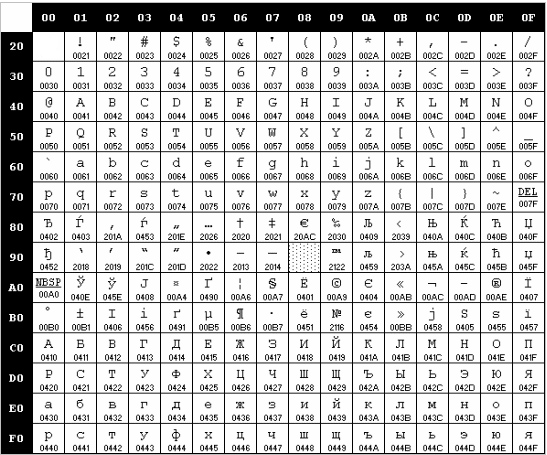


Рисунок 1.1. Таблица кодировки Windows-1251

## Типы данных

В языке SDA-2019 есть 2 типа данных: беззнаковый целый и строковый. Описание типов данных, предусмотренных в данном языке представлено в таблице 1.2.

Таблица 1.2. Типы данных языка программирования SDA-2019

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| int | Фундаментальный тип данных. Предусмотрен для объявления целочисленных положительных данных (4 байта). Диапазон от 0 до 232-1  Автоматически инициализируется нулевым значением.  Возможные операции:  + – бинарный, суммирование;  - – бинарный, вычитание;  \* – бинарный, умножение;  / – бинарный, деление  : – бинарный, остаток от деления  = – присваивание значения; |
| str | Фундаментальный тип данных. Предусмотрен для объявления строк. (1 символ – 1 байт). Автоматическая инициализация строкой нулевой длины. Максимальное количество символов в строке – 255.  Возможные операции:  = – копирование одной строки в другую; |

## Преобразование типов данных

В языке программирования ShDA–2019 преобразование типов данных не поддерживается.

## Идентификаторы

В имени идентификатора допускаются символы латинского алфавита нижнего и верхнего регистра. Также имя идентификатора может содержать цифры, но цифра не должна находиться на первой позиции. Максимальная длина идентификатора равна 5. Если его длина будет превышать максимальное значение, то имя идентификатора будет урезаться.

## Литералы

Литерал – лексема, которая непосредственно представляет некоторое значение. В языке существует 2 типа литералов: целого типа и cтроковые. Краткое описание литералов представлено в таблице 1.3.

Таблица 1.3. Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные литералы в восьмеричной или десятичной системе счисления. Не имеют дробных частей, экспонент и знаков. Литералы только rvalue. |
| Строковые литералы | Символы, заключённые в “” (двойные кавычки), инициализируются как “str”, строковые переменные. Только rvalue. Максимальная длина – 255 символов |

## Объявление данных

Переменные объявляются при помощи конструкции:

var <тип данных><идентификатор>;

Область видимости сверху вниз (по принципу С++). Все переменные являются видимыми внутри функций.

## Инициализация данных

Для инициализации переменной каким-либо значением применяется конструкция:

<идентификатор>=<значение>;

При объявлении числовой переменной её значение по умолчанию будет равняться нулю, строковой переменной – строка нулевой длины.

## Инструкции языка

Все возможные инструкции языка программирования SDA-2019 представлены в таблице 1.4.

Таблица 1.4. Инструкции языка программирования SDA-2019

|  |  |
| --- | --- |
| Инструкция | Синтаксис на языке программирования SDA-2019 |
| Точка входа | def Main{  } |
| Объявление переменной | var <тип данных> <идентификатор>; |
| Присваивание | <идентификатор>=<значение>; |
| Объявление функции | def <тип данных><идентификатор>(<тип данных> <идентификатор>,…)  {  …  } |
| Условный оператор | if(условие)  [  …  ]  else  [  …  ] |
| Возврат данных из функции | return <идентификатор>|<литерал>; |
| Вывод данных | out(<идентификатор>|<литерал>);  outn(<идентификатор>|<литерал>); |

## Операции языка

Язык программирования ShDA-2019 может выполнять арифметические операции, представленные в таблице 1.5.

Таблица 1.5. Операции и их приоритет

|  |  |
| --- | --- |
| Операция | Приоритетность операции |
| (  ) | 1 |
| : | 2 |
| \*  / | 3 |
| +  - | 4 |

Максимальный приоритет имеют операции с значением 1, минимальным - 4 соответственно.

## Выражения и их вычисления

Выражением называется совокупность переменных, знаков операций, имён функций, скобок, которая может быть вычислена в соответствии с синтаксисом языка программирования. Результатом вычисления выражения является величина определённого типа. Выражения должны быть записаны в одну строку. Круглые скобки в выражении используются для изменения приоритета операций. Не допускается запись двух подряд идущих арифметических операций.

## Конструкции языка

Ключевые программные конструкции языка программирования ShDA-2019 представлены в таблице 1.6.

Таблица 1.6. Программные конструкции языка программирования ShDA-2019

|  |  |
| --- | --- |
| Название | Синтаксис |
| Точка входа | def Main{  …  } |
| Функция | def <тип данных><идентификатор>(<тип данных> <идентификатор>,…){  } |
| Условный оператор | if(условие) {  …  }  else {  …  } |

## Область видимости

Область видимости языка программирования ShDA-2019 используется по принципу сверху вниз. Все переменные являются видимыми внутри объявленной функции.

## Семантические проверки

Семантические проверки невозможно проверить с помощью грамматик. На этапе семантического анализа код проверяется на наличие следующих ошибок:

1. типы данных в выражении не совпадают
2. недопустимый тип данных в выражении
3. отсутствует точка входа
4. обнаружено несколько точек входа
5. тип функции и возвращаемого значения не совпадают

## Распределение оперативной памяти на этапе выполнения

Транслированный код использует две области памяти: сегмент констант(литералы) и сегмент данных(переменные и параметры функций).

## Стандартная библиотека и её вызов

Функции стандартной библиотеки и их описание представлено в таблице 1.7.

Таблица 1.7. Состав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Параметры функции | Описание функции |
| strCheck(str1,str2) | void | str1,str2-строки | Функция выполняет лексикографическое сравнение строк и выводит результат |
| isGreather(int1,int2) | void | int1, int2- число | Функция проверяет, является ли первое число больше второго и выводит результат в консоль. |
| outint | void | int – число | Функция выводит число в консоль |
| outintn | void | int – число | Функция выводит число в консоль и перемещает каретку на новую строку |
| outstr | void | str – строка | Функция выводит строку в консоль |
| outstrn | void | str – строка | Функция выводит строку в консоль и перемещает каретку на новую строку |

## Ввод и вывод данных

Вывод данных осуществляется программной конструкцией out либо outn, в скобках указывается имя идентификатора либо литерала. Вычисление выражений внутри конструкции не предусматривается.

## Точка входа

Точкой входа является функция Main.

## Препроцессор

Препроцессор в языке программирования ShDA-2019 не предусмотрен.

## Соглашения о вызовах

В языке ShDA-2019 используется соглашение о вызовах stdcall.

Особенности stdcall:

1. Все параметры передаются через стек
2. Память высвобождает вызываемый код
3. Занесение параметров в стек идёт справа налево

## Объектный код

Язык программирования SDA-2019 транслируется в язык Assembler.

