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 PAS-2017»

Выполнил студент Пахолко Алена Степановна
(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна
(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Пацей Н.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты ст.пр. Наркевич Аделина Сергеевна
(учен. степень, звание, должность, подпись, Ф.И.О.)

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

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2017

Содержание

Введение

Целью курсового проекта поставлена задача разработки компилятора для моего языка программирования PAS-2017. Этот язык программирования предназначен для выполнения простейших операций и арифметических действий над числами.

Компилятор PAS-2017 – это программа, задачей которого является перевод программы, написанной на языке программирования PAS-2017 в программу на язык ассемблера.

Транслятор PAS-2017 состоит из следующих частей:

– лексический и семантический анализаторы;

– синтаксический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбратка структуры транслятора;

– разработка лексического и семантического анализаторов;

– разработка синтаксического анализатора;

– преобразование выражений;

– генерация кода на язык ассемблера;

– тестирование транслятора.

Решения каждой из поставленных задач буду приведены в соответствующих главах курсового проекта.

Глава 1. Спецификация языка программирования

1.1. Характеристика языка программирования

Язык программирования PAS-2017 классифицируется как процедурный, универсальный, строготипизированный, компилируемый и не объектно-ориентированный язык.

1.2. Алфавит языка

Алфавит языка PAS-2017 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

На этапе выполнения могут использоваться символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, спецсимволы, а также непечатные символы пробела, табуляции и перевода строки. 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	Название	Область применения
‘ ‘	Пробел	Допускается везде, кроме идентификаторов и ключевых слов
;	Точка с запятой	Разделение конструкций
{…}	Фигурные скобки	Заключение программного блока
(…)	Круглые скобки	Приоритет операций, параметры функции
[…]	Квадратные скобки	Заключение программного блока условной конструкции
‘…’	Одинарные кавычки	Допускается везде, кроме идентификаторов и ключевых слов
=	Знак «равно»	Присваивание значения
,	Запятая	Разделение параметров
+ - *	Знаки «плюс», «минус», «умножение»	Выражения

1.4. Применяемые кодировки

Для написания исходного кода на языке программирования PAS-2017 используется кодировка Windows-1251.

1.5. Типы данных

В языке PAS-2017 реализованы два типа данных: целочисленный и строковый. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка PAS-2017

Тип данных	Описание типа данных
Целочисленный тип данных count	Фундаментальный тип данных. Используется для работы с целочисленными положительными значениями. В памяти занимает 4 байта. Максимальное значение: 32768. Минимальное значение: 0. Инициализация по умолчанию: значение 0.
Строковый тип данных road	Фундаментальный тип данных. Используется для работы с символами, каждый символ в памяти занимает 1 байт. Максимальное количество символов: 255. Инициализация по умолчанию: длина 0, символ конца строки “\0”.

1.6. Преобразование типов данных

Преобразование типов данных не поддерживается, т.е. язык является строго-типизированным.

1.7. Идентификаторы

В имени идентификатора допускаются только символы латинского алфавита нижнего регистра. Максимальная длина имени идентификатора - 5 символов. Максимальная длина имени идентификатора функции - 10 символов. При вводе идентификатора длиной более разрешенного количества символов, он будет усекаться. Имя идентификатора не может совпадать с ключевыми словами и не может иметь имя, как функция, уже содержащаяся в стандартной библиотеке.

1.8. Литералы

С помощью литералов осуществляется инициализация переменных. В языке существует два типа литералов. Краткое описание литералов языка PAS-2017 представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

Тип литерала	Регулярное выражение	Описание	Пример
Целочисленный литерал	[1-9]+[0-9]*	Целочисленные неотрицательные литералы, по умолчанию инициализируются 0. Литералы могут быть только rvalue.	oops count sum = 15; 15 – целочисленный литерал.
Строковые литерал	[a-z|A-Z|A-Я|a-я|0-9|!-/]+	Символы, заключённые в ‘’ (одинарные кавычки), по умолчанию инициализируются пустой строкой. Литералы могут быть только rvalue.	oops road str = ‘TEXT’; TEXT – строковый литерал.

