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 SAA-2018»

Выполнил студент Септилко Анастасия Антоновна
(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 Наталья Владимировна
(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 2018

Содержание

Введение

Главной целью данной курсовой работы является разработка транслятора для языка программирования SAA-2018. Основная задача транслятора заключается в том, чтобы сделать программу, написанную языке программирования SAA-2018, понятной компьютеру. В данном курсовом проекте трансляция будет осуществляться в код на языке Assembler.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;
– разбратка структуры транслятора;
– разработка лексического анализатора;
– разработка синтаксического анализатора;
– разработка семантического анализатора;
– обработка выражений;
– генерация кода на язык Assebler;
– тестирование транслятора.

Язык программирования SAA-2018 предназначен для выполнения простейших арифметический действий и операций над строками.

1 Спецификация языка программирования

1.1. Характеристика языка программирования

Язык SAA-2018 – это универсальный, строго типизированный, процедурный, компилируемый язык. Не является объектно-ориентированным.

1.2. Алфавит языка

Алфавит языка SAA-2018 основан на кодировке Windows-1251, представленной на рисунке 1.1.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F
80	Ђ 0402	Ѓ 0403	‚ 201A	ѓ 0453	„ 201E	… 2026	† 2020	‡ 2021	€ 20AC	‰ 2030	Љ 0409	‹ 2039	Њ 040A	Ќ 040C	Ћ 040B	Џ 040F
90	ђ 0452	‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014		™ 2122	љ 0459	› 203A	њ 045A	ќ 045C	ћ 045B	џ 045F
A0	NBSP 00A0	Ў 040E	ў 045E	Ј 0408	¤ 00A4	Ґ 0490	¦ 00A6	§ 00A7	Ё 0401	© 00A9	Є 0404	« 00AB	¬ 00AC	- 00AD	® 00AE	Ї 0407
B0	° 00B0	± 00B1	І 0406	і 0456	ґ 0491	µ 00B5	¶ 00B6	· 00B7	ё 0451	№ 2116	є 0454	» 00BB	ј 0458	Ѕ 0405	ѕ 0455	ї 0457
C0	А 0410	Б 0411	В 0412	Г 0413	Д 0414	Е 0415	Ж 0416	З 0417	И 0418	Й 0419	К 041A	Л 041B	М 041C	Н 041D	О 041E	П 041F
D0	Р 0420	С 0421	Т 0422	У 0423	Ф 0424	Х 0425	Ц 0426	Ч 0427	Ш 0428	Щ 0429	Ъ 042A	Ы 042B	Ь 042C	Э 042D	Ю 042E	Я 042F
E0	а 0430	б 0431	в 0432	г 0433	д 0434	е 0435	ж 0436	з 0437	и 0438	й 0439	к 043A	л 043B	м 043C	н 043D	о 043E	п 043F
F0	р 0440	с 0441	т 0442	у 0443	ф 0444	х 0445	ц 0446	ч 0447	ш 0448	щ 0449	ъ 044A	ы 044B	ь 044C	э 044D	ю 044E	я 044F

Рисунок 1.1 – Алфавит входных символов

Исходный код SAA-2018 может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, русские символы разрешены только в строковых литералах.

1.3. Символы сепараторы

Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Сепараторы

Сепаратор	Назначение
; ‘ ‘ (пробел) , = + - *	разделение инструкций
[ ]	программный блок
( )	параметры/приоритетность операций (в выражениях)

1.4. Применяемые кодировки

Для написания исходного кода на языке программирования SAA-2018 используется кодировка Windows-1251.

1.5. Типы данных

В языке SAA-2018 есть 2 типа данных: целочисленный и строковый. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка SAA-2018

Тип данных	Описание типа данных
real	Фундаментальный тип данных. Предусмотрен для объявления целочисленных положительных данных (4 байта). Автоматически инициализируется нулевым значением. Возможные операции: *арифметические* + – бинарный, суммирование; - – бинарный, вычитание; * – бинарный, умножение; = – присваивание значения;
word	Фундаментальный тип данных. Предусмотрен для объявления строк. (1 символ – 1 байт). Автоматическая инициализация строкой нулевой длины. Максимальное количество символов в строке – 255.

1.6. Преобразование типов данных

В языке программирования SAA-2018 преобразование типов данных не поддерживается.

1.7. Идентификаторы

В имени идентификатора допускаются символы латинского алфавита нижнего регистра. Максимальная длина имени - 6 символов.

1.8. Литералы

В языке существует 2 типа литералов: целого и символьного типов. Краткое описание литералов представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

Тип литерала	Описание
Литералы целого типа	Целочисленные неотрицательные литералы, инициализируются 0. Литералы только rvalue.
Строковые литералы	Символы, заключённые в “” (двойные кавычки), инициализируются пустой строкой, строковые переменные. Только rvalue.

1.9. Область видимости идентификаторов

Область видимости «сверху вниз» (по принципу C++). В языке SAA-2018 требуется обязательное объявление переменной перед её использованием. Все переменные должны находиться внутри программного блока языка. Имеется возможность объявления одинаковых переменных в разных блоках. Каждая переменная получает префикс – название функции, в которой она объявлена.

1.10. Инициализация данных

Таблица 1.4 – Способы инициализации переменных

Вид инициализации	Примечание
dim <тип данных> <идентификатор>;	Автоматическая инициализация: переменные типа real инициализируются нулём, переменные типа word – пустой строкой.
<идентификатор> = <значение>;	Присваивание переменной значения.

1.11. Инструкции языка

Все возможные инструкции языка программирования SAA-2018 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования SAA-2018

Инструкция	Запись на языке SAA-2018
Объявление переменной	dim <тип данных> <идентификатор>;
Присваивание	<идентификатор> = <значение> | <идентификатор>;
Объявление внешней функции	<тип данных> procedure <идентификатор> (<тип данных> <идентификатор>, …) […]
Точка входа	entry […]
Возврат значения из подпрограммы	endp <идентификатор> | <литерал>;
Вывод данных	read (<идентификатор> | <литерал>);

1.12. Операции языка

Язык программирования SAA-2018 может выполнять арифметические операции, представленные в таблице 1.6.

Таблица 1.6 – Приоритетности операций языка программирования SAA-2018

Операция	Приоритетность операции
( )	0 или 4
,	1
*	2
+ -	3

Максимальным значением приоритетности является “0”, минимальным “4” соответственно.

1.13. Выражения и их вычисления

Круглые скобки в выражении используются для изменения приоритета операций. Также не допускается запись двух подряд идущих арифметических операций. Выражение может содержать вызов функции.

1.14. Программные конструкции языка

Ключевые программные конструкции языка программирования SAA-2018 представлены чуть ниже в таблице 1.7.

Таблица 1.7 – Программные конструкции языка SAA-2018

Главная функция (точка входа в приложение)	entry […]
Функция	<тип> procedure <идентификатор>(<тип> <идентификатор>, …) [… endp <выражение>;]

1.15. Область видимости

В языке SAA-2018 переменные обязаны находиться внутри программного блока функций (по принципу C++). Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

1.16. Семантические проверки

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

Таблица 1.8 – Семантические проверки

Номер	Правило
1	Идентификаторы функций не должны повторяться
2	Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении
3	Тип данных передаваемых значений в функцию стандартной библиотеки должен соответствовать заявленному.
4	Идентификатор должен быть объявлен до его использования.
5	Операнды в арифметическом выражении не могут быть разных типов

1.17. Распределение оперативной памяти на этапе выполнения

Переменные целочисленного типа находятся в стеке, так же в стеке находятся указатели на строки. Распределение оперативной памяти происходит на этапе генерации. Промежуточный код, таблица лексем и таблица идентификаторов

сохраняются в структуры с выделенной под них динамической памятью, которая очищается по окончанию работы транслятора.

1.18. Стандартная библиотека и её состав

Функции стандартной библиотеки с описанием представлены в таблице 1.9. Стандартная библиотека написана на языке программирования C++.

Таблица 1.9 – Состав стандартной библиотеки

Имя функции	Возвращаемое значение	Принимаемые параметры	Описание
power	real	real x – число real n – степень	Функция возводит число x в степень n
strln	real	word x - строка	Функция вычисляет длину строки x
readr	0	real x - число	Функция выводит на консоль число x
readw	0	word x - строка	Функция выводит на консоль строку x

1.19. Ввод и вывод данных

Ввод данных не поддерживается языком программирования SAA-2018.

read (<идентификатор или литерал>); – вывод в стандартный поток вывода.

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

1.20. Точка входа

Точкой входа является функция entry.

1.21. Препроцессор

Препроцессор в языке программирования SAA-2018 не предусмотрен.

1.22. Соглашения о вызовах

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

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

1.23. Объектный код

Язык программирования SAA-2018 транслируется в язык ассемблера.

1.24. Классификация сообщений транслятора

В случае возникновения ошибки в коде программы на языке SAA-2018 и выявления её транслятором в текущий файл протокола выводится сообщение. Их классификация сообщений приведена в таблице 1.10.

Таблица 1.10. – Классификация сообщений транслятора

Интервал	Описание ошибок
0-99	Системные ошибки
100-109	Ошибки параметров
110-119	Ошибки открытия и чтения файлов
120-129	Ошибки лексического анализа
600-699	Ошибки синтаксического анализа
700-799	Ошибки семантического анализа

1.25. Контрольный пример

Контрольный пример представлен в главе «Приложения».

2 Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор преобразует программу, написанную на языке SAA-2018 в программу на языке ассемблера. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.