## Классификация сообщений транслятора

В случае возникновения ошибки в коде языка программирования ShDA-2019 и выявления её транслятором на экран выводится сообщение

Сообщения транслятора:

0-200 - системные ошибки

201-299 - ошибки лексического анализа

300-399 - ошибки семантического анализа

600-699 – ошибки синтаксического анализа

## Контрольный пример

Контрольный пример показывает результат работы всех этапов трансляции исходного кода на языке ShDA-2019 в язык Assembler.

Пример представлен в приложении А.

## Структура транслятора

## Компоненты транслятора, их назначение и принципы взаимодействия

Все компоненты, входящие в состав транслятора языка программирования ShDA-2019, представлены на рисунке 2.1.

 Рисунок 2.1. Структура языка программирования ShDA-2019

Лексический анализатор – часть транслятора, выполняющая лексический анализ. На данном этапе распознаётся правильность составления лексем и идентификаторов. На вход лексического анализатора приходит файл с исходным кодом. После окончания данного этапа получается таблица лексем и таблица идентификаторов.

Синтаксический анализатор – часть транслятора, выполняющая синтаксический анализ. Проверяется правильность расположения идентификаторов и ключевых слов в исходном коде. Для того, чтобы провести данную операцию используются таблица лексем и идентификаторов. На выходе получается дерево разбора.

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть исходный код проверяется на наличие ошибок, которые невозможно проверить 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.

## Перечень входных параметров транслятора

Входные параметры, которые можно передать транслятору представлены в таблице 2.1.

Табл. 2.1 Входные параметры транслятора

|  |  |
| --- | --- |
| Входной параметр | Описание параметра |
| -in:\*.txt | Файл с исходным кодом на языке ShDA-2019, имеющий название \* и расширение .txt |
| -log | Параметр для вывода протокола |

Параметр –in:\*.txt должен быть указан обязательно. В случае его отсутствия трансляция выполняться не будет.

Если какой-либо из других параметров задан не будет, то информация, соответствующая этому параметру, не будет выведена в файл протокола.

## Параметры, формируемые транслятором

Протоколы работы транслятора приведены в таблице 2.2.

Табл. 2.2 Протоколы транслятора

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| “log.txt” | Файл с протоколом работы транслятора языка программирования MOA-2018. |
| “asm.asm” | Файл с ассемблерным кодом. |

Файл “log.txt” содержит в себе краткую информацию об исходном коде на языке ShDA-2019.

## Разработка лексического анализатора

## Структура лексического анализатора

Структура лексического анализатора приведена на рисунке 3.1.

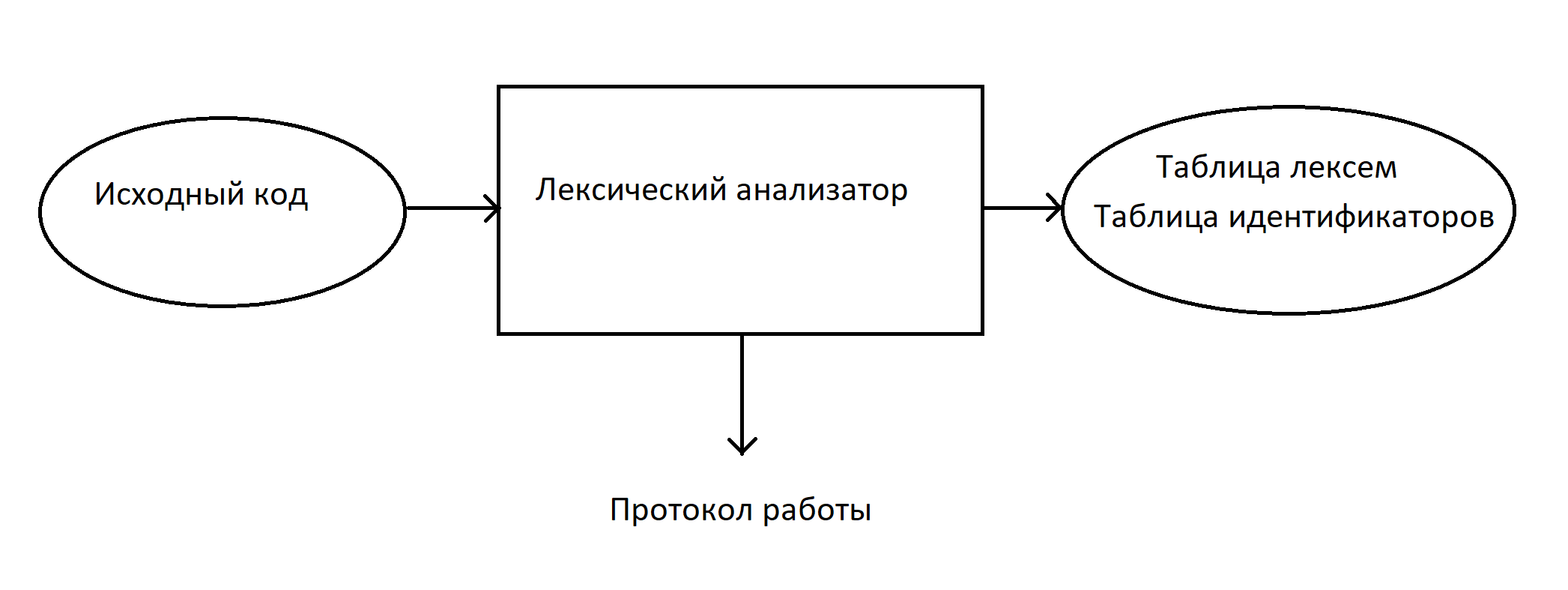


Рисунок 3.1. Структура лексического анализатора

Лексический анализатор – часть транслятора, выполняющая лексический анализ. На этом этапе распознаётся правильность составления лексем и идентификаторов.

Обязательным входным параметром для лексического анализаторов является файл с исходным кодом программы на языке программирования ShDA-2019, имеющий расширение .txt.

Результатом работы лексического анализатора является таблица лексем и идентификаторов, которые заполняются в процессе анализа.

## Контроль входных символов

Исходный код на языке программирования ShDA-2019 перед лексическим анализом проверяется на допустимость символов. Файл с исходным кодом считывается посимвольно и каждый символ проверяется на допустимость.

Для проверки символов была создана структура, аналогичная таблице кодировки Windows-1251. Данная структура представлена в приложении Б.

Каждый считанный из файла с исходным кодом символ проверяется на допустимость. Если символ в таблице обозначен как T, то он будет записан в строку, если символ обозначен буквой F, то его позиция в исходном коде записывается в структуру ошибок. В случае обнаружения недопустимого символа будет сгенерирована ошибка и она будет выведена в консоль вместе с позицией в исходном коде.

## Удаление избыточных символов

Избыточными в языке программирования ShDA-2019 считаются символы пробела, не влияющие на ход выполнения программы. Удаление избыточных символов осуществляется проверкой исходного кода. Если между словами больше одного пробела, то лишние пробелы будут удаляться.

## **Перечень ключевых слов**

Каждому токену соответствует особый символ-лексема. Соответствие токенов и лексем приведено в таблице 3.1.