1.9. Область видимости идентификаторов

Область видимости «сверху вниз» (по принципу C++). В языке PAS-2017 требуется обязательное объявление переменной перед её инициализацией и последующим использованием. Все переменные должны находиться внутри программного блока. Имеется возможность объявления одинаковых переменных в разных блоках, т. к. переменные, объявленные в одной функции, недоступны в другой.

Каждая переменная получает постфикс – название функции, в которой она объявлена.

1.10. Инициализация данных

При объявлении переменной не допускается инициализация данных. Краткое описание способов инициализации переменных языка PAS-2017 представлено в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

Конструкция	Описание	Пример
oops <тип данных> <идентификатор>;	Автоматическая инициализация: переменные типа count инициализируются нулём, переменные типа road – пустой строкой.	oops count sum; oops road str;
<идентификатор> = <значение>;	Присваивание переменной значения.	sum = 15; str = ‘TEXT’;

1.11. Инструкции языка

Все возможные инструкции языка программирования PAS-2017 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования PAS -2017

Инструкция	Запись на языке PAS -2017
Объявление переменной	oops <тип данных> <идентификатор>;
Присваивание	<идентификатор> = <значение>/<идентификатор>;
Объявление внешней функции	destiny <тип данных> <идентификатор> (<тип данных> <идентификатор>, …)

Блок инструкций	birth
	{
	…
	}

Возврат из подпрограммы	quintessence <идентификатор> / <литерал>;
Условная инструкция	if (условие) [/программный блок если условие верно/] else [/программный блок если условие ложно/]
Вывод целочисленных данных	ququed (<идентификатор> / <литерал>);
Вывод строковых данных	quques (<идентификатор> / <литерал>);

1.12. Операции языка

Язык программирования PAS-2017 может выполнять арифметические операции, представленные в таблице 1.6.

Таблица 1.6 – Операции языка программирования PAS-2017

Операция	Примечание	Типы данных	Пример
=	Присваивание	(count, count) (road, road)	sum = 15; str = ‘TEXT’;
<	Знак «меньше» для условной инструкции	(count, count)	if (sum < diff) […] else […]
(	Приоритет операций	-	sum = (a + b) * c;
)			
+	Суммирование	(count, count)	sum = a + b;
-	Разность	(count, count)	diff = a – b;
*	Умножение	(count, count)	pr = a * b;

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, если эта функция уже содержится в стандартной библиотеке.

1.14. Программные конструкции языка

Ключевые программные конструкции языка программирования PAS-2017 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка PAS-2017

Конструкция	Запись на языке PAS-2017

Главная функция (точка входа)	birth
	{
	…
	quintessence <идентификатор> / <литерал>;
	}

Функция	destiny <тип> <идентификатор> (<тип> <идентификатор>, …)
	{
	…
	quintessence <идентификатор> / <литерал>;
	}

1.15. Область видимости

В языке PAS-2017 все переменные являются локальными. Они обязаны находится 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 не должны повторяться
2	Тип возвращаемого значения должен совпадать с типом функции при её объявлении
3	Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении
4	В функцию должны быть переданы параметры
5	Тип данных результата выражения должен совпадать с типом данных идентификатора, которому оно присваивается

1.17. Распределение оперативной памяти на этапе выполнения

Все переменные размещаются в куче.

1.18. Стандартная библиотека и её состав

Стандартная библиотека PAS-2017 написана на языке программирования C++. Функции стандартной библиотеки с описанием представлены в таблице 1.9.

Таблица 1.9 – Состав стандартной библиотеки

Функция(C++)	Возвращаемое значение	Описание
int isqr(int x, int y);	count	Функция возводит число x в степень y
int isqrt(int x, int y);	count	Функция берет корень степени y из числа x
int strl(string str);	count	Функция вычисляет длину строки str
int ququed(int x)	0	Функция вывода на консоль целочисленного идентификатора/литерала
int quques(char* str)	0	Функция вывода на консоль строкового идентификатора/литерала

1.19. Ввод и вывод данных

В языке PAS-2017 не реализованы средства ввода данных.

Для вывода данных в стандартный поток вывода предусмотрены функции ququed(для данных целочисленного типа) и quques(для данных строкового типа), которые входят в состав стандартной библиотеки и описаны в таблице 1.9.