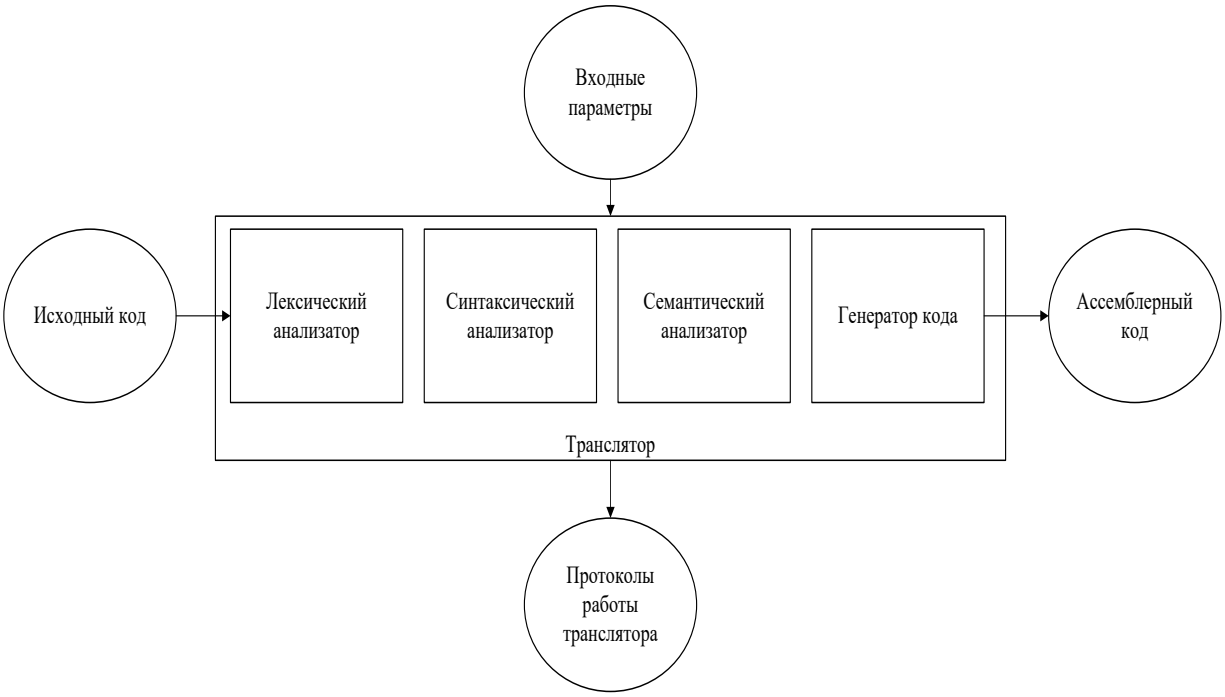

Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка и формирование таблицы лексем и таблицы идентификаторов. Подробнее описан в 3 главе.

Семантический анализ в свою очередь является проверкой исходной программы на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики. Подробное описание представлено в 5 главе.

В моём трансляторе этапы лексического и семантического анализа являются единым целым и выполняются одновременно.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода. Входным параметром для синтаксического анализа является таблица лексем. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора. Подробнее рассмотрен в главе 4.

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах

трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке SAA-2018, прошедший все предыдущие этапы, в код на языке Ассемблера. Более полно описан в главе 7.

2.2 Перечень входных параметров транслятора

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка SAA-2018

Входной параметр	Описание	Значение по умолчанию
-in:<имя_файла>	Входной файл с расширением .txt, в котором содержится исходный код на SAA-2018	Не предусмотрено
-log:<имя_файла>	Файл для записи результата проверки входного файла на допустимость символов	<имя_файла>.log
-out:<имя_файла>	Файл для записи результата работы транслятора	<имя_файла>.asm
-lex:<имя_файла>	Файл для записи результата работы лексического и семантического анализа.	<имя_файла>.lex
-sin:<имя_файла>	Файл для записи результата работы синтаксического анализа	<имя_файла>.sin

2.3 Перечень протоколов, формируемых транслятором и их содержимое

Таблица с перечнем протоколов, формируемых транслятором языка SAA-2018 и их назначением представлена в таблице 2.2

Таблица 2.2 – Протоколы, формируемые транслятором языка SAA-2018

Формируемый протокол	Описание протокола
Файл журнала с параметром <log>	Содержит информацию о входных параметрах в приложение и о этапе проверки символов на допустимость.
Выходной файл с параметром <out>	Содержит сгенерированный код на языке Ассемблера.
lex:<имя_файла>	Результат работы лексического и семантического анализа. Содержит таблицы лексем и идентификаторов.
sin:<имя_файла>	Результат работы синтаксического анализа. Содержит правила разбора, трассировку, а также преобразованные после польской записи таблицы лексем и идентификаторов.

3 Разработка лексического анализатора

3.1 Структура лексического анализатора

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке SAA-2018. На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1

Рисунок 3.1 – Структура лексического анализатора SAA-2018

3.2 Контроль входных символов

Таблица для контроля входных символов представлена на рисунке 3.2

```
//  ICP::F -- запрещенный символ,
//  ICP::T  --  разрешенный символ,
//  ICP::S -- сепараторы    (   )   *   +   ,   -   =   ;   [   ]

#define ICP_CODE_TABLE {\
        /* 0 */ /* 1 */ /* 2 */ /* 3 */ /* 4 */ /* 5 */ /* 6 */ /* 7 */ /* 8 */ /* 9 */ /* A */ /* B */ /* C */ /* D */ /* E */ /* F */ \
/*0*/    ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::T, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, \
/*1*/    ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, \
/*2*/    ICP::T, ICP::F, ICP::T, ICP::T, ICP::F, ICP::F, ICP::F, ICP::T, ICP::S, ICP::S, ICP::S, ICP::S, ICP::S, ICP::S, ICP::T, ICP::F, \
/*3*/    ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::S, ICP::S, ICP::S, ICP::S, ICP::T, ICP::F, \
/*4*/    ICP::F, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, \
/*5*/    ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::S, ICP::F, ICP::S, ICP::F, ICP::F, \
/*6*/    ICP::F, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, \
/*7*/    ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, \
    \
/*8*/    ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, \
/*9*/    ICP::F, ICP::F, ICP::F, ICP::T, ICP::T, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, \
/*A*/    ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, \
/*B*/    ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, ICP::F, \
/*C*/    ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, \
/*D*/    ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, \
/*E*/    ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, \
/*F*/    ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T, ICP::T \
}
```

Рисунок 3.2. – Таблица контроля входных символов

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

Описание значения символов: T – разрешённый символ, F – запрещённый символ, S – сепаратор.

3.3 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на токены.

Описание алгоритма удаления избыточных символов:

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

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

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

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

Тип цепочки	Цепочка	Лексема
Ключевые слова	dim	d
	real	r
	word	w
	procedure	p
	read	s
	power	q
	strl	n
	endp	e
	entry	m
Иное	Идентификатор	i
	Целочисленный литерал	r
	Строковый литерал	w

Продолжение таблицы 3.1

Сепараторы	;	;
	,	,
	[	[
	]	]
	(	(
	)	)
	=	=
Операторы	-	a
	*	a
	+	a

Пример реализации таблицы лексем представлен в приложении А.

Также в приложении А находятся конечные автоматы, соответствующие лексемам языка SAA-2018.

3.5 Основные структуры данных

Основные структуры таблиц лексем и идентификаторов данных языка SAA-2018, используемых для хранения, представлены в приложении А. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде и приоритет. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, смысловой тип идентификатора и его значение.

3.6 Принцип обработки ошибок

При возникновении критической ошибки – работа транслятора прекращается.

3.7 Структура и перечень сообщений лексического анализатора

Перечень сообщений лексического анализатора представлен на рисунке 3.3.

```
ERROR_ENTRY(120, "|LA|: Цепочка символов не разобрана"),
ERROR_ENTRY(121, "|LA|: Таблица лексем переполнена"),
ERROR_ENTRY(122, "|LA|: Таблица идентификаторов переполнена"),
ERROR_ENTRY(123, "|LA|: Дублирование идентификатора"),
ERROR_ENTRY(124, "|LA|: Дублирование арифметических операций"),
ERROR_ENTRY_NODEF(125),
ERROR_ENTRY_NODEF(126),ERROR_ENTRY_NODEF(127),ERROR_ENTRY_NODEF(128), ERROR_ENTRY_NODEF(129),
```

Рисунок 3.3 – Перечень ошибок лексического анализатора

3.8 Параметры лексического анализатора и режимы его работы

Входным параметром лексического анализа является очередь, состоящая из структур, полями которых являются лексема и номер её строки в исходном файле, полученные на этапе проверки исходного кода на допустимость символов.

3.9 Алгоритм лексического анализа

Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором. Цель лексического анализа — выделение и классификация лексем в тексте исходной программы. Лексический анализатор распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов.

Регулярные выражения — аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа.

Пример. Регулярное выражение для ключевого слова real: ‘real’.

Граф конечного автомата для этой лексемы представлен на рисунке 3.4. S0 – начальное состояние, S4 – конечное состояние автомата.

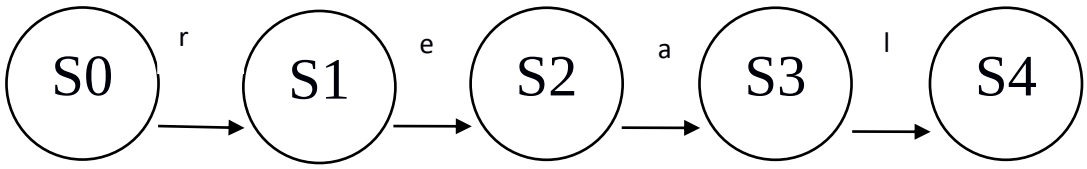

Рисунок 3.4 – Граф переходов для цепочки ‘real’

3.10 Контрольный пример

Результат работы лексического анализатора – таблицы лексем и идентификаторов – представлен в приложении А.