Таблица 3.1. Таблица соответствия токенов и лексем

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Примечание |
| int | t | Беззнаковый целый тип данных |
| str | t | Строковый тип данных |
| var | а | Объявление переменной |
| def | d | Объявление функции |
| return | r | Возврат значения из функции |
| Main | m | Главная функция |
| out | o | Оператор вывода |
| if | f | Условный оператор |
| else | s |
| + | + | Арифметические операции |
| - | - |
| \* | \* |
| / | / |
| : | : |
| = | = | Операция присваивания |
|  | |
| Продолжение таблицы 3.1. | | |
| [ | [ | Блок условного оператора |
| ] | ] |
| { | { | Блок функции |
| } | } |
| ( | ( | Изменение приоритетности в выражениях и отделение параметров функции |
| ) | ) |
| ; | ; | Символы-сепараторы |
| , | , |

Для каждого токена соответствует автомат, по которому происходит разбор выражения. Автомат является детерменированным, то есть имеет конечное количество состояний.

## Основные структуры данных

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы, лексему, полученную при разборе, номер строки в исходном коде, является ли лексема арифметической операцией и номер в таблице идентификаторов, если лексема является идентификатором. Таблица идентификаторов содержит имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора, значение (или параметры функций), тип литерала, область видимости. Код C++ со структурой таблицы лексем и таблицы идентификаторов представлен в приложении Б.

## Структура и перечень сообщений лексического анализатора

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 200-299. Также при нахождении ошибки записывается строка кода в исходном файле, где была обнаружена ошибка. Ошибки, которые могут получиться в результате процесса лексического анализа, представлены в таблице 3.2.

Таблица 3.2. Ошибки лексического анализатора

|  |  |
| --- | --- |
| Ошибка | Расшифровка |
| 200 | Лексема не распознана |
| 201 | Превышен размер таблицы лексем |
| 202 | Превышен размер таблицы идентификаторов |
| 203 | Превышена длина строкового литерала |

## Принцип обработки ошибок

Если в процессе лексического анализа обнаруживается ошибка, то заполняется структура, содержащая ошибку, произошедшую в процессе лексического анализа, номер строки в исходном коде. Максимально допустимое количество ошибок в лексическом анализе ограничено числом 1. В случае обнаружения ошибки в консоль будет выведена информация об данной ошибке и дальнейший процесс будет прерван.

## Параметры лексического анализатора

В качестве входных параметров лексический анализатор принимает файл in с исходным кодом программы и файл lex, в который будет записываться исходный код замененный на лексемы – результат лексического анализа.

## Алгоритм лексического анализа

После чтения и редактирования входного файла лексический анализатор получает строку, в которой содержится токен.

Лексический анализатор пробует разобрать токен при помощи набора конечных автоматов. В случае, если автомат не был подобран, запоминается номер строки, в которой находился этот токен и в последствии будет выведено сообщение об ошибке. Если токен разобран, то дальнейшие действия, которые будут с ним производиться, будут зависеть от того, чем он является.

В случае, если токен является литералом, то заносится в таблицу идентификаторов с именем “l<m><n>”, где m - тип литерала, n - номер литерала.

Когда встречается токен, являющийся ключевым словом, которое отвечает за тип данных, заносится лексема, соответствующая ему, в таблицу лексем и запоминается тип данных, которому он соответствует. В последствии, когда встречается идентификатор, он заносится в таблицу идентификаторов с соответствующим ему типом данных и именем переменной.

## Контрольный пример

Часть результата работы лексического анализатора при разборе контрольного примера представлены в приложении В.

## Разработка синтаксического анализатора

## Структура синтаксического анализатора

Синтаксический анализатор – часть транслятора, выполняющая синтаксический анализ. Проверяется правильность расположения идентификаторов и ключевых слов в исходном коде. Для того, чтобы провести данную операцию, используются таблица лексем и идентификаторов. На выходе получается дерево разбора и протокол работы анализатора. Структура синтаксического анализатора представлена на рисунке 4.1.

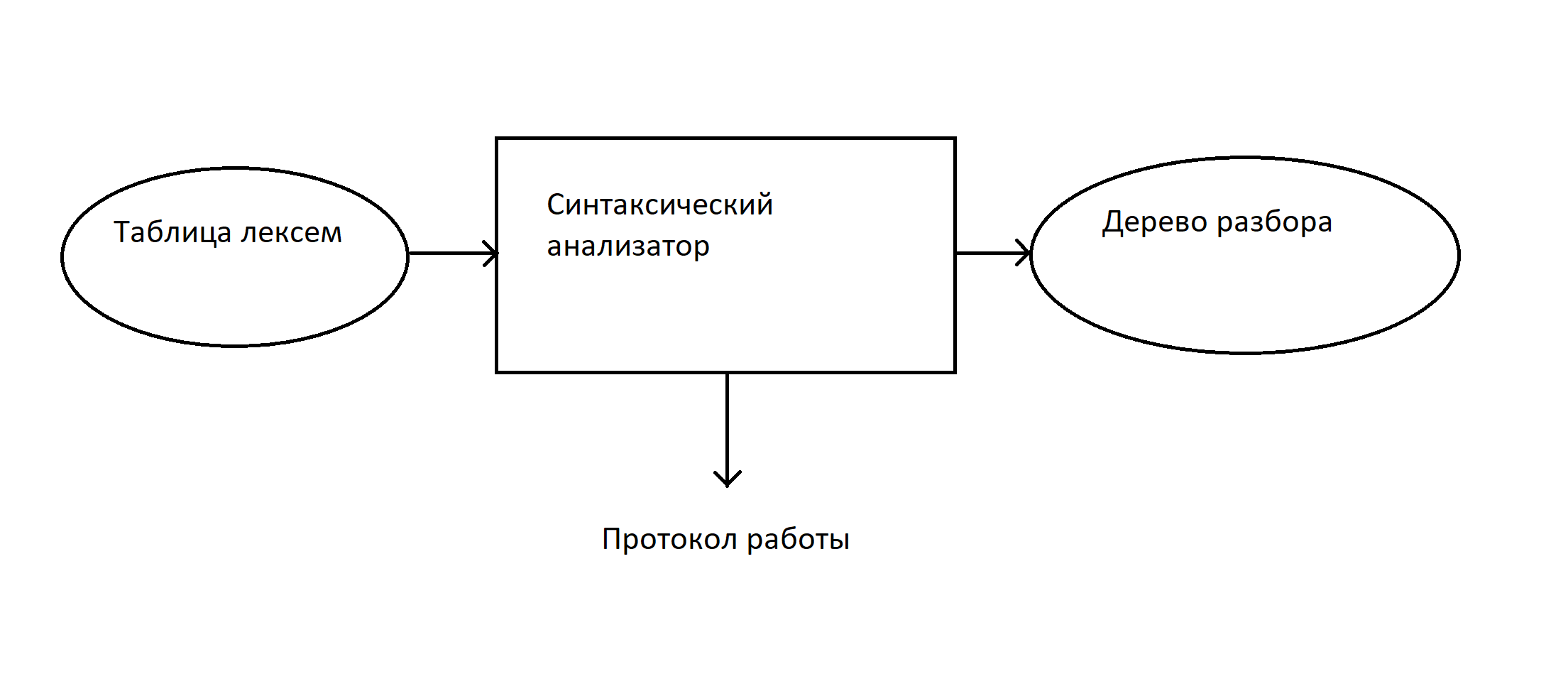


Рисунок 4.1. Структура синтаксического анализатора

## Контекстно свободная грамматика, описывающая синтаксис языка

Синтаксис языка ShDA-2019 описывается при помощи контекстно свободной грамматики.

