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 DMM-2017»

Выполнил студент Деликатная Маргарита Михайловна

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич А. С.

(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 А. С.

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

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

Нормоконтролер ст.пр. Наркевич А. С.

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

Курсовой проект защищен с оценкой

Минск 2017

Содержание

[Введение 5](#_Toc501701751)

[Глава 1 Спецификация языка программирования 6](#_Toc501701752)

[1.1 Характеристика языка программирования 6](#_Toc501701753)

[1.2 Алфавит языка 6](#_Toc501701754)

[1.3 Применяемые сепараторы 6](#_Toc501701755)

[1.4 Применяемые кодировки 7](#_Toc501701756)

[1.5 Типы данных 7](#_Toc501701757)

[1.6 Преобразование типов данных 7](#_Toc501701758)

[1.7 Идентификаторы 7](#_Toc501701759)

[1.8 Литералы 8](#_Toc501701760)

[1.9 Объявление данных и область видимости 8](#_Toc501701761)

[1.10 Инициализация данных 8](#_Toc501701762)

[1.11 Инструкции языка 8](#_Toc501701763)

[1.12 Операции языка 9](#_Toc501701764)

[1.13 Выражения и их вычисления 10](#_Toc501701765)

[1.14 Программные конструкции языка 10](#_Toc501701766)

[1.15 Область видимости идентификаторов 11](#_Toc501701767)

[1.16 Семантические проверки 11](#_Toc501701768)

[1.17 Распределение оперативной памяти на этапе выполнения 11](#_Toc501701769)

[1.18 Стандартная библиотека и ее состав 11](#_Toc501701770)

[1.19 Ввод и вывод данных 12](#_Toc501701771)

[1.20 Точка входа 12](#_Toc501701772)

[1.21 Препроцессор 12](#_Toc501701773)

[1.22 Соглашения о вызовах 12](#_Toc501701774)

[1.23 Объектный код 12](#_Toc501701775)

[1.24 Классификация сообщений транслятора 12](#_Toc501701776)

[1.25 Контрольный пример 13](#_Toc501701777)

[Глава 2 Структура транслятора 15](#_Toc501701778)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc501701779)

[2.2 Перечень входных параметров транслятора 16](#_Toc501701780)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 16](#_Toc501701781)

[Глава 3 Разработка лексического анализатора 17](#_Toc501701782)

[3.1 Структура лексического анализатора 17](#_Toc501701783)

[3.2 Контроль входных символов 17](#_Toc501701784)

[3.3 Удаление избыточных символов 18](#_Toc501701785)

[3.4 Перечень ключевых слов, сепараторов, символов операций соответствующим им лексемам 18](#_Toc501701786)

[3.5 Основные структуры данных 18](#_Toc501701787)

[3.6 Принцип обработки ошибок 19](#_Toc501701788)

[3.7 Структура и перечень сообщений лексического анализатора 19](#_Toc501701789)

[3.8 Параметры лексического анализатора и режимы его работы 20](#_Toc501701790)

[3.9 Алгоритм лексического анализа 20](#_Toc501701791)

[3.10 Контрольный пример 20](#_Toc501701792)

[Глава 4 Разработка синтаксического анализатора 21](#_Toc501701793)

[4.1 Структура Синтаксического анализатора 21](#_Toc501701794)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 21](#_Toc501701795)

[4.3 Построение конечного магазинного автомата 22](#_Toc501701796)

[4.4 Основные структуры данных 24](#_Toc501701797)

[4.5 Описание алгоритма синтаксического разбора 25](#_Toc501701798)

[4.6 Структура и перечень сообщений синтаксического анализатора 25](#_Toc501701799)

[4.7 Параметры синтаксического анализатора и режимы его работы 26](#_Toc501701800)

[4.8 Принцип обработки ошибок 26](#_Toc501701801)

[4.9 Контрольный пример 26](#_Toc501701802)

[Глава 5 Разработка семантического анализатора 27](#_Toc501701803)

[5.1 Структура семантического анализатора 27](#_Toc501701804)

[5.2 Функции семантического анализа 27](#_Toc501701805)

[5.3 Структура и перечень сообщений семантического анализатора 28](#_Toc501701806)

[5.4 Принцип обработки ошибок 28](#_Toc501701807)

[5.5 Контрольный пример 28](#_Toc501701808)

[Глава 6 Преобразование выражений 29](#_Toc501701809)

[6.1 Выражения, допускаемые языком 29](#_Toc501701810)

[6.2 Польская запись и принцип ее построения 29](#_Toc501701811)

[6.3 Программная реализация обработки выражений 29](#_Toc501701812)

[6.4 Контрольный пример 30](#_Toc501701813)

[Глава 7 Генерация кода 31](#_Toc501701814)

[Глава 8 Тестирование транслятора 33](#_Toc501701815)

[Заключение 36](#_Toc501701816)

[Приложение А 37](#_Toc501701817)

[Приложение Б 43](#_Toc501701818)

[Приложение В 44](#_Toc501701819)

[Приложение Г 47](#_Toc501701820)

[Приложение Д 49](#_Toc501701821)

[Приложение Е 51](#_Toc501701822)

[Приложение Ж 52](#_Toc501701823)

[Приложение З 53](#_Toc501701824)

[Список литературы 54](#_Toc501701825)

# Введение

Целью курсового проекта является разработка транслятора DMM-2017. Трансляция будет осуществляться в код на языке ассемблер.

Задачи для достижения цели:

-разбработка спецификации языка программирования (описана в главе «Спецификация языка программирования»);

-разбработка структуры транслятора (описана в главе «Структура транслятора»);

-разработка лексического анализатора (описана в главе «Разработка лексического анализатора»);

-разработка синтаксического анализатора (описана в главе «Разработка синтаксического анализатора»);

-разработка семантического анализатора (описана в главе «Разработка семантического анализатора»);

-обработка выражений (описана в главе «Преобразование выражений»);

-генерация кода на язык ассемблера (описана в главе «Генерация кода»);

-тестирование транслятора (описана в главе «Тестирование транслятора»);

Разработанный язык программирования позволяет работать с массивами, выполнять простейшие арифметические действия над целыми числами и операцию конкатенации над строками.

При написании данного курсового проекта были использованы источники из раздела «Список литературы».

# Глава 1 Спецификация языка программирования

# 1.1 Характеристика языка программирования

Язык программирования DMM-2017 – процедурный, универсальный язык. Строго типизированный, так как преобразование типов не допускается. Не является объектно-ориентированным языком. Компилируемый.

# 1.2 Алфавит языка

Символы, используемые для написания кода программы – латинские символы кодировки ASCII.

Допустимые символы:

<цифры> ::= 0 | 1 | 2 | … | 9

<буквы нижнего регистра> ::= a | b | … | z

<буквы верхнего регистра> ::= A | B | … | Z

<символы> ::= (пробел) | ! | " | # | $ | % | & | ' | ( | ) | \* | + | , | - | . | / | : | ; | < | = | > | ? | @ | [ | \ | ] | ^ | \_ | ` | { | ‘|’ | }

Иные символы недопустимы. Таблица приведена на рисунке 1.1.

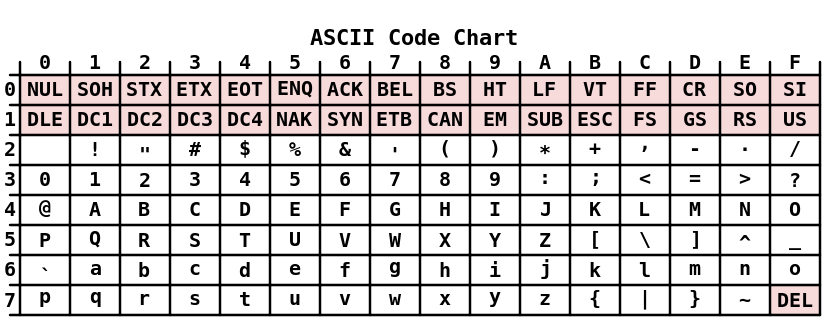


Рисунок 1.1 - Алфавит языка DMM-2017

# 1.3 Применяемые сепараторы

Сепараторы, или символы-разделители, применяемые в языке DMM-2017, приведены в таблице 1.1.

Таблица 1.1 - Сепараторы

|  |  |
| --- | --- |
| **Символы** | **Описание** |
| ( ) | Приоритетность операций |
| ( ) | Параметры |
| {} | Программный блок |
| +, -, \*, / | Символы бинарных операций |
| ; | Разделитель инструкций |
| Пробел | Допускается везде кроме идентификаторов и ключевых слов |

# 1.4 Применяемые кодировки

Кодировка символов, используемых для написания программы –стандартная кодировка ASCII (представлена на рисунке 1.1). Разрешены символы с кодом не менее чем 0x20 и не более чем 0x7D.

# 1.5 Типы данных

В данном языке предусмотрены целочисленные массивы, целочисленный и строковый типы данных, представленные в таблице 1.2.

Таблица 1.2 – Типы данных языка DMM-2017

| **Тип** | **Описание** |
| --- | --- |
| massiv | Одномерный массив целых чисел. Размерность элементов массива соответствует размерности целочисленного типа данных. Максимальное количество элементов массива – 1000. |
| entero | Целочисленный тип данных. Размерность – 4 байта, значения в интервале от –231+1 до 231-1. Операции: сложение, вычитание, умножение, деление. |
| cuerda | Строковый тип данных. Размерность символа в строке – 1 байт. Максимальная длина строки – 255 символов. Операции: конкатенация. |

# 1.6 Преобразование типов данных

Язык DMM-2017 не поддерживает преобразование типов данных, так как является строго типизированным.

# **1.7 Идентификаторы**

В языке DMM-2017 для именования идентификаторов разрешается использовать только латинские буквы нижнего регистра. Длина идентификатора усекается до 10 символов. Идентификаторы не могут совпадать с ключевыми словами.

<идентификатор> ::= <буквы нижнего регистра> (<буквы нижнего регистра>)\*

Примеры:

entero integer;

cuerda string;

massiv array[5];

# 1.8 Литералы

Литералы, которые поддерживаются в языке DMM-2017, описаны в таблице 1.3

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| **Тип** | **Описание** |
| Целочисленный | Числа от -231+1 до 231-1, интерпретируются как entero. Для записи используются цифры от 0(нуля) до 9(девяти) |
| Строковый | Строковые литералы, заключенные в ״ ״ (двойные кавычки), интерпретируются как cuerda. Используются символы латинского алфавита. Максимальная длина – 255. |

# 1.9 Объявление данных и область видимости

Область видимости языка DMM-2017 - сверху вниз. Функции объявляются глобально, а переменные могут быть объявлены только локально (объявляются внутри функции и недоступны снаружи неё). Параметры функции видны только внутри функции. Размерность массива при объявлении может быть задана только целочисленным литералом (тип entero).

# 1.10 Инициализация данных

Данные инициализируются по принципу:

declare <тип> <имя идентификатора> = <значение>;

<имя идентификатора> [идентификатор/литерал] = <значение>;

Примеры:

entero integer = 23;

cuerda string = “Hello”;

array[2]=3;

# 1.11 Инструкции языка

Язык DMM-2017 предусматривает инструкции, описанные в табл. 1.4.