4 Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

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

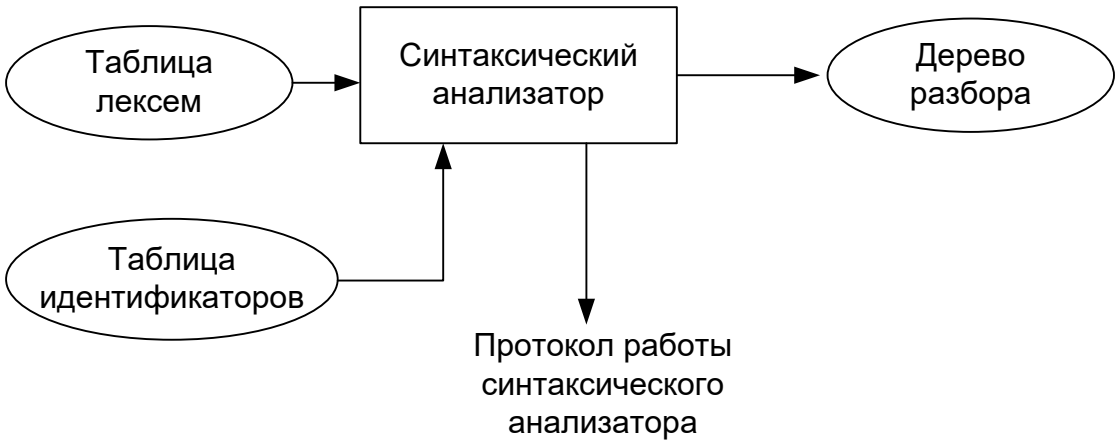

Рисунок 4.1 – Структура синтаксического анализатора

4.2 Контекстно свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка SAA-2018 используется контекстно-свободная грамматика $G = \langle T, N, P, S \rangle$, где

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

1) $A \to a\alpha$, где $a \in T, \alpha \in (T \cup N) \cup \{\lambda\}$; (или $\alpha \in (T \cup N)^*$, или $\alpha \in V^*$)

2) $S \to \lambda$, где $S \in N$ — начальный символ, при этом если такое правило существует, то нетерминал S не встречается в правой части правил.

Грамматика языка SAA-2018 представлена в приложении Б.

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

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

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов SAA-2018 (назначни в таблицу)

Нетерминал	Цепочки правил
S	tpi(F)[N]S m[N]

N	dti;N
	eE;
	i=E;N
	s(i);N
	s(w);N
	s(r);N
	s(w);
	s(i);
	s(r);

E	i w r (E) rM i(W) iM dM (E)M i(W)M q(i,r) q(r,r) q(i,i) q(r,i) n(i) n(w)
F	ti ti,F
W	i w r i,W w,W r,W
M	a aE aEM

4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку $M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle$, описание которой представлено в таблице 4.2. Структура данного автомата показана в приложении В.

Таблица 4.2 – Описание компонентов магазинного автомата

Компонента	Определение	Описание
Q	Множество состояний автомата	Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата
V	Алфавит входных символов	Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1.
Z	Алфавит специальных магазинных символов	Алфавит магазинных символов содержит стартовый символ и маркер дна стека
δ	Функция переходов автомата	Функция представляет из себя множество правил грамматики, описанных в таблице 4.1.
q_0	Начальное состояние автомата	Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ S)
z_0	Начальное состояние магазина автомата	Символ маркера дна стека ($)
F	Множество конечных состояний	Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты.

4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающей правила языка SAA-2018. Данные структуры представлены в приложении В.

4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата следующий:

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

4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен на рисунке 4.1.

```
ERROR_ENTRY(600, "|SA|: Неверная структура программы"),
ERROR_ENTRY(601, "|SA|: Ошибочный оператор"),
ERROR_ENTRY(602, "|SA|: Неверное выражение"),
ERROR_ENTRY(603, "|SA|: Ошибка в параметрах функции или операторе объявления"),
ERROR_ENTRY(604, "|SA|: Ошибка в параметре вызываемой функции стандартной библиотеки"),
ERROR_ENTRY(605, "|SA|: Неверное завершение блока кода"),
ERROR_ENTRY_NODEF(606),
ERROR_ENTRY(607, "|SA|: Ошибка в параметрах вызываемой функции"),
ERROR_ENTRY(608, "|SA|: Ошибка арифметического оператора"),

ERROR_ENTRY_NODEF(609), ERROR_ENTRY_NODEF(610),

ERROR_ENTRY(611, "|SA|: Не найден конец правила"),
ERROR_ENTRY(612, "|SA|: Цепочка разобрана не полностью (стек не пустой)"),
ERROR_ENTRY(613, "|SA|: Точка входа в программу не задана"),
ERROR_ENTRY_NODEF(614), ERROR_ENTRY_NODEF(615), ERROR_ENTRY_NODEF(616),
ERROR_ENTRY_NODEF(617), ERROR_ENTRY_NODEF(618), ERROR_ENTRY_NODEF(619),
ERROR_ENTRY_NODEF10(620), ERROR_ENTRY_NODEF10(630), ERROR_ENTRY_NODEF10(640),
ERROR_ENTRY_NODEF10(650), ERROR_ENTRY_NODEF10(660), ERROR_ENTRY_NODEF10(670), ERROR_ENTRY_NODEF10(680),
ERROR_ENTRY_NODEF10(690),
```

Рисунок 4.1 – Перечень сообщений синтаксического анализатора

4.7 Параметры синтаксического анализатора и режимы его работы

Входным параметром синтаксического анализатора является таблица лексем, полученная на этапе лексического анализа, а также правила контекстно-свободной грамматики в форме Грейбах.

Выходными параметрами являются трассировка прохода таблицы лексем (при наличии разрешающего ключа) и правила разбора, которые записываются в файл протокола данного этапа обработки.

4.8 Принцип обработки ошибок

Обработка ошибок происходит следующим образом:

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

4.9 Контрольный пример

Пример разбора синтаксическим анализатором исходного кода на языке SAA-2018 представлен в приложении Г. Дерево разбора исходного кода также представлено в приложении Г.

5 Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализ происходит при выполнении фазы лексического анализа и реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях: установки флага или нахождении в особом месте программы (оператор выхода из функции, оператор ветвления, вызов функции стандартной библиотеки).

5.2 Функции семантического анализатора

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

5.3 Структура и перечень сообщений семантического анализатора

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.1.

```
ERROR_ENTRY(700, "|SMA|: Повторное объявление идентификатора"),
ERROR_ENTRY(701, "|SMA|: Ошибка в типе идентификатора"),
ERROR_ENTRY(702, "|SMA|: Ошибка в передаваемых значениях в функцию"),
ERROR_ENTRY(703, "|SMA|: В функцию не переданы параметры"),
ERROR_ENTRY(704, "|SMA|: Тип данных результата выражения не соответствует присваиваемому идентификатору"),
ERROR_ENTRY(705, "|SMA|: Ошибка в параметре вызываемой функции pow стандартной библиотеки"),
ERROR_ENTRY(706, "|SMA|: Необъявленный идентификатор"),
ERROR_ENTRY(707, "|SMA|: Несоответствие типов в операторе присваивания"),
ERROR_ENTRY(708, "|SMA|: Неверная структура программы"),
ERROR_ENTRY(709, "|SMA|: Превышен максимальный размер идентификатора"),
ERROR_ENTRY_NODEF10(710),ERROR_ENTRY_NODEF10(720),ERROR_ENTRY_NODEF10(730),ERROR_ENTRY_NODEF10(740),
ERROR_ENTRY_NODEF10(750),ERROR_ENTRY_NODEF10(760),ERROR_ENTRY_NODEF10(770),ERROR_ENTRY_NODEF10(780),
ERROR_ENTRY_NODEF10(790),
ERROR_ENTRY_NODEF100(800), ERROR_ENTRY_NODEF100(900)
```

Рисунок 5.1 – Перечень сообщений семантического анализатора

5.4 Принцип обработки ошибок

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

5.5 Контрольный пример

Результат работы контрольного примера расположен в приложении А, где показан результат лексического анализатора, т.к. представленные таблицы лексем и идентификаторов проходят лексическую и семантическую проверки одновременно.

6 Преобразование выражений

6.1 Выражения, допускаемые языком

В языке SAA-2018 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, * и (), и вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке SAA-2018

Приоритет	Операция
0	(
0	)
1	,
2	+
2	-
3	*
4	(– скобка параметров функции
4	) – скобка параметров функции

6.2 Польская запись

Выражения в языке SAA-2018 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок.

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

Алгоритм построения:

– исходная строка: выражение;

– результирующая строка: польская запись;

– стек: пустой;

– результирующая строка: польская запись;

– исходная строка просматривается слева направо;

– операнды переносятся в результирующую строку в порядке их следования;

– операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;

– операция выталкивает все операции с большим или равным приоритетом в результирующую строку;

– запятая не помещается в стек, если в стеке операции, то все выбираются в строку;

– отрывающая скобка помещается в стек;

– закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;

– закрывающая скобка с приоритетом, равным 4, выталкивает все до открывающей с таким же приоритетом и генерирует @ – специальный символ, в которого записывается информация о вызываемой функции, а в поле приоритета для данной лексемы записывается число параметров вызываемой функции;

– по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

Исходная строка	Результирующая строка	Стек
b*2 - n(i)		
*2 - n(i)	b	
2 - n(i)	b	*
- n(i)	b2	*
n(i)	b2*	-
(i)	b2*	-
i)	b2*	-
)	b2*i	-
	b2*i@$_1$-	

6.3 Программная реализация обработки выражений

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

6.4 Контрольный пример

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

В приложении Д приведены изменённые таблицы лексем и идентификаторов, отображающие результаты преобразования выражений в польский формат.