Контекстно-свободная грамматика – грамматика типа 2 по иерархии Хомского. Данная грамматика имеет вид , где

T – множество терминальных символов,

N – множество нетерминальных символов,

P – множество правил переходов,

S – стартовый символ.

В контекстно-свободной грамматике правила имеют вид:

,

где

,

,

 - словарь грамматики .

Грамматика языка ShDA-2019 представлена в приложении Г в виде заполненной структуры на языке С++. В структуре терминальные символы помечены символами TS и все они являются сепараторами, знаками арифметических операций или строчными буквами. Нетерминальные символы помечены словом NS и являются заглавными буквами латинского алфавита.

## **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 компонент 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 таблицах 3.1 и 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (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 |
| Продолжение таблицы 4.2. | | |
|  | 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 |

## **Основные структуры данных**

Основные структуры данных синтаксического анализатора представлены в приложении Д.

## Описание алгоритма синтаксического разбора

Формальное описание алгоритма работы синтаксического анализа в общем виде можно описать следующим образом:

1. В магазин записывается стартовый символ.
2. На основе полученной ранее таблицы лексем формируется входная лента.
3. Запускается автомат.
4. Выбирается цепочка по первому символу, 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 магазина. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другое правило нетерминала.
6. Если в правиле 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.

## Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора содержится в таблице 4.1

Таблица 4.1 Сообщения синтаксического анализатора

|  |  |
| --- | --- |
| Номер сообщения | Содержание сообщения |
| Ошибка 601 | Неверная структура программы |
| Ошибка 602 | Неверно заданы параметры функции |
| Ошибка 603 | Неверно использованы системные конструкции |
| Ошибка 604 | Ошибка в параметрах, передаваемых в функцию |
| Ошибка 605 | Ошибка в выражении |
| Ошибка 606 | Выражение с арифметическими операциями составлено неверно |
| Ошибка 607 | Ошибка в условном операторе |

## Параметры синтаксического анализатора и режимы его работы

Входными данными являются таблицы лексем и идентификаторов. Выходными данными синтаксического анализатора являются дерево разбора и трассировка разбора исходного кода.

## Принцип обработки ошибок

Изначально, если анализатор разбирает часть исходного кода по какому-либо правилу, то запоминает его. При возникновении ошибки синтаксический анализатор откатиться назад до правила, при помощи которого разбор был успешным, если это возможно. После чего пытается применить последующие правила из грамматики. В случае, если правило невозможно подобрать, то выводится сообщение об ошибке.

## Контрольный пример

Пример разбора исходного кода на языке программирования ShDA-2019 синтаксическим анализатором представлен в виде дерева разбора, которое находится в приложении Е.

## Разработка семантического анализатора

## Структура семантического анализатора

Структура семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1. – Структура семантического анализатора

Семантический анализатор – часть транслятора, выполняющая cемантический анализ, то есть исходный код проверяется на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов.

Семантический анализатор является отдельной частью транслятора ShDA-2019

## Функции семантического анализатора

Сообщения семантического анализа ошибки, приведены на рисунке 5.1. Каждой ошибке соответствует функция, которая проверяет наличие данной ошибки в исходном коде языка. Функции вызываются параллельно с работой лексического анализа, а также после работы лексического анализа.

Таблица 5.1. Сообщения семантического анализатора

|  |  |
| --- | --- |
| Номер сообщения | Содержание сообщения |
| Ошибка 300 | Точка входа не обнаружена |
| Ошибка 301 | Обнаружено несколько точек входа |
| Продолжение таблицы 5.1. | |
| Ошибка 302 | Имя переменной не может быть ключевым словом |
| Ошибка 303 | В условном операторе обнаружена попытка объявления переменной |
| Ошибка 304 | Тип функции и возвращаемой переменной не совпадают |
| Ошибка 305 | Точка входа возвращает ненулевое значение |
| Ошибка 306 | Действия со строками недопустимы |
| Ошибка 307 | Неверные параметры |

## Принцип обработки ошибок

При обнаружении хотя бы одной семантической ошибки транслятор прекраитит свою работу.

## Вычисление выражений

## Выражения, допускаемые языком

В языке программирования ShDA-2019 допускается вычисление выражений со стандартными типами данных. В выражениях могут присутствовать операции, которые описаны в пункте 1.12.

Каждая операция в выражении имеет свой приоритет. Значения приоритетов представлены в таблице в таблице 6.1. (по убыванию)

Таблица 6.1. Приоритеты операций

|  |  |
| --- | --- |
| Операция | Приоритет |
| ( | 1 |
| ) |
| : | 2 |
| \* | 3 |
| - |
| + | 4 |
| - |

## Польская запись и принцип её простроения

Все выражения преобразовываются к обратной польской записи.

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

Алгоритм преобразования выражений к польской записи:

1. Считывется символ;
2. Если он является идентификатором или литералом, то символ добавляется к выходной строке;
3. Если символ является открывающей скобкой, то она помещается в стек;
4. Если символ является закрывающей скобкой, то выталкивается из стека в выходную строку все символы пока не встретится открывающая скобку. При этом обе скобки удаляются и не попадают в выходную строку;
5. Как только входная лента закончится все символы из стека выталкиваются в выходную строку;
6. В случае встречи операции выталкивается из стека в выходную строку все операции, которые имеют выше приоритетность чем последняя операция;

## Программная реализация обработки выражений

Программная реализация преобразования выражений к польской записи представлена в приложении Ж.

## Контрольный пример

Пример преобразования выражения в обратную польскую запись представлены в таблице 6.2.

Таблица 6.2. Преобразование выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Выходная строка | Стек |
| (i+i)\*l:i |  |  |
| i+i)\*l:i |  | ( |
| +i)\*l:i | i | ( |
| i)\*l:i | i | (+ |
| )\*l:i | ii | (+ |
| \*l:i | ii+ |  |
| l:i | ii+ | \* |
| :i | ii+l | \* |
| i | ii+l | \*: |
|  | ii+li | \*: |
|  | ii+li: | \* |
|  | ii+li:\* |  |

## Генерация кода

## Структура генератора кода

Структура генератора ассемблерного кода представлена на рисунке 7.1.