1.20. Точка входа

В языке PAS-2017 каждая программа должна содержать главную функцию birth, т. е. точку входа, с которой начнется последовательное выполнение программы.

1.21. Препроцессор

Препроцессор в языке программирования PAS -2017 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

PAS -2017 транслируется в язык ассемблера.

1.24. Классификация сообщений транслятора

В случае возникновения ошибки в коде программы на языке PAS-2017 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
120-599	Ошибки лексического анализа
600-699	Ошибки синтаксического анализа
700-900	Ошибки семантического анализа

1.25. Контрольный пример

Контрольный пример представлен в главе Приложения.

Глава 2. Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор преобразует программу, написанную на языке PAS-2017 в программу на языке ассемблера. Для указания выходных файлов используются входные параметры транслятора, которые описаны в пункте 2.2. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.

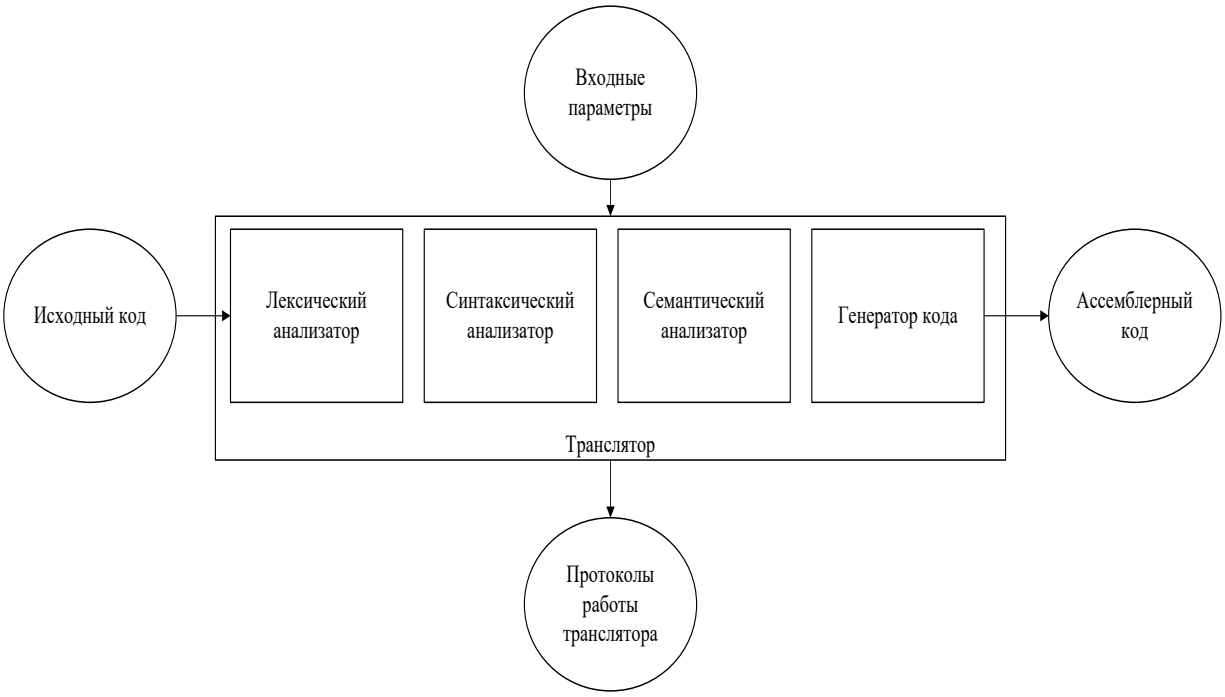

Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка PAS-2017 и формирование таблицы лексем и таблицы идентификаторов.

Семантический анализ в свою очередь является проверкой исходной программы PAS-2017 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода PAS-2017. Для этого используются таблица лексем и идентификаторов. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора

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 PAS-2017, прошедший все предыдущие этапы, в код на языке Ассемблера.

2.2 Перечень входных параметров транслятора

Входные параметры представлены в таблице 2.1.