7 Генерация кода

7.1 Структура генератора кода

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

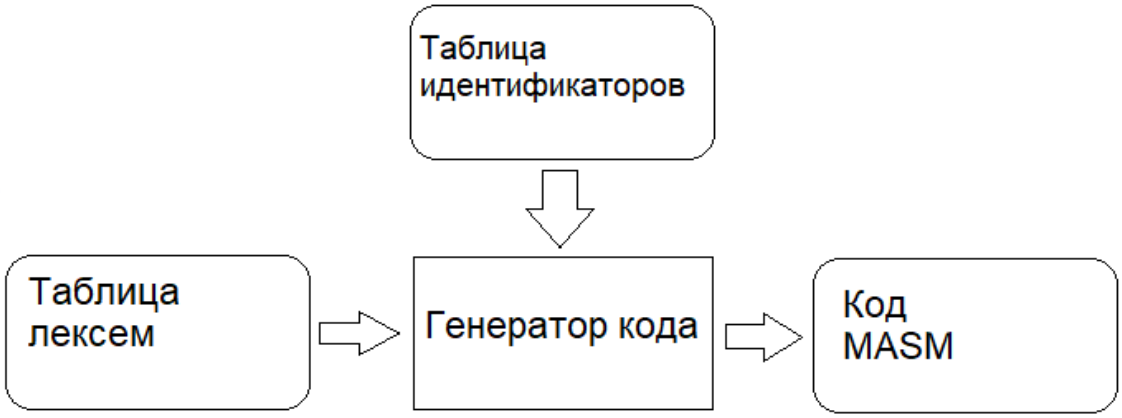

Рисунок 7.1 Структура генератора кода

7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка SAA-2018 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке SAA-2018 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка SAA-2018 и языка Ассемблера

Тип идентификатора на языке SAA-2018	Тип идентификатора на языке ассемблера	Пояснение
real	SDWORD	Хранит целочисленный тип данных со знаком.
word	DWORD	Хранит указатель на начало строки.
L(0-9)	BYTE DWORD	Литералы: символьные, целочисленные

7.3 Алгоритм работы генератора кода

Преобразования происходят по принципу, встретив определённую лексему и зная, в каком месте программы находится сейчас лексема, программа генерирует код на языке Ассемблера.

На рисунке 7.2 представлен пример описания лексемы на языке Ассемблера.

```
case LEX_READ:
{
    if (lex.idtable.table[lex.lextable.table[i + 2].indexTI].idDataType == IT::IDDATATYPE_REAL)
    {
        *(log.stream) << "\tpush " << lex.idtable.table[lex.lextable.table[i + 2].indexTI].id << "\n\t\tcall readr\n";
    }
    else
    {
        *(log.stream) << "\tpush offset " << lex.idtable.table[lex.lextable.table[i + 2].indexTI].id << "\n\t\tcall readw\n";
    }
    break;
}
```

Рисунок 7.2 – Алгоритм для лексемы ‘s’

Генерируемый код записывается в файл, который был параметром -out. Сгенерированный код можно посмотреть в приложении Е.

8 Тестирование транслятора

8.1 Тестирование фазы проверки на допустимость символов

В языке SAA-2018 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

Исходный код	Диагностическое сообщение
real procedure one [real a, real b]	Ошибка 111: |IN|: Недопустимый символ в исходном файле (-in) строка 1, позиция 16

8.2 Тестирование лексического анализатора

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

Таблица 8.2 – Тестирование лексического анализатора

Исходный код	Диагностическое сообщение
real 3procedure one (real a, real)	Ошибка 120: |LA|: Цепочка символов не разобрана строка 1

8.3 Тестирование синтаксического анализатора

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

Таблица 8.3 – Тестирование синтаксического анализатора

Исходный код	Диагностическое сообщение
real procedure one (a, real b)	Ошибка 603: |SA|: Ошибка в параметрах функции или операторе объявления строка 1

8.4 Тестирование семантического анализатора

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

Исходный код	Диагностическое сообщение
dim real x; dim real x;	Ошибка 700: |SMA|: Повторное объявление идентификатора строка 4

Заключение

В ходе выполнения курсовой работы был разработан транслятор для языка программирования SAA-2018. Таким образом, были выполнены основные задачи данной курсовой работы:

- Сформулирована спецификация языка SAA-2018;
- Разработаны конечные автоматы и алгоритмы для реализация лексического анализатора;
- Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
- Разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
- Разработан транслятор с языка программирования SAA-2018 на язык низкого уровня Assembler;
- Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка SAA-2018 включает:

1. 2 типа данных;
2. Поддержка операции вывода;
3. Возможность вызова функций стандартной библиотеки;
4. Наличие 3 арифметических операторов для вычисления выражений;
5. Структурированная система для обработки ошибок пользователя.

Список используемых источников

1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – М.: Вильямс, 2003. – 768с.
2. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.
3. Страуструп, Б. Принципы и практика использования C++ / Б. Стра-уструп – 2009 – 1238 с.
4. __stdcall [Электронный ресурс] – Режим доступа: https://msdn.microsoft.com/en-us/library/zxk0tw93.aspx - Дата доступа: 14.12.2016.

Приложения

Контрольный пример

```
real procedure one (real a, real b)
[
dim real x;
x = (a + b) * 4;
endp x;
]

real procedure two (word str)
[
dim real x;
x = strln(str);
endp x;
]

entry
[
dim real a;
dim real b;
dim real c;
dim real d;
dim real e;
dim real f;
a = 2;
b = 3;
c = one (a,b) - 1;
read (c);
dim word g;
g = "Hello World";
d = two (g);
f = power (a, 3);
read (d);
read (f);
read ("Завершение программы");
]
```

Приложение А

```
1: tpi(ti,ti)
2: [
3: dti;
4: i=(iai)ar;
5: ei;
6: ]
8: tpi(ti)
9: [
10: dti;
11: i=n(i);
12: ei;
13: ]
15: m
16: [
17: dti;
18: dti;
19: dti;
20: dti;
21: dti;
22: dti;
23: i=r;
24: i=r;
25: i=i(i,i)ar;
26: s(i);
27: dti;
28: i=w;
29: i=i(i);
30: i=q(i,r);
31: s(i);
32: s(i);
33: s(w);
34: ]
Количество лексем - 142
```

№	Идентификатор	Тип данных	Тип идентификатора	Индекс в ТЛ	Значение
0	one	real	функция	2	-
1	onea	real	параметр	5	-
2	oneb	real	параметр	8	-
3	onex	real	переменная	13	0
4	+	unknown	оператор	19	-
5	*	unknown	оператор	22	-
6	L0	real	литерал	23	4
7	two	real	функция	31	-
8	twostr	word	параметр	34	-
9	twox	real	переменная	39	0
10	strln	real	функция	43	-
11	entrya	real	переменная	56	0
12	entryb	real	переменная	60	0
13	entryc	real	переменная	64	0
14	entryd	real	переменная	68	0
15	entrye	real	переменная	72	0
16	entryf	real	переменная	76	0
17	L1	real	литерал	80	2
18	L2	real	литерал	84	3
19	-	unknown	оператор	94	-
20	L3	real	литерал	95	1
21	entryg	word	переменная	104	[0]""
22	L4	word	литерал	108	[13]""Hello World""
23	power	real	функция	119	-
24	L5	word	литерал	138	[22]""Завершение программы""