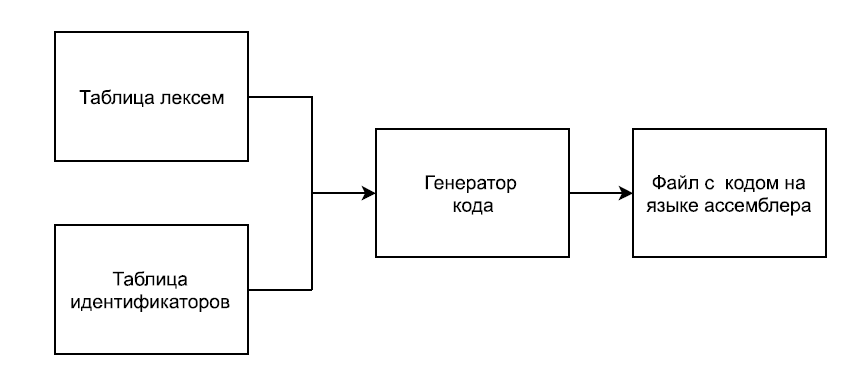


Рисунок 7.1. Структура генератора кода

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

## Представление типов данных в оперативной памяти

Данные в языке assembler представлены в разных сегментах: сегменте .const и сегменте .data. Идентификаторы языка программирования ShDA-2019 размещены в сегменте .data, литералы – в сегменте .const. Соответствие типов данных языка программирования ShDA-2019 и языка assembler приведены в таблице 7.2.

Переменные записываются в виде <Область видимости>\_<имя переменной>

Таблица 7.2. Соответствие типов переменных

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке программирования ShDA-2019 | Тип данных на языке assembler | Пояснение |
| int | SDWORD | Хранит целочисленный тип |
| Str | BYTE для литералов, SDWORD для переменных | Литерал хранит последовательность байтов.  Переменные хранят указатель на начало строки литерала. |

## Статическая библиотека

Статическая библиотека написана на языке программирования C++. Подробное описание функций приведено в пункте 1.18. Подключается библиотека с помощью команды includelib <имя.lib>. В процессе компоновки статическая библиотека подключается к выходному файлу.

## Алг**оритм работы генератора кода**

Преобразование происходит по принципу встречи определённой лексемы и знания местоположения данного кода в языке assembler.

На рисунке 7.2. приведён пример преобразования знака деления в язык assembler

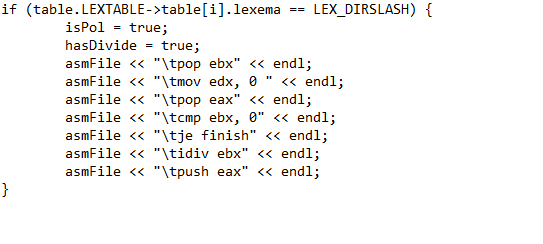


Рисунок 7.2. Преобразование знака деления в язык assembler

## Контрольный пример

Контрольный пример генерации кода представлен в приложении З.

## Тестирование транслятора

## Общие положения

В результате работы транслятора могут возникнуть ошибки, которые могут повлиять на транслятор и выполняемую программу. Смоделированные ошибки приведены в таблице 8.1.

Таблица 8.1. Ошибки и пути их разрешения

|  |  |  |
| --- | --- | --- |
| Код | Тип ошибки | Выходной результат |
| var 9in m; | Лексическая | 9inОшибка 200: Ошибка при разборе лексемы, строка 16, позиция 6 |
| z=((x+y)\*2):k | Синтаксическая | 605: строка 23, Ошибка в выражении |
| def  { | Семантическая | Ошибка 300: Отсутствует точка входа |

## Заключение

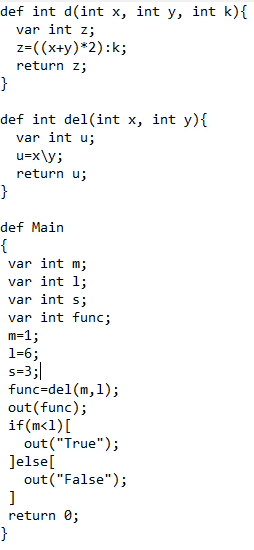
В ходе выполнения курсовой работы был разработан транслятор для языка программирования ShDA-2019. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка ShDA-2019;
2. Разработаны конечные автоматы и алгоритмы для реализация лексического анализатора;
3. Разработана контекстно-свободная, приведённая к ослабленной нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
4. Разработан семантический анализатор, осуществляющий проверку используемых инструкций;
5. Разработан транслятор с языка программирования ShDA-2019 на язык низкого уровня Assembler;
6. Проведено тестирование всех вышеперечисленных компонентов.

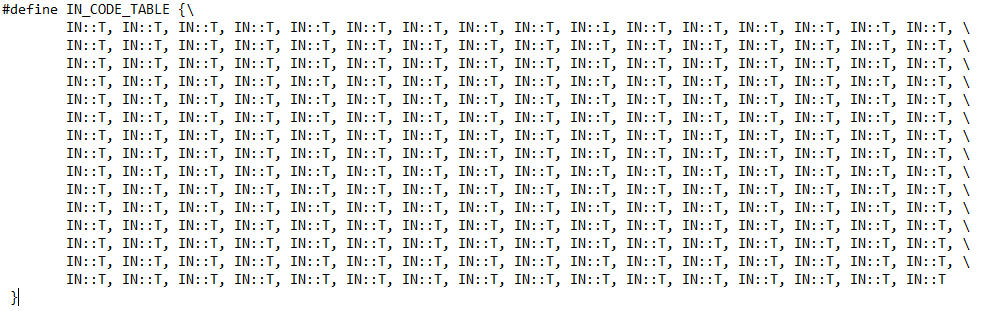
Окончательная версия языка ShDA-2019 включает:

1. 2 типа данных
2. поддержка операции вывода
3. возможность вызова функций стандартной библиотеки
4. наличие 5 арифметических операторов для вычисления выражений
5. Условный оператор
6. Структурированная система для обработки ошибок пользователя.
7. Поддержка функций

## Приложение А. Контрольный пример



## Приложение Б. Таблица символов



## Приложение В. Результат работы лексического анализатора

Таблица лексем:

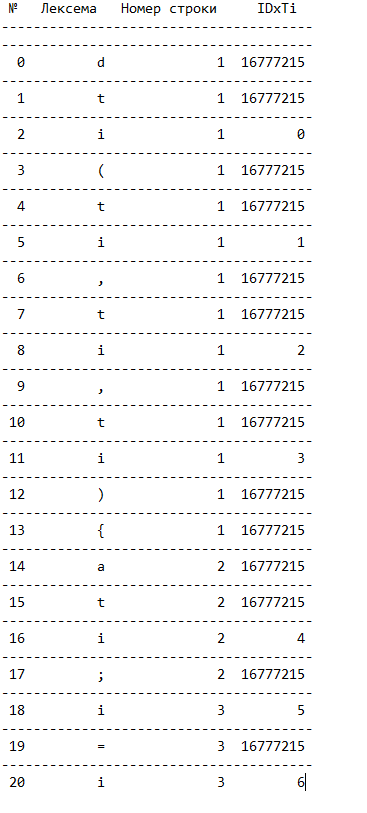
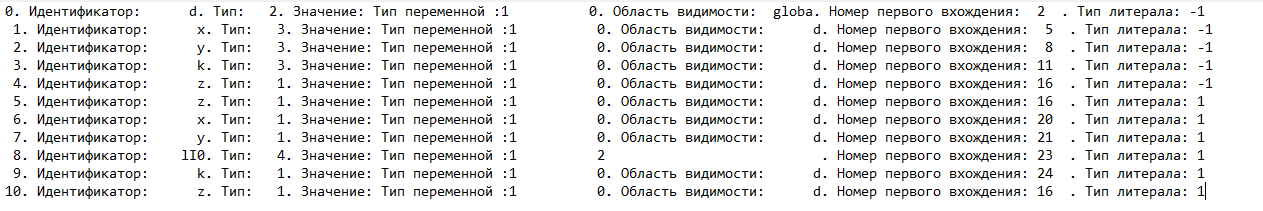
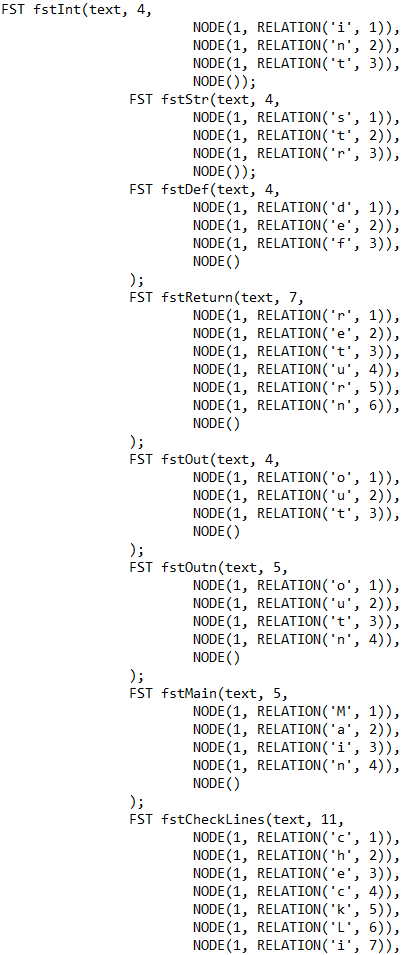
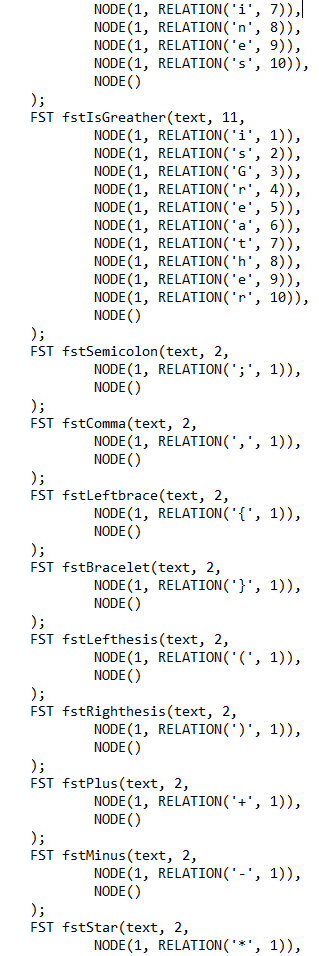


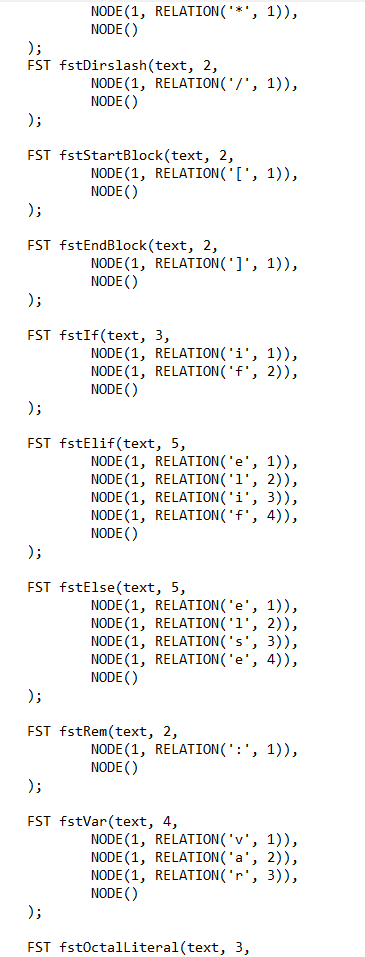
Таблица идентификаторов:

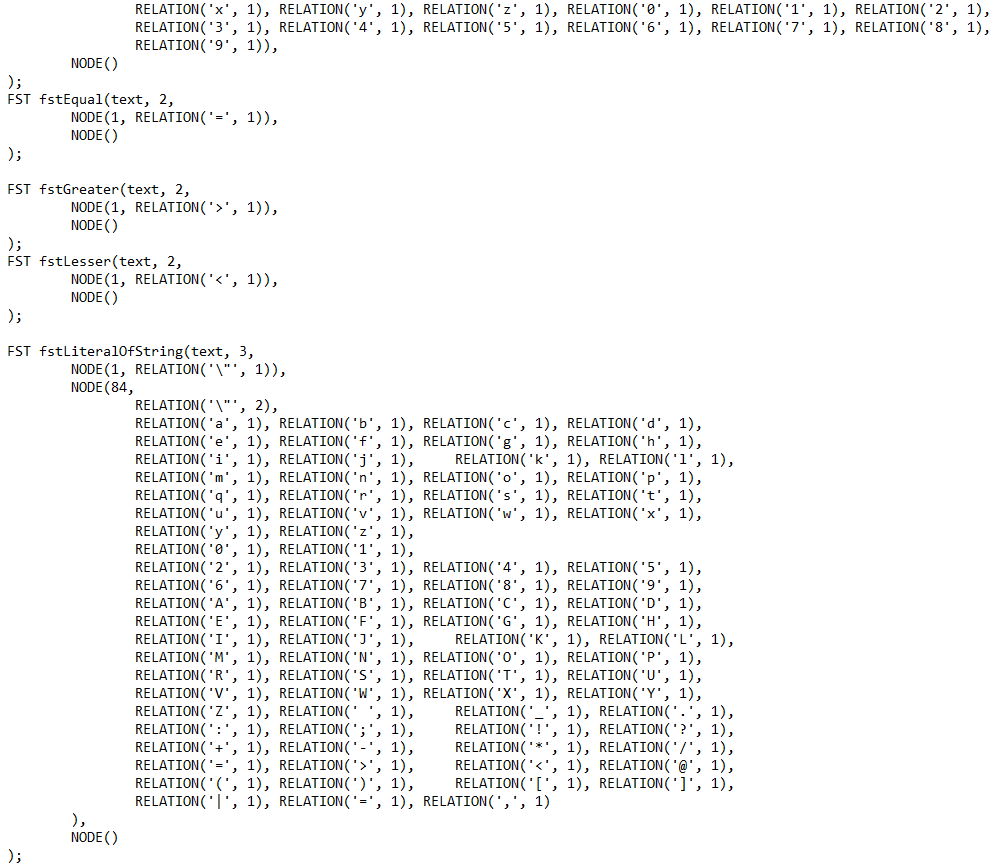
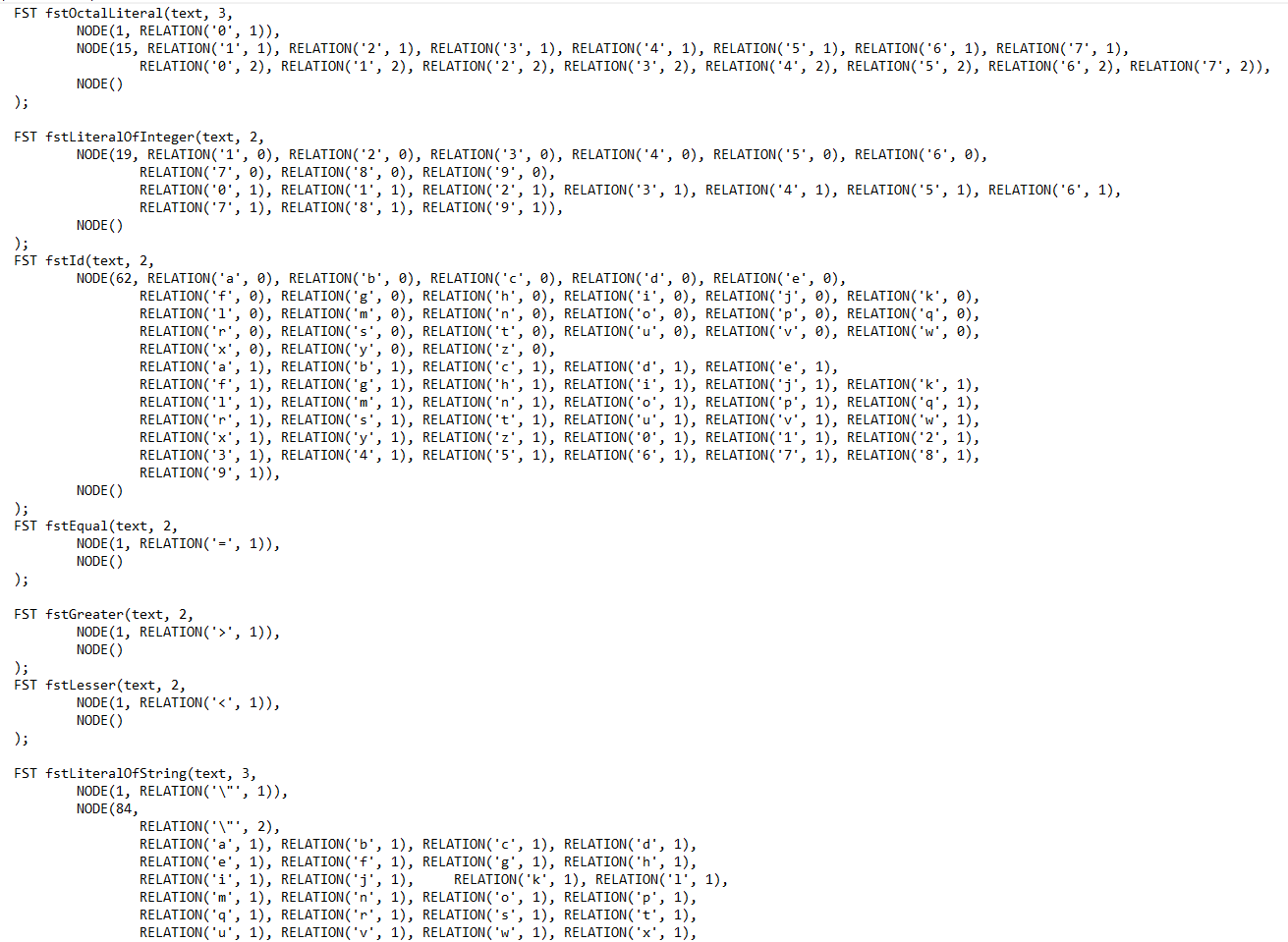


Конечные автоматы:

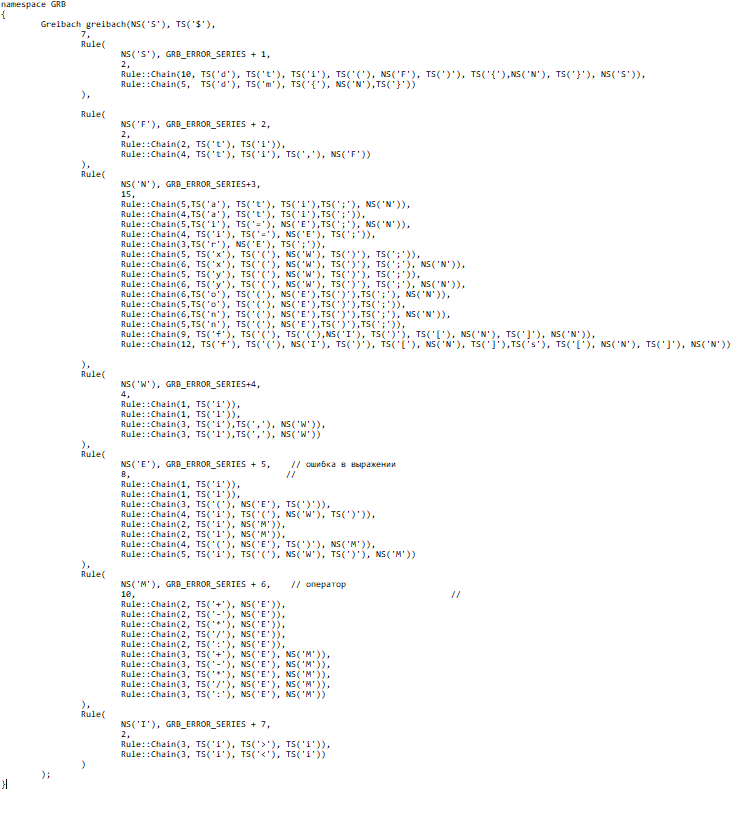








## Приложение Г. Правила синтаксического анализатора

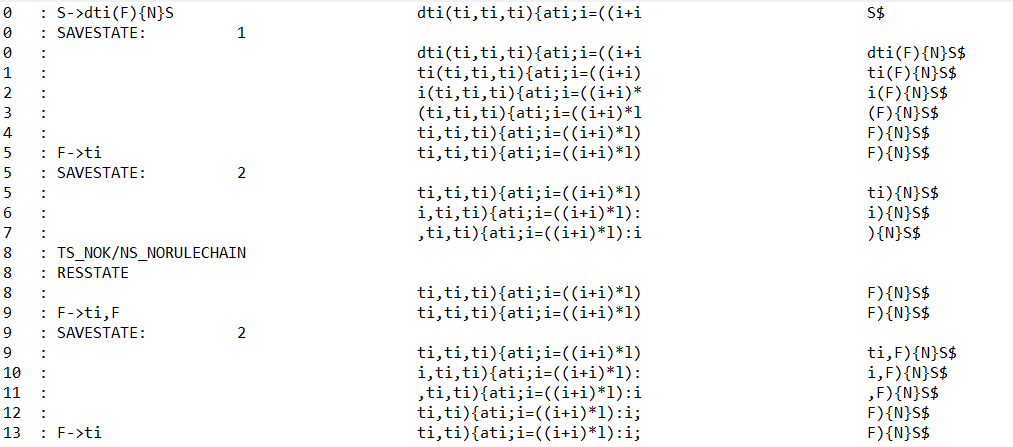


## Приложение Д. Структура синтаксического анализатора

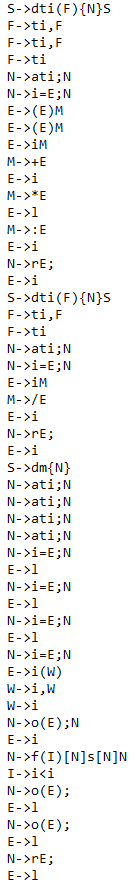


## Приложение Е. Результат работы синтаксического анализатора

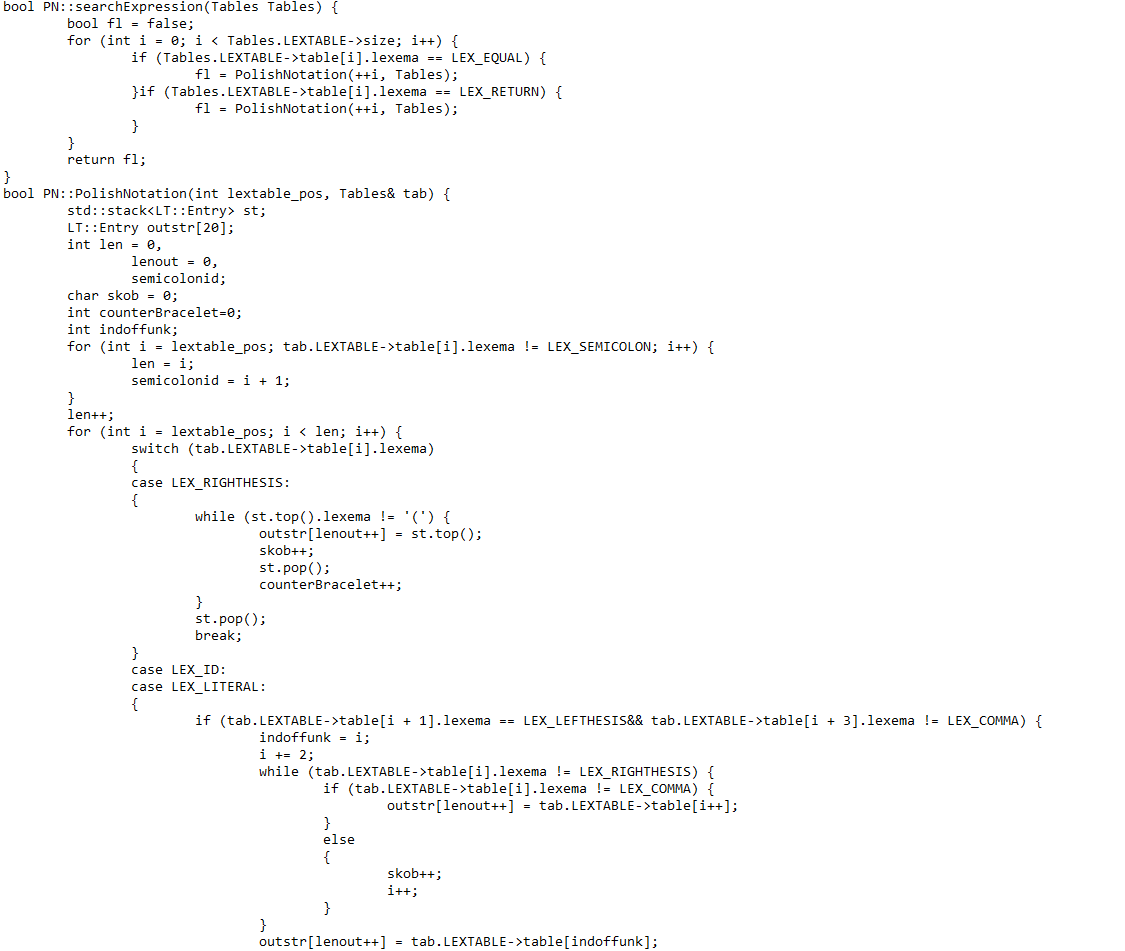
Часть синтаксического разбора

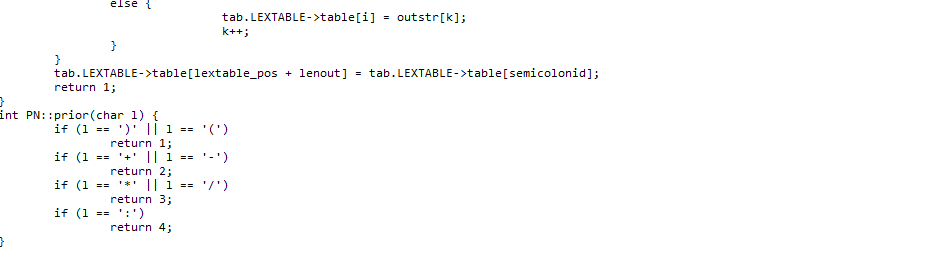


Правила разбора

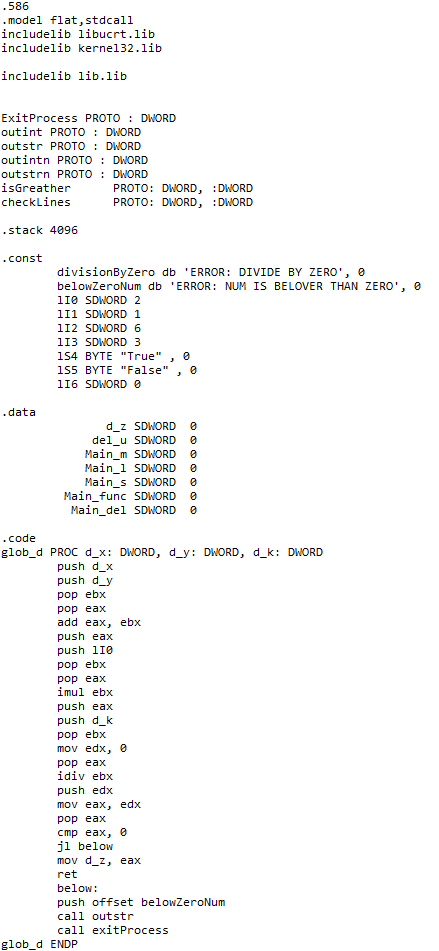


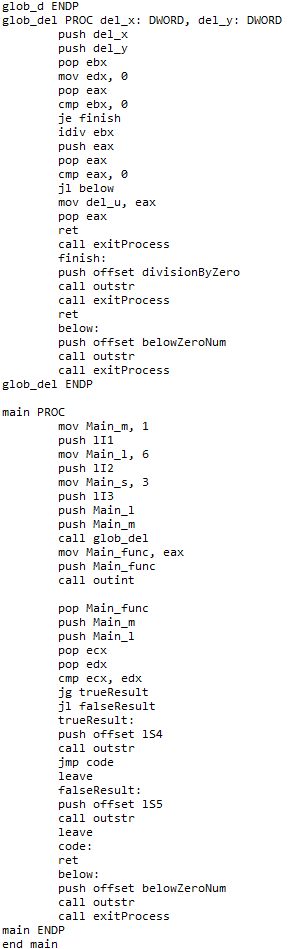
## Приложение Ж. Код программы, приводящей выражения к обратной польской записи

****

****

## Приложение 3. Генерация кода

****

****

## Литература

1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.

2. Смелов, В.В. Курс лекций по предмету “Языки программирования” – 2016

3. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с.