Таблица 1.4 - Инструкции языка DMM-2017

|  |  |
| --- | --- |
| **Реализация** | **Описание** |
| declare <тип данных> <идентификатор>;  declare <тип данных> <идентификатор>[литерал]; | Объявление переменных. |
| <тип данных> function <идентификатор>(< тип данных> <идентификатор>,…); | Объявление функций. |

Продолжение таблицы 1.4

|  |  |
| --- | --- |
| declare <тип данных> <идентификатор> = <выражение>;  <идентификатор> = <выражение>;  declare <тип данных> <идентификатор>[литерал] = <выражение>;  <идентификатор>[идентификатор/литерал] = <выражение>; | Присвоение значений. |
| print<идентификатор>; | Вывод в стандартный поток вывода. |
| for(идентификатор/литерал); | Объявление цикла for |
| ret <выражение> | Возврат значения |

# 1.12 Операции языка

В языке DMM-2017 предусмотрены операции, описанные в таблице 1.5.

Таблица 1.5 - Операции языка

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| + | Суммирование (для entero), конкатенация (для cuerda) |
| - | Вычитание (для entero) |
| \* | Умножение (для entero) |
| / | Деление (для entero) |
| + | Унарный плюс |
| - | Унарный минус |

В таблице 1.6 перечислены приоритет и порядок операций языка. Операции перечислены сверху вниз в порядке убывания приоритета.

Таблица 1.6 - Приоритеты операций языка DMM-2017

|  |  |  |  |
| --- | --- | --- | --- |
| **Приоритет** | **Описание** | **Операция** | **Порядок** |
| 1 | Унарный плюс | + | Слева направо |
| Унарный минус | - |
| 2 | Умножение | \* | Слева направо |
| Деление | / |

Продолжение таблицы 1.6

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | Сумма | + | Слева направо |
| Разность | - |

Для изменения приоритета операций используется ().

# 1.13 Выражения и их вычисления

В языке DMM-2017 допускаются выражения:

- Арифметические, с применением операций +, -, /, \*;

- Строковые, с применением операции +.

Все выражения записываются в одну строку и могут содержать вызов функции. Использование круглых скобок () в выражениях изменяет приоритет операций. Допускается использование унарного минуса

# 1.14 Программные конструкции языка

Конструкции языка DMM-2017 описаны в таблице 1.7

Таблица 1.7 Конструкции языка DMM-2017

|  |  |
| --- | --- |
| **Конструкция** | **Реализация** |
| Точка входа | main()  {  <программный блок>  ret <выражение>  } |
| Функции | <тип данных> function <идентификатор>(  <тип данных> <идентификатор>, ….)  {  <программный блок функции>  ret <выражение>  } |
| Цикл | For(целочисленный литерал/целочисленный идентификатор)  {  <блок цикла for>  } |
| Параметры | передаются по значению |

Все функции должны содержать как минимум один параметр.

Параметрами пользовательской функции не могут быть массивы.

Массивы как параметр допускаются только в функциях стандартной библиотеки.

Программный блок цикла допускает наличие только операторов присвоения и печати.

# 1.15 Область видимости идентификаторов

Локальные идентификаторы объявляются внутри функции или в качестве параметра, доступны внутри текущей функции.

# 1.16 Семантические проверки

В языке DMM-2017 предусмотрен перечень семантических проверок, описанный в таблице 1.8

Таблица - 1.8 Семантические проверки языка DMM-2017

|  |
| --- |
| **Описание** |
| Проверка правильности составленного выражения |
| Проверка параметров функции при её определении |
| Проверка параметров функции при её вызове |
| Проверка правильности арифметического выражения |
| Проверка деления на ноль |
| Проверка на дублирование идентификаторов |
| Проверка на повторное определение функции |
| Проверка на дублирование параметров функции |
| Проверка на несоответствие типов в выражении |
| Проверка на превышение максимального количества параметров функции |
| Проверка на наличие массива в параметрах функции |
| Проверка на несоответствие типов в параметрах функции |
| Проверка на совпадение количества параметров функции |
| Проверка на соответствие типа возвращаемого значения функции |
| Проверка на выход за пределы массива |

# 1.17 Распределение оперативной памяти на этапе выполнения

В языке DMM-2017 предусмотрено следующее распределение памяти на этапе выполнения: код, локальные переменные. Все переменные размещаются в стек.

# 1.18 Стандартная библиотека и ее состав

Функции, входящие в стандартную библиотеку описаны в таблице 1.9

Таблица - 1.9 Функции стандартной библиотеки

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| entero max(massiv, entero) | Входные параметры – массив massiv и количество элементов массива entero. Функция вычисляет максимальный элемент массива из заданного числа элементов массива и возвращает целочисленный результат entero. |
| entero sum(massiv, entero) | Входные параметры – массив massiv и количество элементов массива entero. Функция вычисляет сумму заданного числа элементов массива и возвращает целочисленный результат entero. |

Стандартная библиотека подключается автоматически на этапе генерации кода.

# 1.19 Ввод и вывод данных

Вывод в стандартный поток вывода реализован в виде:

print<идентификатор>

# 1.20 Точка входа

Точка входа в языке DMM-2017 – функция main().

# 1.21 Препроцессор

В языке DMM-2017 наличие препроцессора не предусмотрено.

# 1.22 Соглашения о вызовах

В языке применяется соглашение о вызове \_\_cdecl (по умолчанию). Параметры справа налево, через стек. Стек очищает вызывающий код.

# 1.23 Объектный код

В языке DMM-2017 исходный код транслируется на язык ассемблера.

# 1.24 Классификация сообщений транслятора

Транслятор в ходе работы выдает сообщения об ошибке в соответствии с кодом ошибки. Классификация сообщений транслятора представлена в таблице 1.10.

Таблица 1.10 - Сообщения транслятора

|  |  |
| --- | --- |
| **Код ошибки** | **Описание** |
| 0-99 | Ошибки системы |
| 100-199 | Ошибки входного кода |
| 200-299 | Лексические ошибки |
| 300-399 | Семантические ошибки |
| 400-499 | Синтаксические ошибки |

# 1.25 Контрольный пример

entero function intstr(entero x, entero y, cuerda s)

{

print x;

declare cuerda string = "String /\";

print string;

string = s;

print string;

declare entero res;

declare entero parm = 10;

res=parm\*(x+y);

ret res;

};

entero function integ(entero x)

{

declare entero zero = 0;

declare entero dwx = 0;

declare massiv h[10];

for (10) {

h[zero] = 0;

zero = zero + 1;

};

zero = 0;

declare entero func = intstr(2, 3, "hello");

h[dwx] = func;

for (x) {

h[dwx] = h[dwx] + 1;

zero = h[dwx];

print zero;

dwx = dwx + 1;

};

ret zero;

};

cuerda function ab(cuerda x, cuerda y)

{

declare cuerda one = x;

declare cuerda two = y;

declare cuerda three = one + two;

ret three;

};

main

{

declare entero x = 5;

declare entero z = 0;

declare entero res = 0;

for (x) {

res = (z + 100)/(z+2);

z = z + 1;

print res;

};

declare entero a=integ(4);

declare entero t = 0;

declare massiv h[5];

for (5){

h[t] = (t\*3)/2;

x = h[t];

print x;

t = t + 1;

};

declare entero libfunction;

libfunction = sum(h, 5);

print libfunction;

libfunction = max(h, 5);

print libfunction;

declare entero y = 3;

declare massiv ar[10];

ar[0] = 15;

declare entero pam = 2;

ar[pam] = -6\*4;

y = y + ar[2];

print y;

declare cuerda forstr = ab("Hello ", "World!");

print forstr;

ret 0;

};

# Глава 2 Структура транслятора

# 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор – программа, которая преобразует исходный код на одном языке в исходный код на другом языке программирования.

Основные компоненты транслятора (представлены на рисунке 2.1):

- лексический анализатор

- синтаксический анализатор

- семантический анализатор

- генератор кода

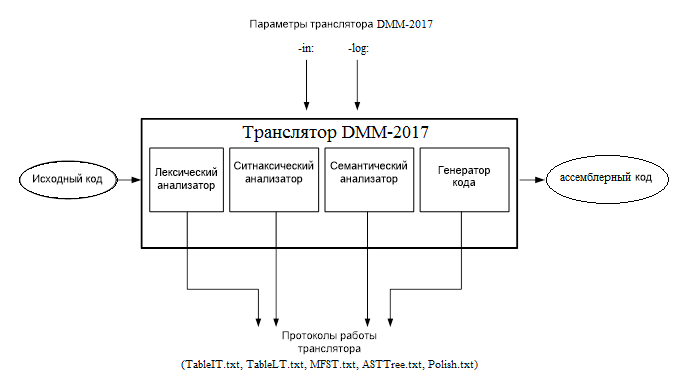


Рисунок 2.1 - Структура транслятора

Назначение лексического анализатора: текст исходной программы считывается посимвольно слева направо и группируется в отдельные токены, представляющие собой последовательности символов с определенным значением. В результате работы формируются таблица лексем и таблица идентификаторов, а также генерируются ошибки, связанные с лексикой данного языка. (Принцип работы описан в главе «Разработка лексического анализатора»)

Назначение синтаксического анализатора: сопоставляет линейные последовательности лексем из таблицы лексем с формальной грамматикой языка. Формирует полное дерево разбора и генерирует ошибки, если дерево построено не было. (Принцип работы описан в главе «Разработка синтаксического анализатора»)

Назначение семантического анализатора: проверяет наличие семантических ошибок в исходной программе и накапливает информацию для генератора кода. (Принцип работы описан в главе «Разработка семантического анализатора»)

Назначение генератора кода: генерирует ассемблерный код. При этом каждая инструкция транслируется в последовательность машинных инструкций, выполняющих ту же самую работу, а переменные назначаются регистрам. (Принцип работы описан в главе «Генерация кода»)

# 2.2 Перечень входных параметров транслятора

Перечень входных параметров транслятора представлен в таблице 2.1

Таблица 2.1 - Входные параметры языка DMM-2017

|  |  |
| --- | --- |
| **Входной параметр** | **Описание** |
| -in: | Параметр **–in** является обязательным и задает полное имя файла с исходным кодом. |
| -log: | Параметр **–log** является необязательным и задает полное имя файла протокола. В том случае, если параметр **–log** не задан, то используется имя файла, образованное от имени файла с исходным кодом (**-in**) путем добавления расширения **.log**. |

# 2.3 Перечень протоколов, формируемых транслятором и их содержимое

В ходе работы транслятор DMM-2017 автоматически формирует протоколы, в которых отображается ход выполнения трансляции. Назначение этих протоколов описано в таблице 2.2

Таблица 2.2 - Протоколы транслятора

|  |  |
| --- | --- |
| **Протокол** | **Назначение** |
| TableIT.txt | Вывод таблицы идентификаторов |
| TableLT.txt | Вывод таблицы лексем |
| MFST.txt | Вывод результата трассировки |
| ASTTree.txt | Вывод полного дерева разбора |
| Polish.txt | Вывод польской нотации |

# Глава 3 Разработка лексического анализатора

# 3.1 Структура лексического анализатора

На вход лексического анализатора поступает набор токенов, предварительно сформированных при считывании исходного файла. Результатом работы лексического анализатора являются сформированные таблицы лексем и идентификаторов. Структура лексического анализатора представлена на рисунке 3.1

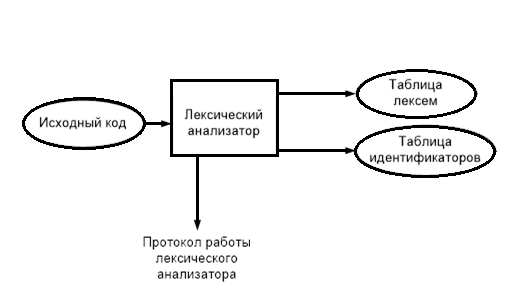


Рисунок 3.1 - Структура лексического анализатора

# 3.2 Контроль входных символов

Так как каждому символу ставится в соответствие числовой код таблицы ASCII, разрешенными являются те символы, значения которых в шестнадцатеричной системе счисления лежат в диапазоне от 0x20 до 0x7D.