Количество идентификаторов: 22

```
FST fstAriph(LEX_ACTION, SYMB_ACTION, "", 2,
    NODE(3, RELATION('+', 1), RELATION('-', 1), RELATION('*', 1)), \
    NODE()
);
FST fstLeftHesis(LEX_LEFTHESIS, SYMB_NEED_PRIORITY, "", 2,
    NODE(1, RELATION('(', 1)), \
    NODE()
);
FST fstRightHesis(LEX_RIGHTHESIS, SYMB_NEED_PRIORITY, "", 2,
    NODE(1, RELATION(')', 1)), \
    NODE()
);
FST fstEqual(LEX_EQUAL, SYMB_LEXEMS, "", 2,
    NODE(1, RELATION('=', 1)), \
    NODE()
);
FST fstLeftBrace(LEX_LEFTBRACE, SYMB_LEXEMS, "", 2,
    NODE(1, RELATION('[', 1)), \
    NODE()
);
FST fstRightBrace(LEX_RIGHTBRACE, SYMB_LEXEMS, "", 2,
    NODE(1, RELATION(']', 1)), \
    NODE()
);
FST fstSemicolon(LEX_SEMICOLON, SYMB_LEXEMS, "", 2,
    NODE(1, RELATION(';', 1)), \
    NODE()
);

FST fstComma(LEX_COMMA, SYMB_NEED_PRIORITY, "", 2,
    NODE(1, RELATION(',', 1)), \
    NODE()
);
```

```
FST fstRealLit(LEX_REAL, SYMB_LITERAL, "", 2,
    NODE(10,
        RELATION('0', 1), RELATION('1', 1), RELATION('2', 1),
        RELATION('3', 1), RELATION('4', 1), RELATION('5', 1),
        RELATION('6', 1), RELATION('7', 1), RELATION('8', 1),
        RELATION('9', 1)),
    NODE()
);
FST fstId(LEX_ID, SYMB_TABLE_ID, "", 2,
    NODE(26, RELATION('a', 1), RELATION('b', 1), RELATION('c', 1), RELATION('d', 1),
        RELATION('e', 1), RELATION('f', 1), RELATION('g', 1), RELATION('h', 1),
        RELATION('i', 1), RELATION('j', 1), RELATION('k', 1), RELATION('l', 1),
        RELATION('m', 1), RELATION('n', 1), RELATION('o', 1), RELATION('p', 1),
        RELATION('q', 1), RELATION('r', 1), RELATION('s', 1), RELATION('t', 1),
        RELATION('u', 1), RELATION('v', 1), RELATION('w', 1), RELATION('x', 1),
        RELATION('y', 1), RELATION('z', 1)),
    NODE()
);
```

```
FST fstReal(LEX_REAL, SYMB_LEXEMS, "", 5,
    NODE(1, RELATION('r', 1)),
    NODE(1, RELATION('e', 2)),
    NODE(1, RELATION('a', 3)),
    NODE(1, RELATION('l', 4)),
    NODE()
);
FST fstWord(LEX_WORD, SYMB_LEXEMS, "", 5,
    NODE(1, RELATION('w', 1)),
    NODE(1, RELATION('o', 2)),
    NODE(1, RELATION('r', 3)),
    NODE(1, RELATION('d', 4)),
    NODE()
);
FST fstProc(LEX_PROC, SYMB_LEXEMS, "", 10,
    NODE(1, RELATION('p', 1)),
    NODE(1, RELATION('r', 2)),
    NODE(1, RELATION('o', 3)),
    NODE(1, RELATION('c', 4)),
    NODE(1, RELATION('e', 5)),
    NODE(1, RELATION('d', 6)),
    NODE(1, RELATION('u', 7)),
    NODE(1, RELATION('r', 8)),
    NODE(1, RELATION('e', 9)),
    NODE()
);
FST fstDim(LEX_DIM, SYMB_LEXEMS, "", 4,
    NODE(1, RELATION('d', 1)),
    NODE(1, RELATION('i', 2)),
    NODE(1, RELATION('m', 3)),
    NODE()
);


FST fstEntry(LEX_ENTRY, SYMB_LEXEMS, "", 6,
    NODE(1, RELATION('e', 1)),
    NODE(1, RELATION('n', 2)),
    NODE(1, RELATION('t', 3)),
    NODE(1, RELATION('r', 4)),
    NODE(1, RELATION('y', 5)),
    NODE()
);

FST fstRead(LEX_READ, SYMB_LEXEMS, "", 5,
    NODE(1, RELATION('r', 1)), \
    NODE(1, RELATION('e', 2)), \
    NODE(1, RELATION('a', 3)), \
    NODE(1, RELATION('d', 4)),
    NODE()
);
FST fstEnd(LEX_END, SYMB_LEXEMS, "", 5,
    NODE(1, RELATION('e', 1)),
    NODE(1, RELATION('n', 2)),
    NODE(1, RELATION('d', 3)),
    NODE(1, RELATION('p', 4)),
    NODE()
);
FST fstRealLit(LEX_REAL, SYMB_LITERAL, "", 2, NODE(20, RELATION('0', 0), RELATION('1', 0), RELATION('2', 0),
    RELATION('3', 0), RELATION('4', 0), RELATION('5', 0),
    RELATION('6', 0), RELATION('7', 0), RELATION('8', 0),
    RELATION('9', 0),
    RELATION('0', 1), RELATION('1', 1), RELATION('2', 1),
    RELATION('3', 1), RELATION('4', 1), RELATION('5', 1),
    RELATION('6', 1), RELATION('7', 1), RELATION('8', 1),
    RELATION('9', 1)),
    NODE()
);
```

```
FST fstId(LEX_ID, SYMB_TABLE_ID, "", 2,
    NODE(52, RELATION('a', 0), RELATION('b', 0), RELATION('c', 0), RELATION('d', 0),
        RELATION('e', 0), RELATION('f', 0), RELATION('g', 0), RELATION('h', 0),
        RELATION('i', 0), RELATION('j', 0), RELATION('k', 0), RELATION('l', 0),
        RELATION('m', 0), RELATION('n', 0), RELATION('o', 0), RELATION('p', 0),
        RELATION('q', 0), RELATION('r', 0), RELATION('s', 0), RELATION('t', 0),
        RELATION('u', 0), RELATION('v', 0), RELATION('w', 0), RELATION('x', 0),
        RELATION('y', 0), RELATION('z', 0),
        RELATION('a', 1), RELATION('b', 1), RELATION('c', 1), RELATION('d', 1),
        RELATION('e', 1), RELATION('f', 1), RELATION('g', 1), RELATION('h', 1),
        RELATION('i', 1), RELATION('j', 1), RELATION('k', 1), RELATION('l', 1),
        RELATION('m', 1), RELATION('n', 1), RELATION('o', 1), RELATION('p', 1),
        RELATION('q', 1), RELATION('r', 1), RELATION('s', 1), RELATION('t', 1),
        RELATION('u', 1), RELATION('v', 1), RELATION('w', 1), RELATION('x', 1),
        RELATION('y', 1), RELATION('z', 1)),
    NODE()
);
```

```
FST fstStrLit(LEX_CHR, SYMB_LITERAL, "", 4, NODE(2, RELATION('"', 1), RELATION('"', 2)), \
    NODE(276, \
        RELATION('A', 1), RELATION('B', 1), RELATION('C', 1), RELATION('D', 1), RELATION('E', 1), RELATION('F', 1), \
        RELATION('G', 1), RELATION('H', 1), RELATION('I', 1), RELATION('J', 1), RELATION('K', 1), RELATION('L', 1), \
        RELATION('M', 1), RELATION('N', 1), RELATION('O', 1), RELATION('P', 1), RELATION('Q', 1), RELATION('R', 1), \
        RELATION('S', 1), RELATION('T', 1), RELATION('U', 1), RELATION('V', 1), RELATION('W', 1), RELATION('X', 1), \
        RELATION('Y', 1), RELATION('Z', 1), \
        RELATION('a', 1), RELATION('b', 1), RELATION('c', 1), RELATION('d', 1), RELATION('e', 1), RELATION('f', 1), \
        RELATION('g', 1), RELATION('h', 1), RELATION('i', 1), RELATION('j', 1), RELATION('k', 1), RELATION('l', 1), \
        RELATION('m', 1), RELATION('n', 1), RELATION('o', 1), RELATION('p', 1), RELATION('q', 1), RELATION('r', 1), \
        RELATION('s', 1), RELATION('t', 1), RELATION('u', 1), RELATION('v', 1), RELATION('w', 1), RELATION('x', 1), \
        RELATION('y', 1), RELATION('z', 1), RELATION('1', 1), RELATION('2', 1), RELATION('3', 1), RELATION('4', 1), \
        RELATION('5', 1), RELATION('6', 1), RELATION('7', 1), RELATION('8', 1), RELATION('9', 1), RELATION('0', 1), \
        \
        RELATION('А', 1), RELATION('Б', 1), RELATION('В', 1), RELATION('Г', 1), RELATION('Д', 1), RELATION('Е', 1), \
        RELATION('Ё', 1), RELATION('Ж', 1), RELATION('З', 1), RELATION('И', 1), RELATION('Й', 1), RELATION('К', 1), \
        RELATION('Л', 1), RELATION('М', 1), RELATION('Н', 1), RELATION('О', 1), RELATION('П', 1), RELATION('Р', 1), \
        RELATION('С', 1), RELATION('Т', 1), RELATION('У', 1), RELATION('Ф', 1), RELATION('Х', 1), RELATION('Ц', 1), \
        RELATION('Ч', 1), RELATION('Ш', 1), RELATION('Щ', 1), RELATION('Ъ', 1), RELATION('Ы', 1), RELATION('Ь', 1), \
        RELATION('Э', 1), RELATION('Ю', 1), RELATION('Я', 1), \
        RELATION('а', 1), RELATION('б', 1), RELATION('в', 1), RELATION('г', 1), RELATION('д', 1), RELATION('е', 1), \
        RELATION('ё', 1), RELATION('ж', 1), RELATION('з', 1), RELATION('и', 1), RELATION('й', 1), RELATION('к', 1), \
        RELATION('л', 1), RELATION('м', 1), RELATION('н', 1), RELATION('о', 1), RELATION('п', 1), RELATION('р', 1), \
        RELATION('с', 1), RELATION('т', 1), RELATION('у', 1), RELATION('ф', 1), RELATION('х', 1), RELATION('ц', 1), \
        RELATION('ч', 1), RELATION('ш', 1), RELATION('щ', 1), RELATION('ъ', 1), RELATION('ы', 1), RELATION('ь', 1), \
        RELATION('э', 1), RELATION('ю', 1), RELATION('я', 1), RELATION(' ', 1), RELATION('.', 1), RELATION(',', 1), \
        RELATION('?', 1), RELATION('!', 1), RELATION(';', 1), RELATION(':', 1), RELATION('-', 1), RELATION(')', 1), \
        RELATION('(', 1), \
        \
        RELATION('A', 2), RELATION('B', 2), RELATION('C', 2), RELATION('D', 2), RELATION('E', 2), RELATION('F', 2), \
        RELATION('G', 2), RELATION('H', 2), RELATION('I', 2), RELATION('J', 2), RELATION('K', 2), RELATION('L', 2), \
        RELATION('M', 2), RELATION('N', 2), RELATION('O', 2), RELATION('P', 2), RELATION('Q', 2), RELATION('R', 2), \
        RELATION('S', 2), RELATION('T', 2), RELATION('U', 2), RELATION('V', 2), RELATION('W', 2), RELATION('X', 2), \
        RELATION('Y', 2), RELATION('Z', 2), \
        RELATION('a', 2), RELATION('b', 2), RELATION('c', 2), RELATION('d', 2), RELATION('e', 2), RELATION('f', 2), \
        RELATION('g', 2), RELATION('h', 2), RELATION('i', 2), RELATION('j', 2), RELATION('k', 2), RELATION('l', 2), \
        RELATION('m', 2), RELATION('n', 2), RELATION('o', 2), RELATION('p', 2), RELATION('q', 2), RELATION('r', 2), \
        RELATION('s', 2), RELATION('t', 2), RELATION('u', 2), RELATION('v', 2), RELATION('w', 2), RELATION('x', 2), \
        RELATION('y', 2), RELATION('z', 2), RELATION('1', 2), RELATION('2', 2), RELATION('3', 2), RELATION('4', 2), \
        RELATION('5', 2), RELATION('6', 2), RELATION('7', 2), RELATION('8', 2), RELATION('9', 2), RELATION('0', 2), \
        \
```

```
        RELATION('A', 2), RELATION('B', 2), RELATION('C', 2), RELATION('D', 2), RELATION('E', 2), RELATION('F', 2), \
        RELATION('G', 2), RELATION('H', 2), RELATION('I', 2), RELATION('J', 2), RELATION('K', 2), RELATION('L', 2), \
        RELATION('M', 2), RELATION('N', 2), RELATION('O', 2), RELATION('P', 2), RELATION('Q', 2), RELATION('R', 2), \
        RELATION('S', 2), RELATION('T', 2), RELATION('U', 2), RELATION('V', 2), RELATION('W', 2), RELATION('X', 2), \
        RELATION('Y', 2), RELATION('Z', 2), \
        RELATION('a', 2), RELATION('b', 2), RELATION('c', 2), RELATION('d', 2), RELATION('e', 2), RELATION('f', 2), \
        RELATION('g', 2), RELATION('h', 2), RELATION('i', 2), RELATION('j', 2), RELATION('k', 2), RELATION('l', 2), \
        RELATION('m', 2), RELATION('n', 2), RELATION('o', 2), RELATION('p', 2), RELATION('q', 2), RELATION('r', 2), \
        RELATION('s', 2), RELATION('t', 2), RELATION('u', 2), RELATION('v', 2), RELATION('w', 2), RELATION('x', 2), \
        RELATION('y', 2), RELATION('z', 2), RELATION('1', 2), RELATION('2', 2), RELATION('3', 2), RELATION('4', 2), \
        RELATION('5', 2), RELATION('6', 2), RELATION('7', 2), RELATION('8', 2), RELATION('9', 2), RELATION('0', 2), \
        \
        RELATION('А', 2), RELATION('Б', 2), RELATION('В', 2), RELATION('Г', 2), RELATION('Д', 2), RELATION('Е', 2), \
        RELATION('Ё', 2), RELATION('Ж', 2), RELATION('З', 2), RELATION('И', 2), RELATION('Й', 2), RELATION('К', 2), \
        RELATION('Л', 2), RELATION('М', 2), RELATION('Н', 2), RELATION('О', 2), RELATION('П', 2), RELATION('Р', 2), \
        RELATION('С', 2), RELATION('Т', 2), RELATION('У', 2), RELATION('Ф', 2), RELATION('Х', 2), RELATION('Ц', 2), \
        RELATION('Ч', 2), RELATION('Ш', 2), RELATION('Щ', 2), RELATION('Ъ', 2), RELATION('Ы', 2), RELATION('Ь', 2), \
        RELATION('Э', 2), RELATION('Ю', 2), RELATION('Я', 2), \
        RELATION('а', 2), RELATION('б', 2), RELATION('в', 2), RELATION('г', 2), RELATION('д', 2), RELATION('е', 2), \
        RELATION('ё', 2), RELATION('ж', 2), RELATION('з', 2), RELATION('и', 2), RELATION('й', 2), RELATION('к', 2), \
        RELATION('л', 2), RELATION('м', 2), RELATION('н', 2), RELATION('о', 2), RELATION('п', 2), RELATION('р', 2), \
        RELATION('с', 2), RELATION('т', 2), RELATION('у', 2), RELATION('ф', 2), RELATION('х', 2), RELATION('ц', 2), \
        RELATION('ч', 2), RELATION('ш', 2), RELATION('щ', 2), RELATION('ъ', 2), RELATION('ы', 2), RELATION('ь', 2), \
        RELATION('э', 2), RELATION('ю', 2), RELATION('я', 2), RELATION(' ', 2), RELATION('.', 2), RELATION(',', 2), \
        RELATION('?', 2), RELATION('!', 2), RELATION(';', 2), RELATION(':', 2), RELATION('-', 2), RELATION(')', 2), \
        RELATION('(', 2)), \
    \
    NODE(1, RELATION('"', 3)), \
    NODE()
);
FST fstPow(LEX_POW, SYMB_STATIC_LIB, "", 6,
     NODE(1, RELATION('p', 1)), \
     NODE(1, RELATION('o', 2)), \
     NODE(1, RELATION('w', 3)), \
     NODE(1, RELATION('e', 4)), \
     NODE(1, RELATION('r', 5)), \
     NODE()
);
FST fstStrl(LEX_STRLN, SYMB_STATIC_LIB, "", 6,
     NODE(1, RELATION('s', 1)), \
     NODE(1, RELATION('t', 2)), \
     NODE(1, RELATION('r', 3)), \
     NODE(1, RELATION('l', 4)), \
     NODE(1, RELATION('n', 5)),
     NODE()
);
```

```
struct Entry
{
    char lexema;                        // лексема
    int sn;                             // номер строки в исходном коде
    int idxTI;                          // индекс в таблице идентификаторов
    short priority;                     // приоритет для операций
};

struct LexTable
{
    int maxsize;                        // емкость таблицы лексем
    int size;                           // текущий размер таблицы лексем
    Entry* table;                       // массив строк таблицы лексем
};
```

```
enum IDDATATYPE { OFF = 0, DIG = 1, CHR = 2, OPR = 3 };
enum IDTYPE { N = 0, F = 1, V = 2, P = 3, L = 4, O = 5 };   //нет типа, функция, переменная, параметр функции, литерал, оператор

struct Entry
{
    int         id_first_LE;        //индекс первого вхождения в таблице лексем
    string      id;                 //идентификатор
    IDDATATYPE  id_data_type;       //тип данных
    IDTYPE      id_type;            //тип идентификатора
    struct
    {
        int vint;                   //значение integer
        struct {
            int len;                        //длина string
            string str;                     //символы string
        } vstr;                             //значение string
    } value;                //значение идентификатора
};

struct IdTable              //экземпляр таблицы идентификаторов
{
    int maxsize;                        //емкость таблицы идентификаторов < TI_MAXSIZE
    int size;                           //текущий размер таблицы идентификаторов < maxsize
    Entry* table;                       //массив строк таблицы идентификаторов
};
```