Таблица 2.1 — Входные параметры транслятора языка PAS-2017

Входной параметр	Описание	Значение по умолчанию
-in:<имя_файла>	Входной файл с расширением .txt, в котором содержится исходный код на PAS-2017. Данный параметр должен быть указан обязательно. В случае если он не будет задан, то выполнение этапа трансляции не начнётся.	Не предусмотрено
-log:<имя_файла>	Файл содержит в себе краткую информацию об исходном коде на языке PAS-2017. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора.	<имя_файла>.log
-lex	Параметр для вывода таблицы лексем в файл “log.txt”.	Не выводится
-id	Параметр для вывода таблицы идентификаторов в файл “log.txt”.	Не выводится
-tree	Параметр для вывода дерева разбора в файл “log.txt”.	Не выводится

2.3 Перечень протоколов, формируемых транслятором и их содержимое

Таблица с перечнем протоколов, формируемых транслятором языка PAS-2017 и их назначением представлена в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка PAS-2017

Формируемый протокол	Описание протокола
Файл журнала, “log.txt”	Файл содержит в себе краткую информацию об исходном коде на языке PAS-2017. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора.
“PASasm.asm”	Содержит сгенерированный код на языке Ассемблера.

Глава 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 PAS-2017. На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1.

Рисунок 3.1 — Структура лексического анализатора PAS-2017

3.2 Контроль входных символов

Исходный код на языке программирования PAS-2017 прежде чем транслироваться проверяется на допустимость символов. То есть изначально из входного файла считывается по одному символу и проверяется является ли он разрешённым.

Таблица для контроля входных символов представлена на рисунке 3.2.

```
IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::T, IN::F, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, IN::S, IN::S, IN::S, IN::S, IN::S, IN::S, IN::T, IN::F,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::S, IN::S, IN::S, IN::S, IN::S, IN::T,
IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::S, IN::F, IN::S, IN::F, IN::F,

IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::S, IN::T, IN::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

Удаление избыточных символов не предусмотрено, так как после проверки на допустимость символов исходный код на языке программирования PAS-2017 разбивается на токены, которые записываются в очередь.

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	Примечание	Цепочка	Лексема
Тип данных	Целочисленный тип данных	count	c
	Строковый тип данных	road	s
Лексема	Объявление переменной	oops	v
	Объявление функции	destiny	f
	Возврат значения из функции	quintessence	r
	Условная инструкция	if	u
		else	e
	Блок условной инструкции	[	[
		]	]
	Блок функции	{	{
		}	}
	Изменение приоритетности в выражении и отделение параметров функций	(	(
		)	)
	Сепараторы	;	;

Продолжение таблицы 3.1

		.	.
		,	,
	Оператор присваивания	=	=
	Условный оператор	<	<
Оператор	Знаки арифметических операций	+	+
		-	-
		*	*
Идентификатор			i
Литерал		Целочисленный литерал	k
		Строковый литерал	l
Точка входа		birth	h
Функция стандартной библиотеки	Возведение числа в степень	isqr	g
	Степенной корень из числа	isqrt	a
	Длина строки	strl	z
	Вывод данных целочисленного типа	quqed	x
	Вывод данных строкового типа	quques	y

Каждому выражению соответствует детерминированный конечный автомат, то есть автомат с конечным состоянием, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся фраза и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Пример реализации таблицы лексем представлен в приложении А.

Также в приложении А находятся конечные автоматы, соответствующие лексемам языка PAS-2017.

3.5 Основные структуры данных

Основные структуры таблиц лексем и идентификаторов данных языка PAS-2017, используемых для хранения, представлены в приложении А. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде и индекс таблицы идентификаторов. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора и его значение.

3.6 Принцип обработки ошибок

При возникновении ошибки типа предупреждение транслятор продолжает свою работу, а предупреждения записываются в специальную структуру с номером ошибки и диагностическим сообщением.

Когда возникает критическая ошибка – работа транслятора прекращается.

В случаях, если на этапе было найдено менее 5 предупреждений, они будут выведены в файл после окончания этапа трансляции.

В случае, если была найдена критическая ошибка или в структуре находятся 5 предупреждений – работа транслятора прекращается и диагностическое сообщение будет выведено в файл.

3.7 Структура и перечень сообщений лексического анализатора

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 120-125. Также сам текст ошибки содержит в себе префикс [LexA]. Перечень сообщений лексического анализатора представлен на рисунке 3.3.