Таблица входных символов представлена на рисунке 3.2, где F- запрещенные символы, T – разрешенные символы.

Если при считывании файла код символа превышает допустимый, генерируется ошибка 111 («недопустимый символ в исходном файле (-in)»).

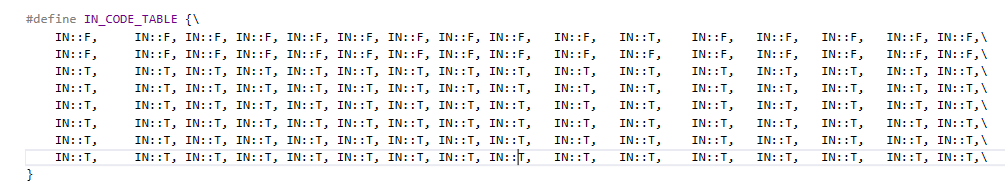


Рисунок 3.2 – Таблица входных символов

# 3.3 Удаление избыточных символов

Символы «пробел» являются избыточными и в процессе работы игнорируются.

# 3.4 Перечень ключевых слов, сепараторов, символов операций соответствующим им лексемам

Весь перечень представлен в таблице 3.1

Таблица 3.1

|  |  |  |
| --- | --- | --- |
| **Токен** | **Тип** | **Лексема** |
| main | Ключевое слово | m |
| for | Ключевое слово | c |
| entero | Ключевое слово | t |
| cuerda | Ключевое слово | t |
| function | Ключевое слово | f |
| ret | Ключевое слово | r |
| declare | Ключевое слово | d |
| print | Ключевое слово | p |
| massiv | Ключевое слово | a |
| Целочисленный литерал | Литерал | l |
| Строковый литерал | Литерал | l |
| Идентификатор | Идентификатор | i |
| + | Символ операции | + |
| - | Символ операции | - |
| \* | Символ операции | \* |
| / | Символ операции | / |
| = | Символ операции | = |
| ; | Сепаратор | ; |
| , | Сепаратор | , |
| { | Сепаратор | { |
| } | Сепаратор | } |
| ( | Сепаратор | ( |
| ) | Сепаратор | ) |
| [ | Сепаратор | [ |
| ] | Сепаратор | ] |

Описание конечных автоматов и графы переходов представлены в приложении А.

# 3.5 Основные структуры данных

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Данные структуры представлены в приложении Б.

Структура таблицы лексем представляет собой структуру C++ (struct LexTable), содержащую максимальный размер массива записей в таблице, текущее количество записей и сам массив таблицы лексем. Запись в таблице лексем (в дальнейшем – «лексема») представляет собой структуру C++ (struct Entry) содержащую символ лексемы, индекс в таблице идентификаторов, номер строки и позицию строки во входном файле, а также тип литерала, задаваемого данной лексемой.

Структура таблицы идентификаторов представляет собой структуру С++ (struct IdTable), содержащую максимальный размер массива записей в таблице, текущее количество записей и массив таблицы идентификаторов. Каждый идентификатор представляет собой структуру C++ (struct Entry), содержащую индекс первой строки в таблице лексем, имя идентификатора, константу из перечисления (enum IDDATATYPE), указывающую на тип данных идентификатора, константу из перечисления (enum IDTYPE), указывающую на тип идентификатора, область видимости идентификатора, количество параметров функции для идентификаторов функций, флаги, определяющие является ли идентификатор массивом или функцией стандартной библиотеки, массив параметров функций (для идентификаторов функций), а также присваиваемые значения (целочисленного и строкового типа данных).

# 3.6 Принцип обработки ошибок

При обнаружении ошибки в исходном коде программы, транслятор прекращает свою работу и выводит сообщение об ошибке с указанием номера строки и номера позиции, в которой произошла ошибка.

# 3.7 Структура и перечень сообщений лексического анализатора

Сообщения лексического анализатора включают в себя: номер ошибки, номер строки и номер позиции, в которых произошла ошибка, а также сообщение об ошибке.

Перечень сообщений, формируемых лексическим анализатором, представлен в таблице 3.2.

Таблица 3.2 - Таблица сообщений лексического анализатора

|  |  |
| --- | --- |
| **Код ошибки** | **Сообщение** |
| 213 | Неизвестная лексема во входном файле |
| 214 | Неизвестный идентификатор |
| 219 | Неверное значение целочисленного литерала (выход за пределы допустимого значения) |
| 220 | Превышена максимальная длина строкового литерала |

# 3.8 Параметры лексического анализатора и режимы его работы

В ходе своей работы лексический анализатор использует входные параметры транслятора -in: и -log:. Принцип их работы описан в таблице 2.1.

# 3.9 Алгоритм лексического анализа

Лексический анализатор посимвольно считывает исходный текст программы и проверяет символы на допустимость. Если код символа выходит за пределы интервала от 0х20 до 0х7D, генерируется ошибка.

Далее текст программы разбивается на отдельные токены, или цепочки символов, при этом избыточные символы игнорируются.

Информацию о функциях стандартной библиотеки заносится в таблицу идентификаторов.

Каждая цепочка проверяется при помощи алгоритма конечных автоматов. Если цепочка распознана, она заносится в таблицу лексем. Из полученной таблицы лексем выбираются токены, имеющие тип идентификатор, и на их основе формируется таблица идентификаторов. В процессе лексического анализа собирается некоторая информация об идентификаторах и заносится в таблицу. Если лексический анализ прошел успешно –результат работы выводится в отдельные файлы.

Пример удачного разбора идентификатора forstr из контрольного примера представлен на рисунке 3.3

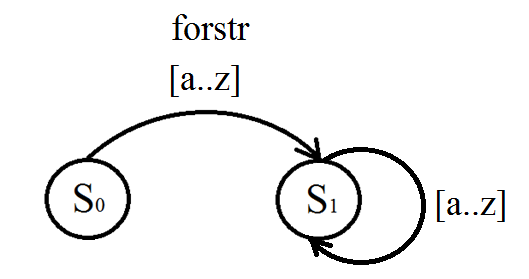


Рисунок 3.3 – Пример разбора

# 3.10 Контрольный пример

Таблица лексем и таблица идентификаторов, полученные в результате работы лексического анализатора, представлены в приложении В.

# Глава 4 Разработка синтаксического анализатора

# 4.1 Структура Синтаксического анализатора

Синтаксический анализатор принимает на вход таблицу лексем, сформированную лексическим анализатором, и проверяет, могут ли эти лексемы порождаться грамматикой языка. Он сообщает о выявленных ошибках в ходе проверки, и, если ошибки не были найдены, формирует полное дерево разбора и протокол работы.

Структура синтаксического анализатора представлена на рисунке 4.1

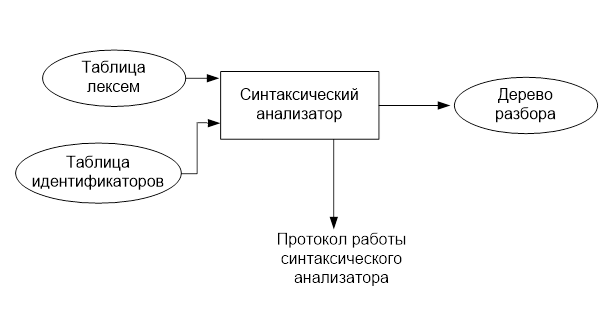


Рисунок 4.1 – Структура синтаксического анализатора

# 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

Грамматика языка DMM-2017 является контекстно-свободной грамматикой и представлена в нормальной форме Грейбах.

Контекстно-свободная грамматика  имеет нормальную форму Грейбах, если она не леворекурсивная (не содержит леворекурсивных правил), а правила  имеют вид:

, где ;

, где  — начальный символ, и если такое правило существует, то нетерминал  не должен встречаться в правой части правил.

Грамматика состоит из терминалов T, нетерминалов N, стартового символа S и множества правил P.

Терминалы представляют собой базовые символы, из которых формируются строки.

Нетерминалы представляют собой переменные, которые обозначают множества строк.

Стартовый символ – это один из нетерминалов грамматики. Множество строк, которые он обозначает, является языком, определяемым грамматикой.

Множества правил грамматики определяют способ, которым терминалы и нетерминалы могут объединяться для создания строк.

Перечень правил, описывавших контекстно-свободную грамматику языка DMM-2017, представлен в таблице 4.1. Каждая цепочка имеет уникальный идентификатор для упрощения работы с правилами языка.

Таблица 4.1 – Правила грамматики DMM-2017

|  |  |
| --- | --- |
| **Нетерминальный символ** | **Цепочки правил** |
| S  Стартовый символ | tfi(F){NrE;};S | m{NrE;}; |
| N  Операторы и конструкции | dai[l]; | dai[l];N  dti; | dti;N  dti=E; | dti=E;N  i=E; | i=E;N  i[l]=E; | i[l]=E;N  i[i]=E; | i[i]=E;N  pi; | pi;N  c(l){C}; | c(l){C};N  c(i){C}; | c(i){C};N |
| E  Выражения | -E | +E  i | iM  l | lM  (E) | (E)M  i(W) | i(W)M  i[l] | i[l]M  i[i] | i[i]M |
| F  Список параметров функции при её определении | ti | ti,F |
| W  Список параметров функции при вызове | i | i,W  l | l,W |
| M  Вспомогательный нетерминал для 'E', позволяет рекурсивно определять составные выражения | +E | -E | \*E | /E |
| C  Операторы внутри цикла | i=E; | i=E;N  i[l]=E; | i[l]=E;N  i[i]=E; | i[i]=E;N  pi; | pi;N |

# 4.3 Построение конечного магазинного автомата

Автомат с магазинной памятью (МП-автомат) - распознаватель контекстно-свободных языков.

Схема разбора МП-автоматом цепочки “dti;” представлена на рисунке 4.2:

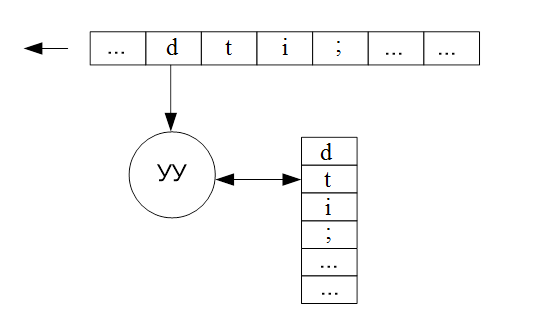
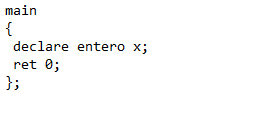


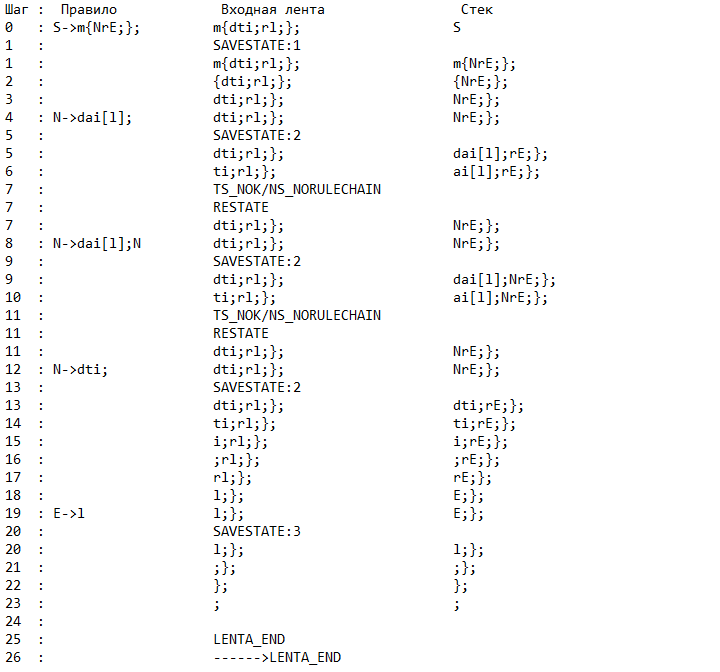
Рисунок 4.2 – структура МП-автомата

Формальное описание МП-автомата:

Конечный автомат с магазинной памятью представляет собой семерку , где  ⎯ множество состояний управляющего устройства,  ⎯ алфавит входных символов,  ⎯ специальный алфавит магазинных символов,  ⎯ функция переходов автомата,  ⎯ начальное состояние автомата,  ⎯ начальное состояние магазина (маркер дна);  ⎯ множество конечных состояний.