Приложение Б

```
Greibach greibach(
    NS('S'), TS('$'),
    6,
    Rule(
        NS('S'), GRB_ERROR_SERIES + 0,
        2,                                  // S->tpi(F)[N]S | m[N]
        Rule::Chain(10, TS('t'), TS('p'), TS('i'), TS('('), NS('F'), TS(')'), TS('['), NS('N'), TS(']'), NS('S')),
        Rule::Chain(4, TS('m'), TS('['), NS('N'), TS(']'))
    ),
    Rule(
        NS('N'), GRB_ERROR_SERIES + 1,     // ошибочный оператор
        9,                                 // N->dti;N | eE; | i=E;N | s(i);N | s(w);N |s(r);N |s(w); |s(i);|s(r);
        Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),
        Rule::Chain(3, TS('e'), NS('E'), TS(';')),
        Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),
        Rule::Chain(6, TS('s'), TS('('), TS('i'), TS(')'), TS(';'), NS('N')),
        Rule::Chain(6, TS('s'), TS('('), TS('w'), TS(')'), TS(';'), NS('N')),
        Rule::Chain(6, TS('s'), TS('('), TS('r'), TS(')'), TS(';'), NS('N')),
        Rule::Chain(5, TS('s'), TS('('), TS('w'), TS(')'), TS(';')),
        Rule::Chain(5, TS('s'), TS('('), TS('i'), TS(')'), TS(';')),
        Rule::Chain(5, TS('s'), TS('('), TS('r'), TS(')'), TS(';'))
    ),

Rule(
    NS('E'), GRB_ERROR_SERIES + 2,    // ошибка в выражении
    16,                                // E->i | w | r | (E) | rM | i(W) | iM | dM | (E)M | i(W)M | q(i,r) | q(r,i)| q(r,r) |q(i,i) |n(w) |n(i) |
    Rule::Chain(1, TS('i')),
    Rule::Chain(1, TS('w')),
    Rule::Chain(1, TS('r')),
    Rule::Chain(3, TS('('), NS('E'), TS(')')),
    Rule::Chain(2, TS('r'), NS('M')),
    Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),
    Rule::Chain(2, TS('i'), NS('M')),
    Rule::Chain(2, TS('d'), NS('M')),
    Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),
    Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M')),
    Rule::Chain(6, TS('q'), TS('('), TS('i'), TS(','), TS('r'), TS(')')),
    Rule::Chain(6, TS('q'), TS('('), TS('r'), TS(','), TS('r'), TS(')')),
    Rule::Chain(6, TS('q'), TS('('), TS('i'), TS(','), TS('i'), TS(')')),
    Rule::Chain(6, TS('q'), TS('('), TS('r'), TS(','), TS('i'), TS(')')),
    Rule::Chain(4, TS('n'), TS('('), TS('i'), TS(')')),
    Rule::Chain(4, TS('n'), TS('('), TS('w'), TS(')'))
),
Rule(
    NS('F'), GRB_ERROR_SERIES + 3,    // ошибка в параметрах функции
    2,                                // F -> ti | ti,F
    Rule::Chain(2, TS('t'), TS('i')),
    Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))
),
```

```
    Rule(
        NS('F'), GRB_ERROR_SERIES + 3,    // ошибка в параметрах функции
        2,                                // F -> ti | ti,F
        Rule::Chain(2, TS('t'), TS('i')),
        Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))
    ),
    Rule(
        NS('W'), GRB_ERROR_SERIES + 4,    // ошибка в параметрах вызываемой функции
        6,                                // W -> i | w | r | i,W | w,W | r,W
        Rule::Chain(1, TS('i')),
        Rule::Chain(1, TS('w')),
        Rule::Chain(1, TS('r')),
        Rule::Chain(3, TS('i'), TS(','), NS('W')),
        Rule::Chain(3, TS('w'), TS(','), NS('W')),
        Rule::Chain(3, TS('r'), TS(','), NS('W'))
    ),
    Rule(
        NS('M'), GRB_ERROR_SERIES + 5,    // оператор
        3,                                // M -> a | aE | aEM
        Rule::Chain(1, TS('a')),
        Rule::Chain(2, TS('a'), NS('E')),
        Rule::Chain(3, TS('a'), NS('E'), NS('M'))
    )
);
```