```
ERROR_ENTRY(120, "[LexA]: Ошибка при разборе токена"),
ERROR_ENTRY(121, "[LexA]: Используется необъявленный идентификатор"),
ERROR_ENTRY(122, "[LexA]: Переполнение таблицы идентификаторов"),
ERROR_ENTRY(123, "[LexA]: Переполнение таблицы лексем"),
ERROR_ENTRY(124, "[LexA]: Отсутствует точка входа"),
ERROR_ENTRY(125, "[LexA]: Обнаружено несколько точек входа"),
```

Рисунок 3.3 – Перечень ошибок лексического анализатора

3.8 Параметры лексического анализатора и режимы его работы

Входные параметры используются для вывода результата работы лексического анализатора. Представлены в таблице 3.2.

Таблица 3.2 Входные параметры транслятора языка PAS-2017

Входной параметр	Описание
-lex	Параметр для вывода таблицы лексем в файл протокола
-id	Параметр для вывода таблицы идентификаторов в файл протокола

3.9 Алгоритм лексического анализа

Алгоритм работы лексического анализа заключается в последовательном распознавании и разборе цепочек исходного кода и заполнение таблиц идентификаторов и лексем. Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. В случае, если автомат не был подобран, запоминается номер строки, в которой находился этот токен и в последствии будет выведено сообщение об ошибке. Если токен разобран, то дальнейшие действия, которые будут с ним производиться, будут зависеть от того, чем он является. 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.

В случае, если токен является знаком арифметической операции либо функцией стандартной библиотеки, то он заносится в таблицу идентификаторов.

В случае, если является литералом, то заносится в таблицу идентификаторов с именем “literal<n>”, где n является номером литерала.

Когда встречаем токен, являющийся ключевым словом, которое отвечает за тип данных, заносим лексему, соответствующую ему, в таблицу лексем и запоминаем тип данных, которому он соответствует. В последствии, когда встречаем идентификатор, заносим его в таблицу идентификаторов с соответствующим ему типом данных и именем вида “aaaaabbbcc”, где a – символы из имени идентификатора, b – постфикс из имени функции, в которой он находится и c – номер функции.

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

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

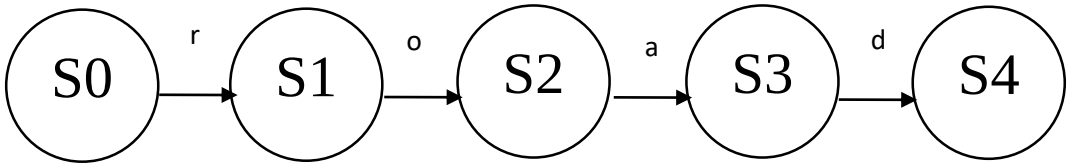

Рисунок 3.4 – Граф переходов для цепочки ‘road’

3.10 Контрольный пример

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

Глава 4. Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций PAS-2017. 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.

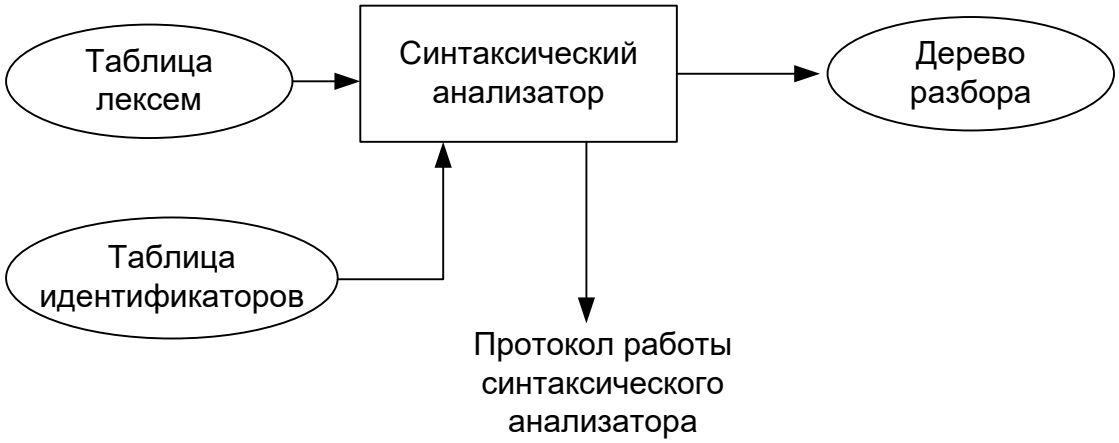

Рисунок 4.1 – Структура синтаксического анализатора PAS-2017

4.2 Контекстно свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка PAS-2017 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 PAS-2017 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 PAS-2017

Нетерминал	Цепочки правил	Описание
S	fci(F){N}S fsi(F){N}S h{N}	Порождает правила, описывающее общую структуру программы
N	vci;N vsi;N i=E;N u(i<i)[N]N u(i<k)[N]N u(k<i)[N]N u(k<k)[N]N u(i<i)[N]Y u(i<k)[N]Y u(k<i)[N]Y u(k<k)[N]Y	Порождает правила, описывающие конструкции языка
E	i iM l lM k kM g(k,i) g(i,i) g(i,k) g(k,k) a(k,i) a(i,i) a(i,k) a(k,k) z(i) z(l) g(k,i)M g(i,i)M g(i,k)M g(k,k)M a(i,i)M a(i,k)M	Порождает правила, описывающие выражения

Продолжение таблицы 4.1

	i(W)	
F	ci ci,F si si,F	Порождает правила, описывающие параметры локальной функции при её объявлении
W	i l k i,W l,W k,W	Порождает правила, описывающие принимаемые параметры функции
Y	e[N]N e[N]	Порождает правила, описывающие обработку условной конструкции, если условие ложно
M	+E -E *E +(E) -(E) *(E)	Порождает правила, описывающие знаки арифметических операций

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.

Продолжение таблицы 4.2

q_0	Начальное состояние автомата	Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ A)
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 конечного автомата и структуру грамматики Грейбах, описывающей правила языка PAS-2017. Данные структуры представлены в приложении В.

4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата следующий:

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

4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен на рисунке 4.1.