Пример разбора фрагмента кода:





# 4.4 Основные структуры данных

Основными структурами данных синтаксического анализатора являются структура грамматики Грейбах и структура конечного магазинного автомата. Структуры представлены в приложении Г.

Структура грамматики Грейбах представляет собой структуру C++(struct Greibach), содержащую количество правил языка, стартовый символ грамматики, дно стека символов грамматики, а также множество правил грамматики. Каждое правило представляет собой структуру C++(struct Rule), содержащую символ нетерминала, идентификатор диагностического сообщения и множество цепочек, описывающих правило. Цепочка является структурой (struct Chain) и содержит: уникальный идентификатор, длину цепочки и множество терминалов и нетерминалов.

Структура конечного магазинного автомата представляет собой структуру C++(struct Mfst), содержащую: константу из перечисления (enum RC\_STEP), указывающую на текущий статус автомата, структуру для сохранения диагностического сообщения, входную ленту (цепочку терминалов и нетерминалов), номер текущей позиции автомата, номер последнего используемого правила для нетерминала, номер последней использованной цепочки, размер входной ленты, номер самой дальней позиции, в которой произошла ошибка разбора, символ, который не был разобран и символ, который ожидался согласно определенному правилу, грамматику Грейбах, таблицу лексем, стек автомата и стек, в котором хранятся состояния автомата. Каждое состояние автомата – это структура (struct MfstState), которая содержит текущую позицию на ленте, номер цепочки текущего правила, стек автомата и номер правила.

# 4.5 Описание алгоритма синтаксического разбора

Синтаксический анализатор получает на вход ранее сформированные таблицы лексем и идентификаторов. В ленту помещаются символы из таблицы лексем, а набор правил грамматики – в стек. Каждый раз, когда встречается нетерминал, автомат сохраняет свое состояние. На каждом шаге автомата считывается один терминальный символ ленты и сравнивается с символом, лежащим на верхушке стека. Если символы совпадают, автомат продвигается на один символ вперед. Если же символы не совпадают, автомат восстанавливает свое предыдущее состояние и устанавливает в верхушку стека другое правило.

Если встречается нетерминальный символ, берется правило, соответствующее данному нетерминалу и продолжается разбор.

Если в результате разбора ни одно из правил не подошло – генерируется ошибка.

В случае, если в ленте заканчиваются символы и стек пуст, синтаксический анализ прошел успешно. В итоге будет сформирован набор правил, по которым были разобраны лексемы, и подготовлена информация для построения дерева разбора.

# 4.6 Структура и перечень сообщений синтаксического анализатора

В случае, если синтаксический анализ не был выполнен успешно, выводятся следующие сообщения из таблицы 4.2:

Таблица 4.2 – Перечень сообщений

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 301 | Ошибочный оператор |
| 305 | Синтаксическая ошибка в строке <номер строки> позиция <номер позиции>  Ожидалось <лексема правила грамматики>, встретилась <лексема исходного текста> |
| 306 | Размерность массива должна быть целочисленной |
| 308 | Неверный тип выражения для печати |
| 309 | Количество повторений цикла может быть только целочисленным |

Продолжение таблицы 4.2

|  |  |
| --- | --- |
| 315 | Обращение к массиву без индекса |
| 325 | Размерность массива не может быть больше 1000 |

# 4.7 Параметры синтаксического анализатора и режимы его работы

Специальные параметры для управления режимом работы синтаксического анализатора не предусмотрены.

# 4.8 Принцип обработки ошибок

При синтаксическом анализе перебираются все возможные правила грамматики. Если ни одно правило не подошло, генерируется соответствующая ошибка. Если же правила были найдены, делается проверка на оставшиеся ошибки. В случае обнаружения любой из ошибок, работа транслятора прекращается.

# 4.9 Контрольный пример

Результаты работы синтаксического анализатора – протокол разбора и дерево разбора – представлены в приложении Д.

Протокол разбора представляет собой вывод результата пошаговой работы конечного автомата с магазинной памятью.

В протокол выводится номер текущего шага, текущее рассматриваемое правило, входная лента и содержимое стека. Если на верхушке стека встречается нетерминал – сохраняется значение (SAVESTATE). Если рассматриваемая цепочка правила не была разобрана, выводится сообщение (TS\_NOK/NS\_NORULECHAIN) и восстанавливается предыдущее сохраненное состояние (RESTATE). Если ни одна из цепочек правила не была разобрана, восстанавливается состояние и выводится сообщение (TNS\_NORULECHAIN/NS\_NORULE). Если анализ прошел прошёл успешно, выводится сообщение (------>LENTA\_END), иначе выводятся сообщения ошибки разбора.

# Глава 5 Разработка семантического анализатора

# 5.1 Структура семантического анализатора

Структура семантического анализатора представлена на рисунке 5.1

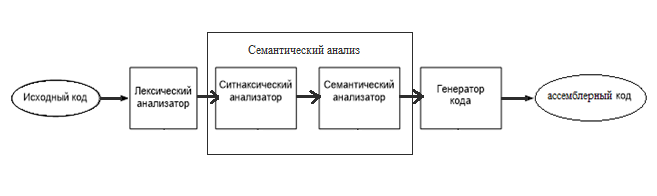


Рисунок 5.1

Используя информацию, полученную в результате работы синтаксического анализатора, семантический анализатор строит полное дерево разбора. Применяя леворекурсивный обход дерева, осуществляются проверки на наличие семантических ошибок. В случае, если ошибки не были выявлены, выполняется последний этап трансляции – генерация кода.

# 5.2 Функции семантического анализа

Функции, выполняющие различные семантические проверки, представлены в таблице 5.1

Таблица 5.1 – Функции семантического анализатора

|  |  |
| --- | --- |
| **Имя функции** | **Назначение** |
| checkFuncParams | Проверяет, совпадают ли типы параметров и их количество при вызове функции |
| assignCheck | Проверяет, совпадает ли тип выражения с типом идентификатора при использовании оператора присвоения, а также проверка на выход за пределы массива |
| declExpressionType | Проверяет, совпадают ли типы элементов, используемых в выражении |
| checkLiteral | Проверяет, совпадает ли тип литерала с типом идентификатора. Размерность массива и количество повторений в цикле должны быть целочисленными. |
| checkId | Проверяет наличие повторного определения функций или переменных, наличие функции, которая вызывается, верно ли использована функция в выражении, корректное описание идентификаторов |

# 5.3 Структура и перечень сообщений семантического анализатора

В случае, если семантический анализ не был выполнен успешно, выводятся следующие сообщения из таблицы 5.2:

Таблица 5.2 - Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| **Код ошибки** | **Сообщение** |
| 402 | Ошибка в выражении |
| 403 | Ошибка в параметрах функции |
| 404 | Ошибка в параметрах вызываемой функции |
| 405 | Ошибка в арифметическом выражении |
| 407 | Повторное определение идентификатора |
| 406 | Деление на ноль |
| 410 | Функция не может быть определена дважды |
| 411 | Неверное использование имени функции |
| 412 | Функция не найдена |
| 413 | Параметр задан повторно |
| 414 | Неподдерживаемый тип операций для строковых данных |
| 416 | Несовместимые типы в выражении |
| 417 | Превышено максимальное количество параметров функции |
| 419 | Фактический тип параметра функции не совпадает с заданным |
| 420 | Невозможна передача массива как параметра функции |
| 421 | Превышено число параметров функции |
| 422 | Недостаточное число параметров функции |
| 423 | Несовпадение возвращаемого типа функции |
| 424 | Выход за пределы массива |

# 5.4 Принцип обработки ошибок

. В результате обхода дерева разбора осуществляется поиск семантических ошибок. В случае, если семантический анализ не был выполнен успешно, генерируется соответствующая ошибка работа транслятора прекращается.

# 5.5 Контрольный пример

В приложении Е представлено три примера с ошибками, обрабатываемыми семантическим анализатором, а также сообщения, выдаваемые при обнаружении данных ошибок.

# Глава 6 Преобразование выражений

# 6.1 Выражения, допускаемые языком

В разработанном языке допускаются арифметические и строковые выражения. К арифметическим относятся выражения с операциями сложения, вычитания, умножения и деления. Строковые выражения используют только операцию конкатенации.

В языке предусмотрены унарные операторы плюс (+) и минус (-), которые имеют наивысший приоритет.

Приоритет операций представлен в таблице 1.7.  
Для изменения приоритета операций используется ().

Примерами выражений из контрольного примера являются:

one + two

(z + 100)/(z+2)

-6\*4

# 6.2 Польская запись и принцип ее построения

Польская нотация – это форма записи выражений, в которой операторы располагаются слева или справа от операндов. Характерной чертой такой записи является отсутствие скобок.

Принцип построения:

- Исходная строка(выражение) просматривается слева направо;

- Операнды переносятся в результирующую строку в порядке их следования;

- Операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;

- Операция выталкивает все операции с большим или равным приоритетом в результирующую строку;

- Отрывающая скобка помещается в стек;

- Закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;

- По концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

- Если результирующая строка меньше исходной – недостающие символы заполняем символами «%»

В языке DMM-2017 польская нотация не используется, все выражения вычисляются при обходе дерева.

# 6.3 Программная реализация обработки выражений

Фрагмент кода транслятора, реализующий преобразование выражений в польский формат, представлен в приложении Ж.

Основной функцией, осуществляющей преобразование, является функция PolishNotationRange, алгоритм которой соответствует основному принципу построения польской записи, рассмотренным в п.6.2. Функция принимает на вход три параметра: структуру, содержащую первую и последнюю лексемы для разбора, ссылки на таблицу лексем и идентификаторов.

Специальный параметр транслятора для вывода результатов преобразования не предусмотрен.

# 6.4 Контрольный пример

Результат преобразования контрольного примера в польскую запись представлен в приложении З. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов вычисления.

# Глава 7 Генерация кода

Трансляция исходного кода программы, написанного на языке DMM-2017, производится на язык ассемблера и осуществляется в процессе обхода полного синтаксического дерева.

При посещении определенной вершины данного дерева происходит генерация кода, отвечающая за сущность, описываемую данной вершиной (функция, выражение и т.д.)