Приложение В

```
struct MfstState                // состояние автомата (для сохранения)
{
    short lenta_position;       // позиция на ленте
    short nrulechain;           // номер текущего правила
    short nrule;                // номер текущей цепочки текущего правила
    MFSTSTSTACK st;             // стек автомата
    MfstState();
    MfstState(short pposition, MFSTSTSTACK pst, short pnrulechain);
    MfstState(
        short pposition,       // позиция на ленте
        MFSTSTSTACK pst,       // стек автомата
        short pnrule,          // номер текущего правила
        short pnrulechain      // номер текущей цепочки
    );
};
struct Mfst                     // магазинный автомат
{
    enum RC_STEP {                      // код возврата функции step
                NS_OK,                  // найдено правило и цепочка, цепочка записана в стек
                NS_NORULE,              // не найдено правило грамматики (ошибка в грамматике)
                NS_NORULECHAIN,         // не найдена подходящая цепочка правила (ошибка в исходном коде)
                NS_ERROR,               // неизвестный нетерминальный символ грамматики
                TS_OK,                  // тек. символ ленты == вершине стека, продвинулась лента, pop стека
                TS_NOK,                 // тек. символ ленты != вершине стека, восстановлено состояние
                LENTA_END,              // текущая позиция ленты >= lenta_size
                SURPRISE                // неожиданный код возврата (ошибка в step)
                };

    struct MfstDiagnosis         // диагностика
    {
        short lenta_position;           // позиция на ленте
        RC_STEP rc_step;                // код завершения шага
        short nrule;                    // номер правила
        short nrule_chain;              // номер цепочки правила
        MfstDiagnosis();
        MfstDiagnosis(           // диагностика
                    short plenta_position,  // позиция на ленте
                    RC_STEP prc_step,       // код завершения шага
                    short pnrule,           // номер правила
                    short pnrule_chain      // номер цепочки правила
                    );
    } diagnosis[MFST_DIAGN_NUMBER];         // последние самые глубокие сообщения
    GRBALPHABET* lenta;                 // перекодированная (TS/NS) лента (из LEX)
    short lenta_position;               // текущая позиция на ленте
    short nrule;                        // номер текущего правила
    short nrulechain;                   // номер текущей цепочки текущего правила
    short lenta_size;                   // размер ленты
    GRB::Greibach greibach;             // грамматика Грейбах
Lex::LEX lex;                       // результат работы лексического анализатора
MFSTSTSTACK stack;                          // стек автомата
std::stack<MfstState> storestate;   // стек для сохранения состояний
Mfst();
Mfst(
    Lex::LEX plex,          // результат работы лексического анализатора
    GRB::Greibach pgrebach          // грамматика Грейбах
);
char* getContainStack(char* buf);           // получить содержимое стека
char* getCLenta(char* buf, short pos, short n = 25);    // лента: n символов с pos
char* getDiagnosis(short n, char* buf);     // получить n-ую строку диагностики или 0x00
bool Mfst::saveState(Log::LOG log);                 // сохранить состояние автомата
bool Mfst::restState(Log::LOG log);                 // восстановить состояние автомата
bool push_chain(                        // поместить цепочку правила в стек
    GRB::Rule::Chain chain      // цепочка правил
);
Mfst::RC_STEP Mfst::step(Log::LOG log);// выполнить шаг автомата
bool start(Log::LOG log);                       // запустить автомат
bool saveDiagnosis(
    RC_STEP pprc_step   // код завершения шага
);
void Mfst::printRules(Log::LOG &log);   // вывести последовательность правил (дерево разбора)
```

Приложение Г

Начало разбора

```
Шаг :Правило              Входная лента                Стек
0   : S->tpi(F)[N]S       tpi(ti,ti)[dti;i=(iai)ar;    S$
0   : SAVESTATE:          1
0   :                     tpi(ti,ti)[dti;i=(iai)ar;    tpi(F)[N]S$
1   :                     pi(ti,ti)[dti;i=(iai)ar;e    pi(F)[N]S$
2   :                     i(ti,ti)[dti;i=(iai)ar;ei    i(F)[N]S$
3   :                     (ti,ti)[dti;i=(iai)ar;ei;    (F)[N]S$
4   :                     ti,ti)[dti;i=(iai)ar;ei;]    F)[N]S$
5   : F->ti               ti,ti)[dti;i=(iai)ar;ei;]    F)[N]S$
5   : SAVESTATE:          2
5   :                     ti,ti)[dti;i=(iai)ar;ei;]    ti)[N]S$
6   :                     i,ti)[dti;i=(iai)ar;ei;]t    i)[N]S$
7   :                     ,ti)[dti;i=(iai)ar;ei;]tp    )[N]S$
8   : TS_NOK/NS_NORULECHAIN
8   : RESTATE
8   :                     ti,ti)[dti;i=(iai)ar;ei;]    F)[N]S$
9   : F->ti,F             ti,ti)[dti;i=(iai)ar;ei;]    F)[N]S$
9   : SAVESTATE:          2
9   :                     ti,ti)[dti;i=(iai)ar;ei;]    ti,F)[N]S$
10  :                     i,ti)[dti;i=(iai)ar;ei;]t    i,F)[N]S$
11  :                     ,ti)[dti;i=(iai)ar;ei;]tp    ,F)[N]S$
12  :                     ti)[dti;i=(iai)ar;ei;]tpi    F)[N]S$
13  : F->ti               ti)[dti;i=(iai)ar;ei;]tpi    F)[N]S$
13  : SAVESTATE:          3
13  :                     ti)[dti;i=(iai)ar;ei;]tpi    ti)[N]S$
14  :                     i)[dti;i=(iai)ar;ei;]tpi(    i)[N]S$
15  :                     )[dti;i=(iai)ar;ei;]tpi(t    )[N]S$
16  :                     [dti;i=(iai)ar;ei;]tpi(ti    [N]S$
17  :                     dti;i=(iai)ar;ei;]tpi(ti)    N]S$
18  : N->dti;N            dti;i=(iai)ar;ei;]tpi(ti)    N]S$
```

Конец разбора

```
285 : TS_NOK/NS_NORULECHAIN
285 : RESTATE
285 :                       s(w);]                          N]$
286 : N->s(w);N             s(w);]                          N]$
286 : SAVESTATE:            49
286 :                       s(w);]                          s(w);N]$
287 :                       (w);]                           (w);N]$
288 :                       w);]                            w);N]$
289 :                       );]                             );N]$
290 :                       ;]                              ;N]$
291 :                       ]                               N]$
292 : TNS_NORULECHAIN/NS_NORULE
292 : RESTATE
292 :                       s(w);]                          N]$
293 : N->s(r);N             s(w);]                          N]$
293 : SAVESTATE:            49
293 :                       s(w);]                          s(r);N]$
294 :                       (w);]                           (r);N]$
295 :                       w);]                            r);N]$
296 : TS_NOK/NS_NORULECHAIN
296 : RESTATE
296 :                       s(w);]                          N]$
297 : N->s(w);              s(w);]                          N]$
297 : SAVESTATE:            49
297 :                       s(w);]                          s(w);]$
298 :                       (w);]                           (w);]$
299 :                       w);]                            w);]$
300 :                       );]                             );]$
301 :                       ;]                              ;]$
302 :                       ]                               ]$
303 :                                                       $
304 : --------------LENTA_END
305 : ----------------------------->LENTA_END
```

Правила разбора

```
0   : S->tpi(F)[N]S
4   : F->ti,F
7   : F->ti
11  : N->dti;N
15  : N->i=E;N
17  : E->(E)M
18  : E->iM
19  : M->aE
20  : E->i
22  : M->aE
23  : E->r
25  : N->eE;
26  : E->i
29  : S->tpi(F)[N]S
33  : F->ti
37  : N->dti;N
41  : N->i=E;N
43  : E->n(i)
48  : N->eE;
49  : E->i
52  : S->m[N]
54  : N->dti;N
58  : N->dti;N
62  : N->dti;N
66  : N->dti;N
70  : N->dti;N
74  : N->dti;N
78  : N->i=E;N
80  : E->r
82  : N->i=E;N
84  : E->r
86  : N->i=E;N
88  : E->i(W)M
90  : W->i,W

92  : W->i
94  : M->aE
95  : E->r
97  : N->s(i);N
102 : N->dti;N
106 : N->i=E;N
108 : E->w
110 : N->i=E;N
112 : E->i(W)
114 : W->i
117 : N->i=E;N
119 : E->q(i,r)
126 : N->s(i);N
131 : N->s(i);N
136 : N->s(w);
```