```
ERROR_ENTRY(600, "[SinA]: Неверная структура программы"),
ERROR_ENTRY(601, "[SinA]: Ошибочный оператор"),
ERROR_ENTRY(602, "[SinA]: Ошибка в выражении"),
ERROR_ENTRY(603, "[SinA]: Ошибка в параметрах функции"),
ERROR_ENTRY(604, "[SinA]: Ошибка в параметрах вызываемой функции"),
ERROR_ENTRY(605, "[SinA]: Ошибка знака в выражении"),
ERROR_ENTRY(606, "[SinA]: Ошибка в синтаксическом анализе"),
ERROR_ENTRY_NODEF(607),ERROR_ENTRY_NODEF(608),
ERROR_ENTRY(609, "[SinA]: Обнаружена синтаксическая ошибка (журнал Log)"),
```

Рисунок 4.1 – Перечень сообщений синтаксического анализатора

4.7 Параметры синтаксического анализатора и режимы его работы

Для управления результата работы синтаксического анализатора используются входные параметры, описанные в пункте 2.2 Перечень входных параметров транслятора в таблице 2.1.

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 в протокол будет выведено диагностическое сообщение.

4.9 Контрольный пример

Пример разбора синтаксическим анализатором исходного кода на языке PAS-2017 представлен в приложении Г. Дерево разбора исходного кода также представлено в приложении Г.

Глава 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). Структура семантического анализатора представлена на рисунке 5.1.

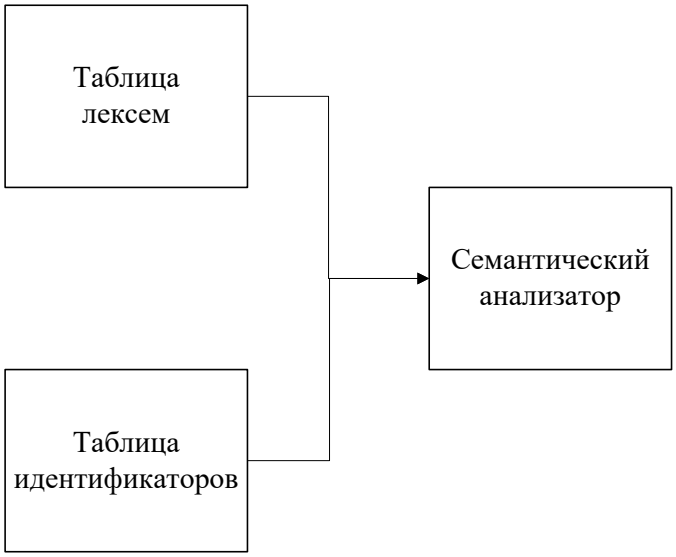

Рисунок 5.1 — структура семантического анализатор

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

Все ошибка семантического анализатора имеют идентификатор свыше 700, а также префикс [SemA]. Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