Функции, отвечающие за генерацию кода, представлены в таблице 4.1

Таблица 4.1

|  |  |
| --- | --- |
| **Название функции** | **Назначение** |
| generate | Записывает в файл ассемблерного кода стандартный заголовок, глобальные переменные и импортируемые функции |
| generateFunction | Генерирует тело функции: записывает в файл заголовок функции, включающий в себя имя функции и список параметров; локальные переменные, объявленные внутри данной функции; генерирует и записывает все инструкции, находящиеся в теле функции при помощи вызова метода generateInstructions; записывает в файл инструкции, отвечающие за возврат значения. |
| generateLoopInstructions | Генерирует инструкции тела цикла, а именно инструкции присвоения и печати на экран. |
| generateInstructions | Генерирует инструкции тела функции. Для генерации тела цикла вызывается метод generateLoopInstructions. |
| computeExpression | Метод, вызываемый из generateLoopInstructions или generateInstructions для генерации кода вычисления выражений в инструкциях присвоения |

Реализация некоторых инструкций языка DMM-2017 представлена в таблице 4.2

Таблица 4.2

|  |  |
| --- | --- |
| **Инструкция** | **Реализация** |
| Объявление переменных. | Переменные объявляются локально внутри функции:  LOCAL <имя переменной>:<тип данных>  LOCAL <имя переменной>[число]:<тип данных> |
| Объявление функций. | Пользовательские функции:  <имя функции> PROC <название переменной>:<тип данных>,…  Главная функция:  main PROC |
| Присвоение значений. | Для целых чисел с помощью операции MOV  Для строк с применением функции lstrcpy |

Продолжение таблицы 4.2

|  |  |
| --- | --- |
| Вывод в стандартный поток вывода. | Осуществляется с помощью функции WriteConsoleA.  Для вывода целых чисел:  INVOKE dwtoa, ebx, addr buffer1  INVOKE lstrlen, addr buffer1  mov edx, offset buffer1  INVOKE WriteConsoleA, consoleOutHandle1, edx, eax, offset bytesWritten1, 0  Для вывода строк:  INVOKE lstrlen, addr <идентификатор>  INVOKE WriteConsoleA, consoleOutHandle1, addr <идентификатор>, eax, offset bytesWritten1, 0 |
| Цикл for | В регистр ecx помещаем значение числа повторений цикла.  .while ecx > 0  <тело цикла>  dec ecx  .endw |
| Обращение к элементу массива | Каждый элемент массива занимает 4 байта.  Если индекс – целочисленный литерал:  PUSH <имя массива>[<индекс>\*4]  Если индекс – идентификатор:  MOV eax, <идентификатор>  MOV ebx, 4  MUL ebx  PUSH < имя массива >[eax] |
| Вычисление выражений | Сложение:  POP ebx  POP eax  ADD eax, ebx  PUSH eax  Вычитание:  SUB eax, ebx  PUSH eax  Умножение:  POP ebx  POP eax  MUL ebx  PUSH eax  Деление:  DIV ebx  PUSH eax  Конкатенация:  Используется функция lstrcat |
| Возврат значения | Сохраняем результат в регистре eax  RET |

# Глава 8 Тестирование транслятора

На разных этапах трансляции исходный код проходит проверку на наличие соответствующих ошибок. При обнаружении ошибки работа транслятора прекращается и выводится соответствующее сообщение об ошибке.

Результаты тестирования лексического анализатора представлены в таблице 8.1

Таблица 8.1

|  |  |
| --- | --- |
| **Пример кода с ошибкой** | **Сообщение транслятора** |
|  |  |
|  |  |
|  |  |
|  |  |

Результаты тестирования синтаксического анализатора представлены в таблице 8.2

Таблица 8.2

|  |  |
| --- | --- |
| **Пример кода с ошибкой** | **Сообщение транслятора** |
|  |  |

Продолжение таблицы 8.2

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

Результаты тестирования семантического анализатора представлены в таблице 8.3

Таблица 8.3

|  |  |
| --- | --- |
| **Пример кода с ошибкой** | **Сообщение транслятора** |
|  |  |
|  |  |
|  |  |

Продолжение таблицы 8.3

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

# Заключение

Результатом выполнения курсового проекта является разработанный транслятор для языка DMM-2017. Данный язык включает в себя:

1. 3 типа данных;
2. Операцию вывода на экран print;
3. Цикл for;
4. 2 функции стандартной библиотеки;
5. 4 арифметических оператора для вычисления выражений;

Генерация кода осуществляется на язык ассемблера.

Таким образом, все поставленные задачи курсового проекта выполнены.

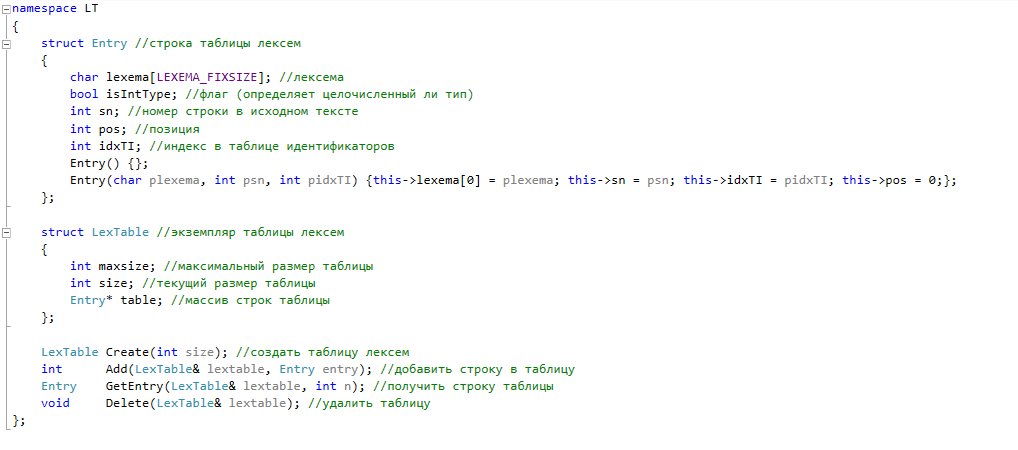
# Приложение А

|  |  |
| --- | --- |
| **Лексема** | **Соответствующий граф и КА** |
| m | #define GRAPH\_MAIN 5, \  Fst::NODE(1,Fst::RELATION('m',1)),\  Fst::NODE(1,Fst::RELATION('a',2)),\  Fst::NODE(1,Fst::RELATION('i',3)),\  Fst::NODE(1,Fst::RELATION('n',4)),\  Fst::NODE() |
| c | #define GRAPH\_FOR 4, \  Fst::NODE(1,Fst::RELATION('f',1)),\  Fst::NODE(1,Fst::RELATION('o',2)),\  Fst::NODE(1,Fst::RELATION('r',3)),\  Fst::NODE() |
| t | #define GRAPH\_INTEGER 7, \  Fst::NODE(1,Fst::RELATION('e',1)),\  Fst::NODE(1,Fst::RELATION('n',2)),\  Fst::NODE(1,Fst::RELATION('t',3)),\  Fst::NODE(1,Fst::RELATION('e',4)),\  Fst::NODE(1,Fst::RELATION('r',5)),\  Fst::NODE(1,Fst::RELATION('o',6)),\  Fst::NODE() |
| t | #define GRAPH\_STRING 7, \  Fst::NODE(1,Fst::RELATION('c',1)),\  Fst::NODE(1,Fst::RELATION('u',2)),\  Fst::NODE(1,Fst::RELATION('e',3)),\  Fst::NODE(1,Fst::RELATION('r',4)),\  Fst::NODE(1,Fst::RELATION('d',5)),\  Fst::NODE(1,Fst::RELATION('a',6)),\  Fst::NODE() |
| f | #define GRAPH\_FUNCTION 9, \  Fst::NODE(1, Fst::RELATION('f', 1)),\  Fst::NODE(1, Fst::RELATION('u', 2)),\  Fst::NODE(1, Fst::RELATION('n', 3)),\  Fst::NODE(1, Fst::RELATION('c', 4)),\  Fst::NODE(1, Fst::RELATION('t', 5)),\  Fst::NODE(1, Fst::RELATION('i', 6)),\  Fst::NODE(1, Fst::RELATION('o', 7)),\  Fst::NODE(1, Fst::RELATION('n', 8)),\  Fst::NODE() |
| r | #define GRAPH\_RETURN 4, \  Fst::NODE(1,Fst::RELATION('r',1)),\  Fst::NODE(1,Fst::RELATION('e',2)),\  Fst::NODE(1,Fst::RELATION('t',3)),\  Fst::NODE() |
| d | #define GRAPH\_DECLARE 8, \  Fst::NODE(1, Fst::RELATION('d', 1)),\  Fst::NODE(1, Fst::RELATION('e', 2)),\  Fst::NODE(1, Fst::RELATION('c', 3)),\  Fst::NODE(1, Fst::RELATION('l', 4)),\  Fst::NODE(1, Fst::RELATION('a', 5)),\  Fst::NODE(1, Fst::RELATION('r', 6)),\  Fst::NODE(1, Fst::RELATION('e', 7)),\  Fst::NODE() |
| p | #define GRAPH\_PRINT 6, \  Fst::NODE(1, Fst::RELATION('p', 1)),\  Fst::NODE(1, Fst::RELATION('r', 2)),\  Fst::NODE(1, Fst::RELATION('i', 3)),\  Fst::NODE(1, Fst::RELATION('n', 4)),\  Fst::NODE(1, Fst::RELATION('t', 5)),\  Fst::NODE() |
| a | #define GRAPH\_ARRAY 7, \  Fst::NODE(1, Fst::RELATION('m', 1)),\  Fst::NODE(1, Fst::RELATION('a', 2)),\  Fst::NODE(1, Fst::RELATION('s', 3)),\  Fst::NODE(1, Fst::RELATION('s', 4)),\  Fst::NODE(1, Fst::RELATION('i', 5)),\  Fst::NODE(1, Fst::RELATION('v', 6)),\  Fst::NODE() |
| l | #define GRAPH\_INT\_LITERAL 2, \  Fst::NODE(10,Fst::RELATION('0',1),Fst::RELATION('1',1),\  Fst::RELATION('3',1),Fst::RELATION('2',1),\  Fst::RELATION('5',1),Fst::RELATION('4',1),\  Fst::RELATION('7',1),Fst::RELATION('6',1),\  Fst::RELATION('8',1),Fst::RELATION('9',1)),\  Fst::NODE(10,Fst::RELATION('0',1),Fst::RELATION('1',1),\  Fst::RELATION('3',1),Fst::RELATION('2',1),\  Fst::RELATION('5',1),Fst::RELATION('4',1),\  Fst::RELATION('7',1),Fst::RELATION('6',1),\  Fst::RELATION('8',1),Fst::RELATION('9',1)) |
| l | #define GRAPH\_STRING\_LITERAL 3, \  Fst::NODE(1, Fst::RELATION('"',1)),\  Fst::NODE(96,Fst::RELATION(32,1),Fst::RELATION(33,1),Fst::RELATION(34,1),Fst::RELATION(35,1),Fst::RELATION(36,1),Fst::RELATION(37,1),\  Fst::RELATION(38,1),Fst::RELATION(39,1),Fst::RELATION(40,1),Fst::RELATION(41,1),Fst::RELATION(42,1),Fst::RELATION(43,1),\  Fst::RELATION(44,1),Fst::RELATION(45,1),Fst::RELATION(46,1),Fst::RELATION(47,1),Fst::RELATION(48,1),Fst::RELATION(49,1),\  Fst::RELATION(50,1),Fst::RELATION(51,1),Fst::RELATION(52,1),Fst::RELATION(53,1),Fst::RELATION(54,1),Fst::RELATION(55,1),\  Fst::RELATION(56,1),Fst::RELATION(57,1),Fst::RELATION(58,1),Fst::RELATION(59,1),Fst::RELATION(60,1),Fst::RELATION(61,1),\  Fst::RELATION(62,1),Fst::RELATION(63,1),Fst::RELATION(64,1),Fst::RELATION(65,1),Fst::RELATION(66,1),Fst::RELATION(67,1),\  Fst::RELATION(68,1),Fst::RELATION(69,1),Fst::RELATION(70,1),Fst::RELATION(71,1),Fst::RELATION(72,1),Fst::RELATION(73,1),\  Fst::RELATION(74,1),Fst::RELATION(75,1),Fst::RELATION(76,1),Fst::RELATION(77,1),Fst::RELATION(78,1),Fst::RELATION(79,1),\  Fst::RELATION(80,1),Fst::RELATION(81,1),Fst::RELATION(82,1),Fst::RELATION(83,1),Fst::RELATION(84,1),Fst::RELATION(85,1),\  Fst::RELATION(86,1),Fst::RELATION(87,1),Fst::RELATION(88,1),Fst::RELATION(89,1),Fst::RELATION(90,1),Fst::RELATION(91,1),\  Fst::RELATION(92,1),Fst::RELATION(93,1),Fst::RELATION(94,1),Fst::RELATION(95,1),Fst::RELATION(96,1),Fst::RELATION(97,1),\  Fst::RELATION(98,1),Fst::RELATION(99,1),Fst::RELATION(100,1),Fst::RELATION(101,1),Fst::RELATION(102,1),Fst::RELATION(103,1),\  Fst::RELATION(104,1),Fst::RELATION(105,1),Fst::RELATION(106,1),Fst::RELATION(107,1),Fst::RELATION(108,1),Fst::RELATION(109,1),\  Fst::RELATION(110,1),Fst::RELATION(111,1),Fst::RELATION(112,1),Fst::RELATION(113,1),Fst::RELATION(114,1),Fst::RELATION(115,1),\  Fst::RELATION(116,1),Fst::RELATION(117,1),Fst::RELATION(118,1),Fst::RELATION(119,1),Fst::RELATION(120,1),Fst::RELATION(121,1),\  Fst::RELATION(122,1),Fst::RELATION(123,1),Fst::RELATION(124,1),Fst::RELATION(125,1),Fst::RELATION(126,1),\  Fst::RELATION('"',2)),\  Fst::NODE() |
| i | #define GRAPH\_ID 2, \  Fst::NODE(26, Fst::RELATION('a',1), Fst::RELATION('b',1), \  Fst::RELATION('c', 1), Fst::RELATION('d', 1), \  Fst::RELATION('e', 1), Fst::RELATION('f', 1), \  Fst::RELATION('g', 1), Fst::RELATION('h', 1), \  Fst::RELATION('i', 1), Fst::RELATION('j', 1), \  Fst::RELATION('k', 1), Fst::RELATION('l', 1), \  Fst::RELATION('m', 1), Fst::RELATION('n', 1), \  Fst::RELATION('o', 1), Fst::RELATION('p', 1), \  Fst::RELATION('q', 1), Fst::RELATION('r', 1), \  Fst::RELATION('s', 1), Fst::RELATION('t', 1), \  Fst::RELATION('u', 1), Fst::RELATION('v', 1), \  Fst::RELATION('w', 1), Fst::RELATION('x', 1), \  Fst::RELATION('y', 1), Fst::RELATION('z', 1)), \  Fst::NODE(26, Fst::RELATION('a',1), Fst::RELATION('b',1), \  Fst::RELATION('c', 1), Fst::RELATION('d', 1), \  Fst::RELATION('e', 1), Fst::RELATION('f', 1), \  Fst::RELATION('g', 1), Fst::RELATION('h', 1), \  Fst::RELATION('i', 1), Fst::RELATION('j', 1), \  Fst::RELATION('k', 1), Fst::RELATION('l', 1), \  Fst::RELATION('m', 1), Fst::RELATION('n', 1), \  Fst::RELATION('o', 1), Fst::RELATION('p', 1), \  Fst::RELATION('q', 1), Fst::RELATION('r', 1), \  Fst::RELATION('s', 1), Fst::RELATION('t', 1), \  Fst::RELATION('u', 1), Fst::RELATION('v', 1), \  Fst::RELATION('w', 1), Fst::RELATION('x', 1), \  Fst::RELATION('y', 1), Fst::RELATION('z', 1)) |
| + | #define GRAPH\_ARYTHM\_PLUS 2,\  Fst::NODE(11, Fst::RELATION('+',1)),\  Fst::NODE() |