Приложение Д

```
void CallPolishNotation(LT::LexTable* lextable, IT::IdTable* idtable)
{
    for (int i = 0; i < lextable->size; i++) {
        if (lextable->table[i].lexema == LEX_EQUAL)
        {
            PolishNotation(++i, lextable, idtable);
        }
    }
}

void AddToResult(IT::IdTable* idtable, LT::Entry* result, LT::Entry elementLT, int *pos, int *flag, int lenout, int lextable_pos)
{
    if (idtable->table[elementLT.indexTI].idFirstInLT >= lextable_pos && idtable->table[elementLT.indexTI].idFirstInLT < (lextable_pos + lenout))
    {
        idtable->table[elementLT.indexTI].idFirstInLT = lextable_pos + *flag;
    }
    *flag += 1;
    result[*pos] = elementLT;
    *pos += 1;
}

void AddToResult(stack<LT::Entry>* stk, IT::IdTable* idtable, LT::Entry* result, int *pos, int *flag, int lenout, int lextable_pos)
{
    if (idtable->table[stk->top().indexTI].idFirstInLT >= lextable_pos && idtable->table[stk->top().indexTI].idFirstInLT < (lextable_pos + lenout))
    {
        idtable->table[stk->top().indexTI].idFirstInLT = lextable_pos + *flag;
    }
    *flag += 1;
    result[*pos] = stk->top();
    *pos += 1;
    stk->pop();
}
```

```
bool PolishNotation(int lextable_pos, LT::LexTable* lextable, IT::IdTable* idtable)
{
    stack<LT::Entry> stackLTelements;
    LT::Entry* elementsLT = new LT::Entry[lextable->size];
    int ncomma = 0,
        flag = 0,
        waste = 0,
        funcPositionTI = -1,
        lenght = 0,                         //общая длина
        lenout = 0,                         //длина выходной строки
        semicolonid;                        //ид для элемента таблицы с точкой с запятой
    LT::Entry tempLTEntry, bufEntry;

    for (int i = lextable_pos; lextable->table[i].lexema != LEX_SEMICOLON; i++) {
        lenout = i + 1;
        semicolonid = i + 1;
    }

    for (int i = lextable_pos; i < lenout; i++) {
        tempLTEntry = lextable->table[i];
        if (tempLTEntry.lexema == LEX_ID || tempLTEntry.lexema == LEX_REAL || tempLTEntry.lexema == LEX_WORD ||
            tempLTEntry.lexema == LEX_POW || tempLTEntry.lexema == LEX_STRLN)
        {
            if (idtable->table[tempLTEntry.indexTI].idType == IT::IDTYPE_FUN)
            {
                funcPositionTI = tempLTEntry.indexTI;
                continue;
            }
            AddToResult(idtable, elementsLT, tempLTEntry, &lenght, &flag, lenout, lextable_pos);
        }
        else {
            if (tempLTEntry.lexema == LEX_ACTION)
            {
                while (!stackLTelements.empty() && stackLTelements.top().priority >= tempLTEntry.priority)
                {
                    if (stackLTelements.top().lexema != LEX_LEFTHESIS)
                    {
                        AddToResult(&stackLTelements, idtable, elementsLT, &lenght, &flag, lenout, lextable_pos);
                    }
                    else break;
                }
                stackLTelements.push(tempLTEntry);
```

```
            if (tempLTEntry.lexema == LEX_COMMA)
            {
                ncomma++;
                while (stackLTelements.top().lexema == LEX_ACTION) {
                    AddToResult(&stackLTelements, idtable, elementsLT, &lenght, &flag, lenout, lextable_pos);
                }
            }
            else if (tempLTEntry.lexema != LEX_RIGHTHESIS)
            {
                if (stackLTelements.empty() || stackLTelements.top().lexema == LEX_LEFTHESIS || tempLTEntry.lexema == LEX_LEFTHESIS)
                {
                    stackLTelements.push(tempLTEntry);
                }
            }
            if (tempLTEntry.lexema == LEX_RIGHTHESIS && tempLTEntry.priority != 4)
            {
                waste += 2;
                while (stackLTelements.top().lexema != LEX_LEFTHESIS) {
                    AddToResult(&stackLTelements, idtable, elementsLT, &lenght, &flag, lenout, lextable_pos);
                }
                stackLTelements.pop();
            }
            if (tempLTEntry.lexema == LEX_RIGHTHESIS && tempLTEntry.priority == 4)
            {
                bufEntry.lexema = LEX_SUBST;
                bufEntry.indexTI = funcPositionTI;
                bufEntry.lineNo = elementsLT[lenght - 1].lineNo;
                bufEntry.priority = ncomma + 1; //здесь хранится кол-во параметров в функции
                elementsLT[lenght++] = bufEntry;
                if (ncomma != 0)
                {
                    waste += ncomma;
                    ncomma = 0;
                }
                while (stackLTelements.top().lexema != LEX_LEFTHESIS) {
                    AddToResult(&stackLTelements, idtable, elementsLT, &lenght, &flag, lenout, lextable_pos);
                }
                stackLTelements.pop();
                waste += 2;
            }
        }
    }
    while (!stackLTelements.empty()) {
        AddToResult(&stackLTelements, idtable, elementsLT, &lenght, &flag, lenout, lextable_pos);
    }

    for (int i = lextable_pos, k = 0; i < lextable_pos + lenght; i++, k++) {
        lextable->table[i] = elementsLT[k];                                          //запись в таблицу польской записи
    }
    lextable->table[lextable_pos + lenght] = lextable->table[semicolonid];          //вставка элемента с точкой с запятой

    for (int i = 0; i < waste; i++) {
        lextable->size--;
        for (int j = lextable_pos + lenght + 1; j < lextable->size; j++) {  // сдвигаем на удалённые литералы
            lextable->table[j] = lextable->table[j + 1];
            if (lextable->table[j].indexTI != TL_TI_NULLIDX && idtable->table[lextable->table[j].indexTI].idFirstInLT == (j + 1))
            {
                idtable->table[lextable->table[j].indexTI].idFirstInLT -= 1;
            }
        }
    }
    return true;
}
```

```
1: tpi(ti,ti)
2: [
3: dti;
4: i=iiara;
5: ei;
6: ]
8: tpi(ti)
9: [
10: dti;
11: i=i@;
12: ei;
13: ]
15: m
16: [
17: dti;
18: dti;
19: dti;
20: dti;
21: dti;
22: dti;
23: i=r;
24: i=r;
25: i=ii@ra;
26: s(i);
27: dti;
28: i=w;
29: i=i@;
30: i=ir@;
31: s(i);
32: s(i);
33: s(w);
34: ]
Количество лексем - 130
```

№	Идентификатор	Тип данных	Тип идентификатора	Индекс в ТЛ	Значение
0	one	real	функция	2	-
1	onea	real	параметр	5	-
2	oneb	real	параметр	8	-
3	onex	real	переменная	13	0
4	+	unknown	оператор	19	-
5	*	unknown	оператор	21	-
6	L0	real	литерал	20	4
7	two	real	функция	29	-
8	twostr	word	параметр	32	-
9	twox	real	переменная	37	0
10	strln	real	функция	41	-
11	entrya	real	переменная	52	0
12	entryb	real	переменная	56	0
13	entryc	real	переменная	60	0
14	entryd	real	переменная	64	0
15	entrye	real	переменная	68	0
16	entryf	real	переменная	72	0
17	L1	real	литерал	76	2
18	L2	real	литерал	80	3
19	-	unknown	оператор	87	-
20	L3	real	литерал	86	1
21	entryg	word	переменная	97	[0]""
22	L4	word	литерал	101	[13]""Hello World""
23	power	real	функция	110	-
24	L5	word	литерал	126	[22]""Завершение программы"

Количество идентификаторов: 22

Приложение Е

```
.586
    .model flat, stdcall
    includelib libucrt.lib
    includelib kernel32.lib
        includelib ../Debug/Lib.lib
    ExitProcess PROTO :DWORD

    readw PROTO: DWORD
    readr PROTO: DWORD
    strln PROTO: DWORD
    power PROTO: DWORD, :DWORD

.stack 4096
.const
    L0 DWORD 4
    L1 DWORD 2
    L2 DWORD 3
    L3 DWORD 1
    L4 BYTE "Hello World", 0
    L5 BYTE "Завершение программы", 0
.data
    onex DWORD ?
    twox DWORD ?
    entrya DWORD ?
    entryb DWORD ?
    entryc DWORD ?
    entryd DWORD ?
    entrye DWORD ?
    entryf DWORD ?
    entryg DWORD ?

.code


one PROC onea : SDWORD, oneb : SDWORD
    push onea
    push oneb
    pop eax
    pop ebx
    add eax, ebx
    push eax
    push L0
    pop eax
    pop ebx
    mul ebx
    push eax
    pop onex
    push onex
    ret
one ENDP

two PROC twostr : DWORD
    push twostr
    pop edx
    push twostr
        call strln
    push eax
    pop twox
    push twox
    ret
two ENDP
```

```
main PROC
    push L1
    pop entrya
    push L2
    pop entryb
    push entrya
    push entryb
    pop edx
    pop edx
    push entryb
    push entrya
        call one
    push eax
    push L3
    pop ebx
    pop eax
    sub eax, ebx
    push eax
    pop entryc
    push entryc
        call readr
    push offset L4
    pop entryg
    push entryg
    pop edx
    push entryg
        call two
    push eax
    pop entryd
    push entrya
    push L2
    pop edx
    pop edx
    push L2
    push entrya

        call power
     push eax
     pop entryf
     push entryd
        call readr
     push entryf
        call readr
     push offset L5
        call readw
     push 0
     call ExitProcess
 main ENDP
 end main
```