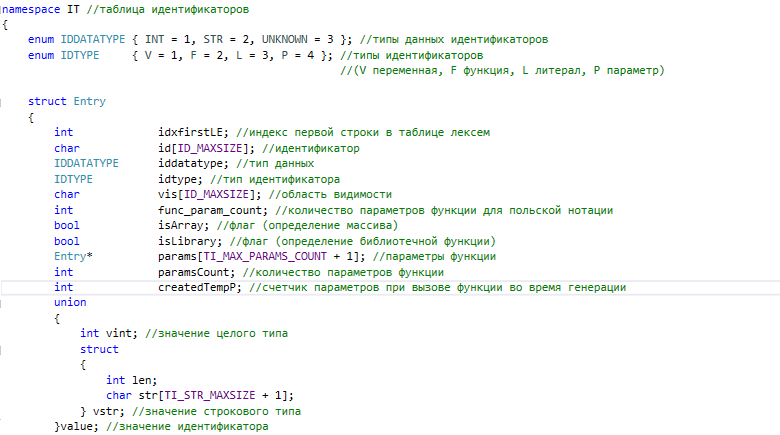
|  |  |
| --- | --- |
| - | #define GRAPH\_ARYTHM\_MINUS 2,\  Fst::NODE(11, Fst::RELATION('-',1)),\  Fst::NODE() |
| \* | #define GRAPH\_ARYTHM\_MULT 2,\  Fst::NODE(11, Fst::RELATION('\*',1)),\  Fst::NODE() |
| / | #define GRAPH\_ARYTHM\_DIV 2,\  Fst::NODE(11, Fst::RELATION('/',1)),\  Fst::NODE() |
| = | #define GRAPH\_EQUALS 2,\  Fst::NODE(1,Fst::RELATION('=',1)),\  Fst::NODE() |
| ; | #define GRAPH\_SEMICOLON 2,\  Fst::NODE(1,Fst::RELATION(';',1)),\  Fst::NODE() |
| , | #define GRAPH\_COMMA 2,\  Fst::NODE(1,Fst::RELATION(',',1)),\  Fst::NODE() |
| { | #define GRAPH\_LEFT\_BRACE 2,\  Fst::NODE(1,Fst::RELATION('{',1)),\  Fst::NODE() |
| } | #define GRAPH\_RIGHT\_BRACE 2,\  Fst::NODE(1,Fst::RELATION('}',1)),\  Fst::NODE() |
| ( | #define GRAPH\_OPEN\_PARENTHESIS 2,\  Fst::NODE(1,Fst::RELATION('(',1)),\  Fst::NODE() |
| ) | #define GRAPH\_CLOSE\_PARENTHESIS 2,\  Fst::NODE(1,Fst::RELATION(')',1)),\  Fst::NODE() |
| [ | #define GRAPH\_OPEN\_SQUARE 2,\  Fst::NODE(1,Fst::RELATION('[',1)),\  Fst::NODE() |
| ] | #define GRAPH\_CLOSE\_SQUARE 2,\  Fst::NODE(1,Fst::RELATION(']',1)),\  Fst::NODE() |

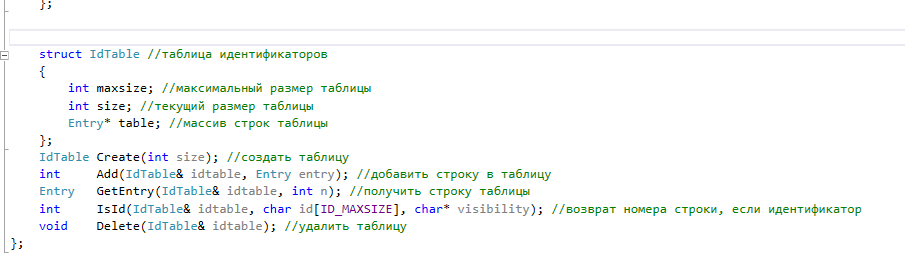
# Приложение Б

Структура таблицы лексем:



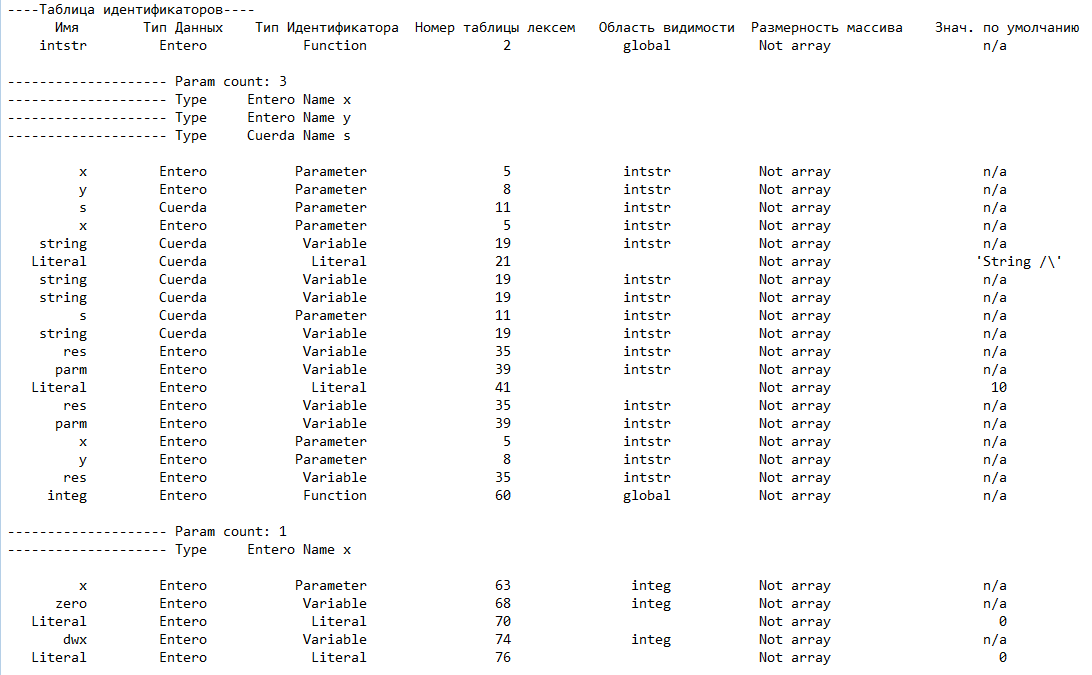
Структура таблицы идентификаторов:

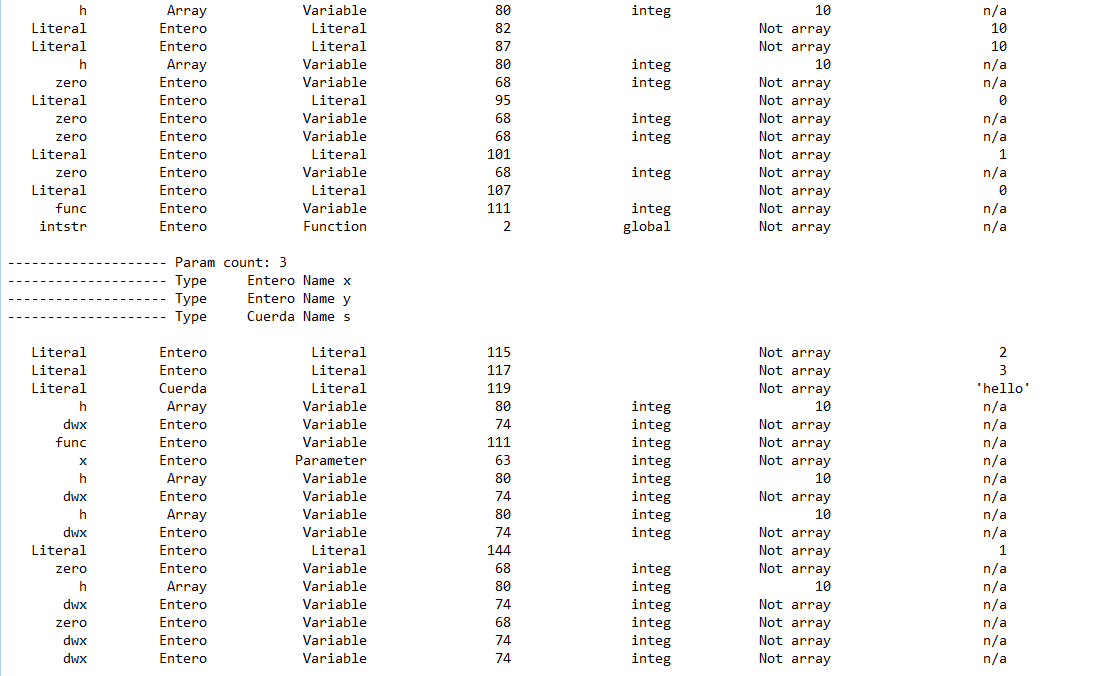


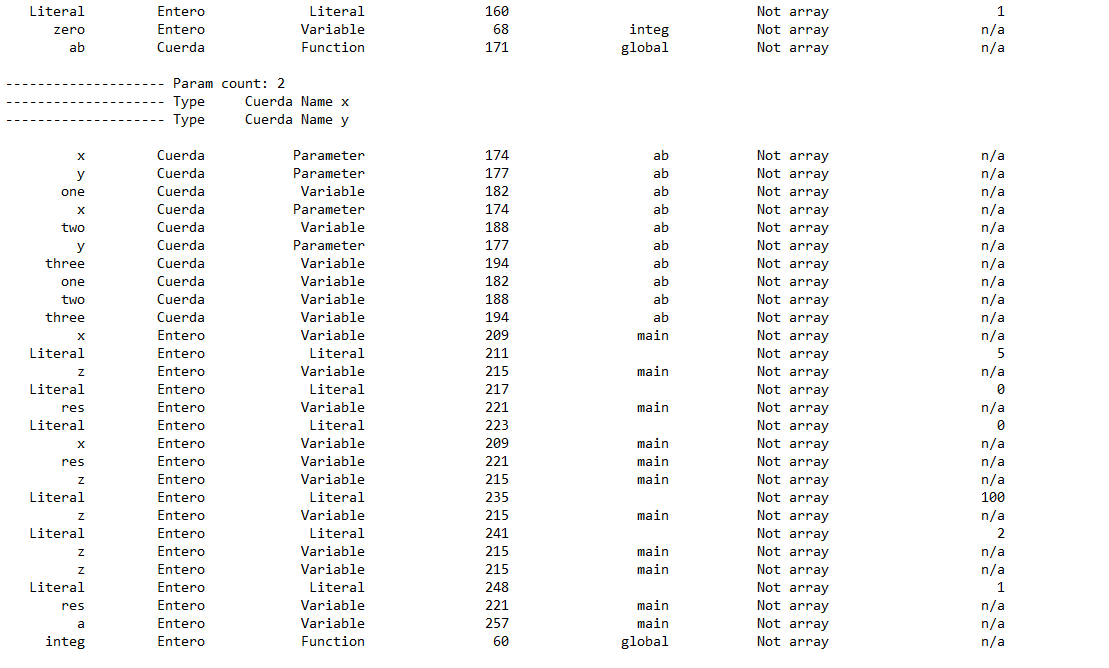


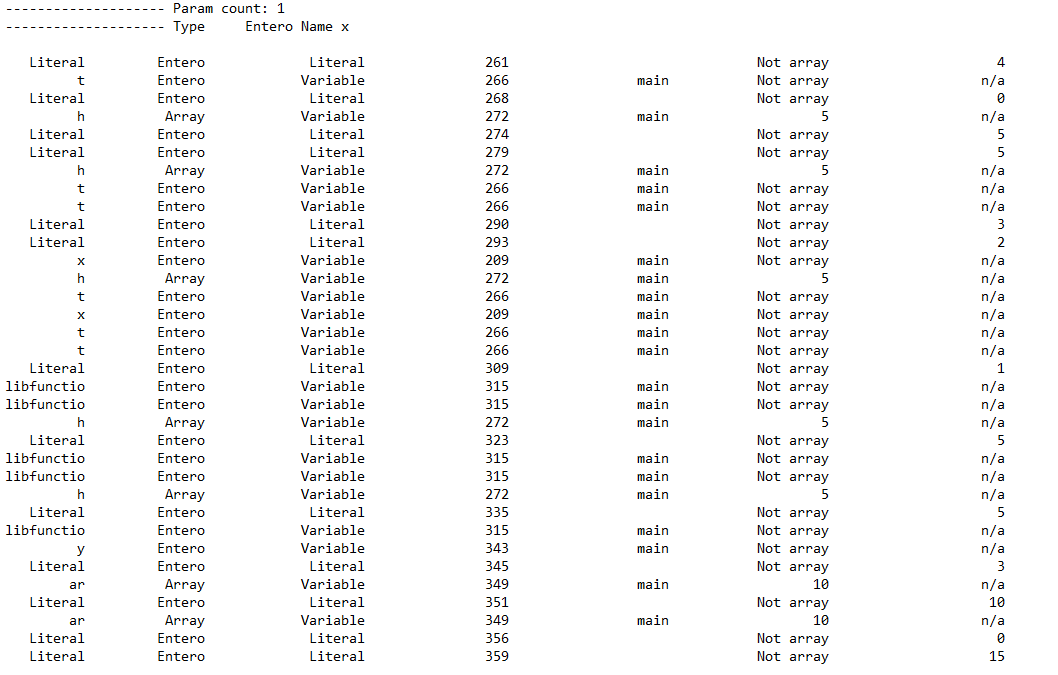
# Приложение В





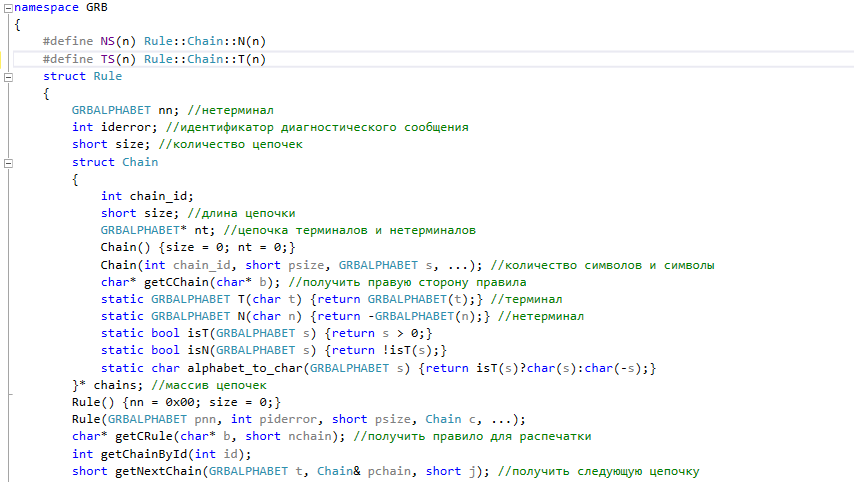


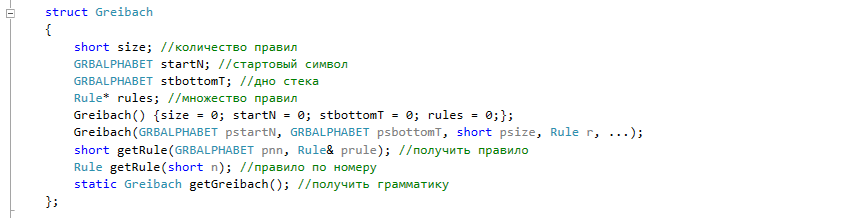




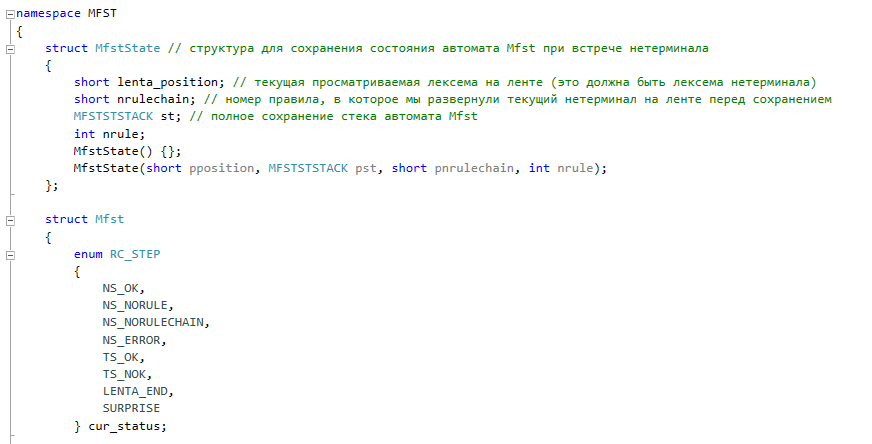
# Приложение Г

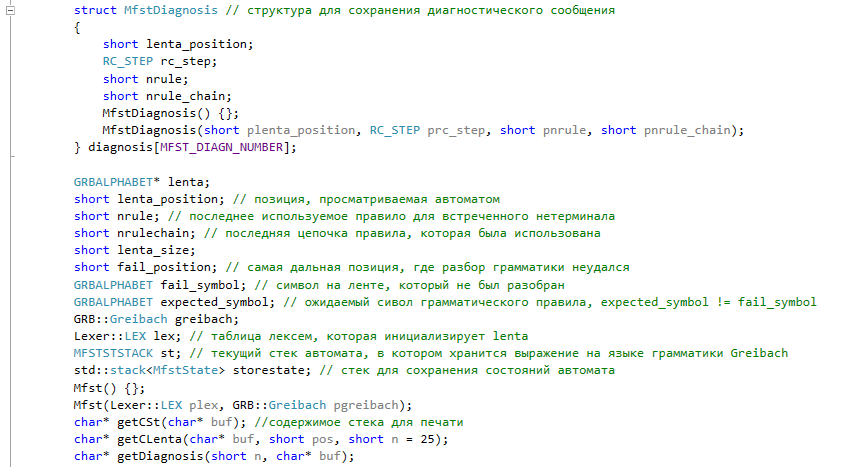
Структура грамматики Грейбах:

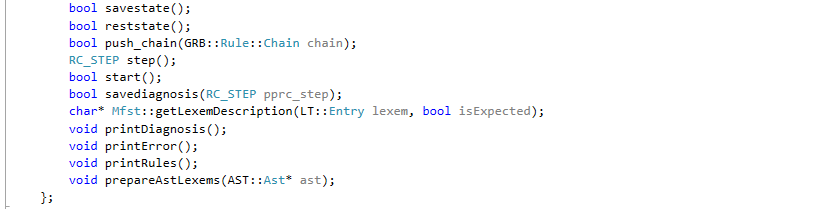




Структура магазинного автомата:

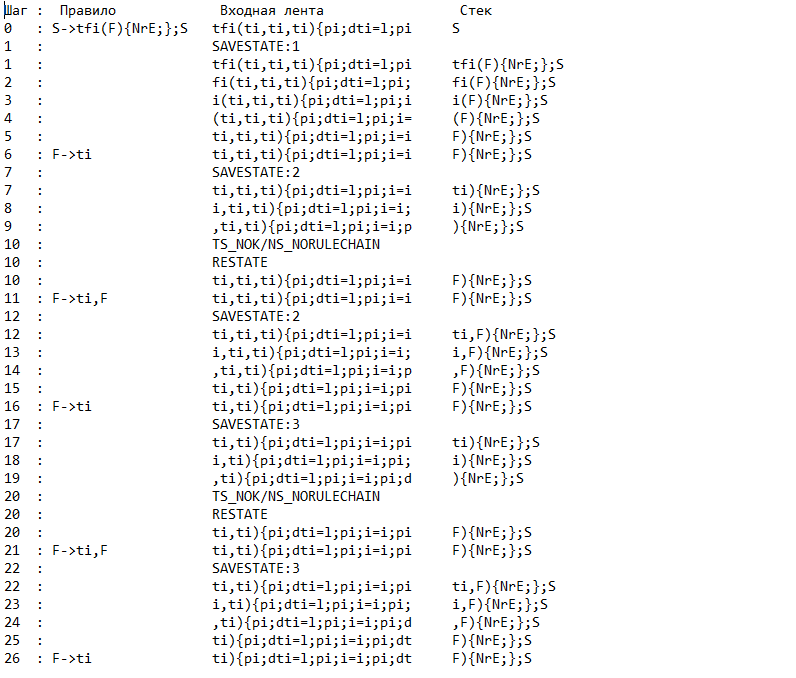
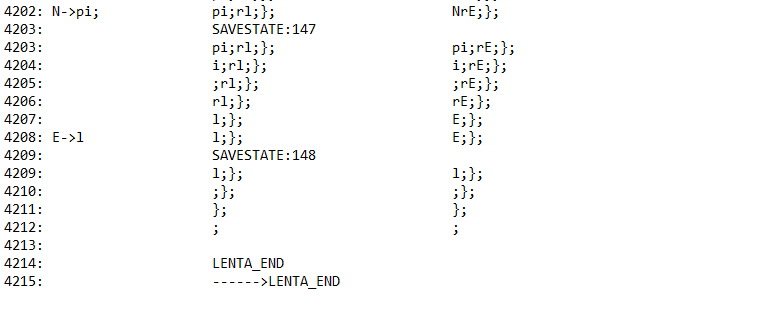




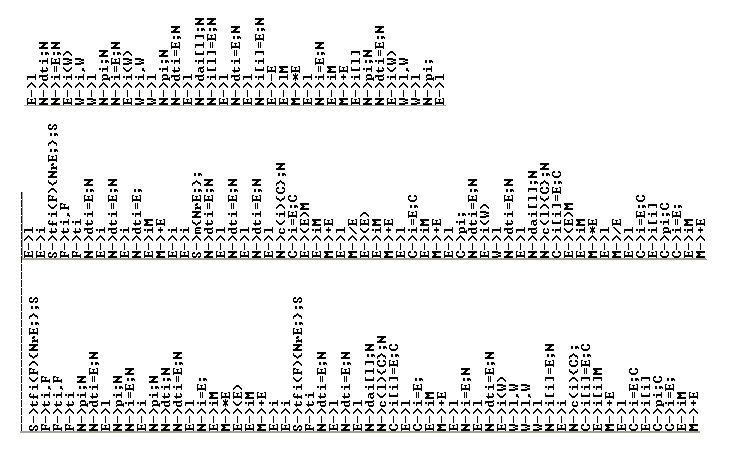


# Приложение Д

Протокол разбора кода из контрольного примера:

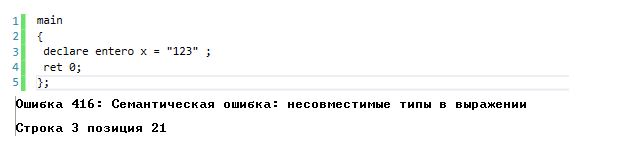
Дерево разбора:



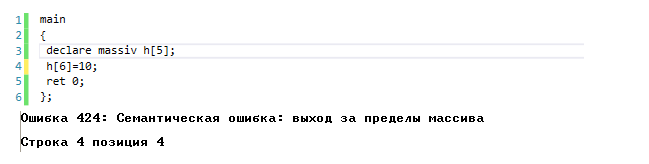
# Приложение Е

Пример работы семантического анализатора:

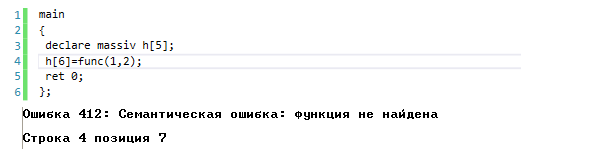
Пример ошибки при несовпадении типов: идентификатор x имеет целочисленный тип данных, литерал – строковый.



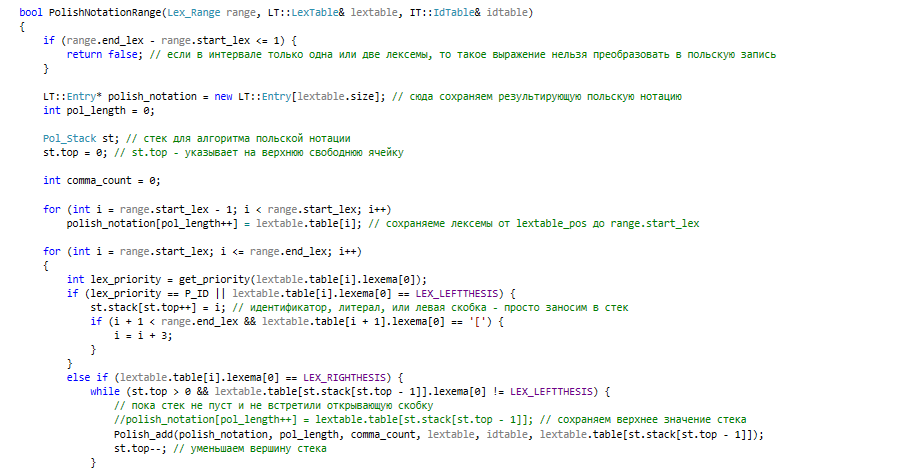
Пример ошибки при выходе за пределы массива: объявлен массив размерностью 5, обращаемся к элементу с индексом 6.

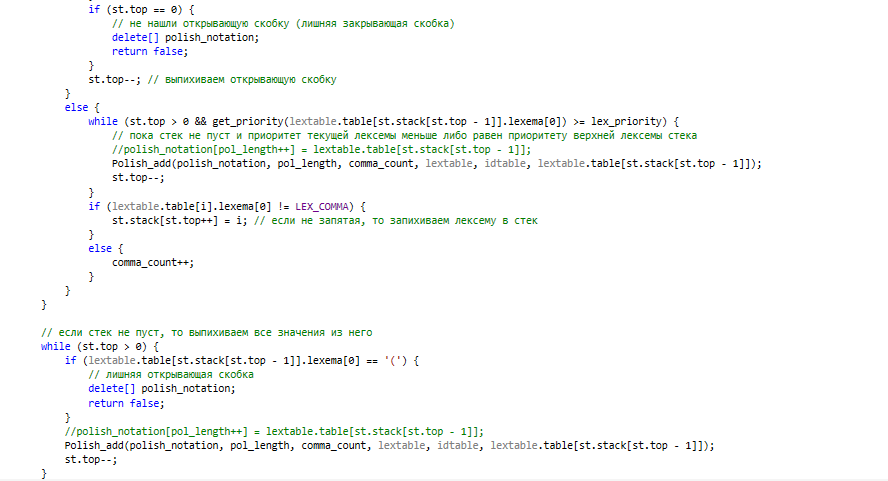


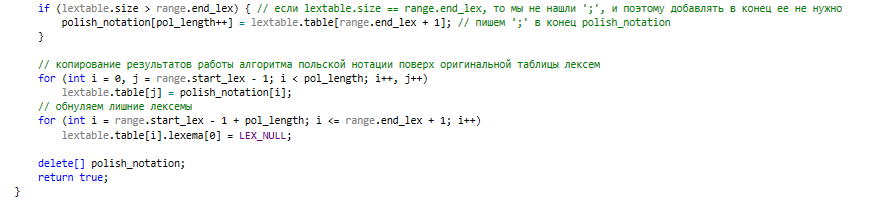
Пример ошибки при вызове неопределенной функции: пользовательская функция func нигде не была определена ранее.



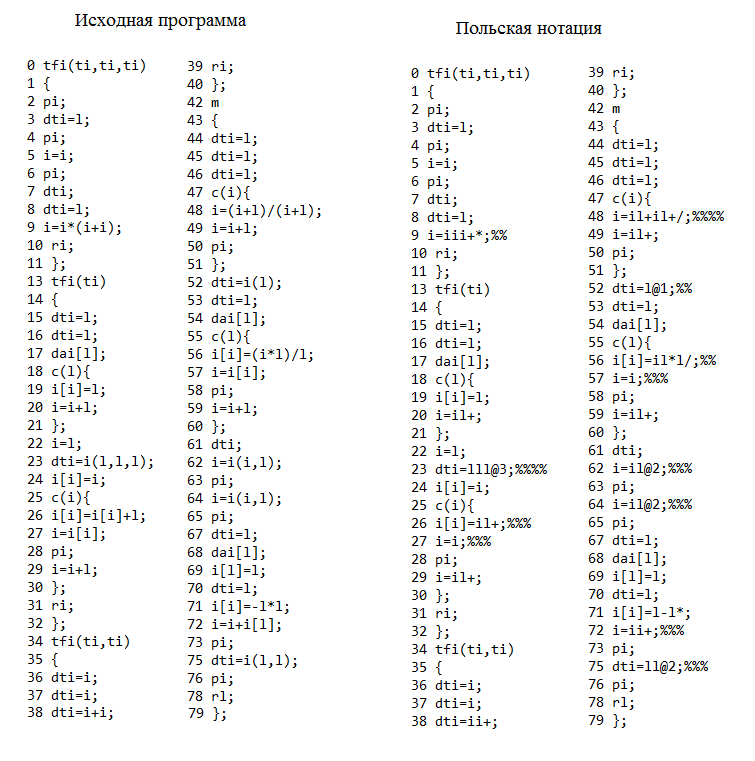
# Приложение Ж







# Приложение З



# Список литературы

1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
2. ASCII Table and Description [Электронный ресурс] – Режим доступа: <http://www.asciitable.com/> - Дата доступа: 02.12.2016
3. Let’s Build A Simple Interpreter [Электронный ресурс] – Режим доступа: <https://ruslanspivak.com/lsbasi-part1/>- Дата доступа: 14.12.2016