```
ERROR_ENTRY(700, "[SemA]: Повторное объявление идентификатора"),
ERROR_ENTRY(701, "[SemA]: Ошибка в возвращаемом значении"),
ERROR_ENTRY(702, "[SemA]: Ошибка в передаваемых значениях в функции"),
ERROR_ENTRY(703, "[SemA]: В функцию не переданы параметры"),
ERROR_ENTRY(704, "[SemA]: Тип данных результата выражения не соответствует присваиваемуму идентификатору"),
```

Рисунок 5.2 – Перечень сообщений семантического анализатора

5.4 Принцип обработки ошибок

При обнаружении хотя бы одной ошибки транслятор проведёт только семантический анализ, после чего завершит свою работу. В случае обнаружения нескольких ошибок они будут записаны в массив. По окончании семантического анализа либо достижении максимального количества ошибок в массиве они будут выведены.

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.

Глава 6. Преобразование выражений

6.1 Выражения, допускаемые языком

В языке PAS-2017 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 PAS-2017

Операция	Значение приоритета
(	1
)	1
+	2
-	2
*	3

6.2 Польская запись

Выражения в языке PAS-2017 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:

– читаем очередной символ;

– если он является идентификатором или литералом, то добавляем его к выходной строке;

– если символ является символом функции, то помещаем его в стек;

– если символ является открывающей скобкой, то она помещается в стек;

– исходная строка просматривается слева направо;

– если символ является закрывающей скобкой, то выталкиваем из стека в выходную строку все символы пока не встретим открывающую скобку. При этом обе скобки удаляются и не попадают в выходную строку;

– как только входная лента закончится все символы из стека выталкиваются в выходную строку;

– в случае если встречаются операции, то выталкиваем из стека в выходную строку все операции, которые имеют выше приоритетность чем последняя операция;

– также, если идентификатор является именем функции, то он заменяется на спецсимвол «@».

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

Исходная строка	Результирующая строка	Стек
(x+y)*z		
x+y)*z		(
+y)*z	x	
y)*z	x	(+
)*z	x y	(+
*z	x y +	
z	x y +	*
	x y + z	*
	x y + z *	

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.

Глава 7. Генерация кода

7.1 Структура генератора кода

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы PAS-2017 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 PAS-2017 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке PAS-2017 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка PAS-2017 и языка Ассемблера

Тип идентификатора на языке PAS-2017	Тип идентификатора на языке ассемблера	Пояснение
count	SDWORD	Хранит целочисленный тип данных со знаком.
road	DWORD	Хранит указатель на начало строки.
literal(0-9)	BYTE DWORD	Литералы: символьные, целочисленные

7.3 Алгоритм работы генератора кода

Алгоритм генерации кода выглядит следующим образом:

1) Проходим полностью таблицу идентификаторов и заполняет поле .const литералами. Результат представлен на рисунке 7.2.

```
.const
    literal0 DWORD 2
    literal1 DWORD 3
    literal2 DWORD 15
    literal3 DWORD 10
    literal4 BYTE 'Hello World', 0
    literal5 BYTE 'Google', 0
    literal6 BYTE '10111997', 0
    literal7 DWORD 0
```

Рисунок 7.2 — Пример заполнения поля .const

2) Проходим таблицу лексем и ищем объявление переменных. В случае, если нашли литерал “v”, объявляем данную переменную в поле .data. Результат заполнения поля .data представлен на рисунке 7.3. После того, как данную переменную объявили, она удаляется из таблицы лексем, чтобы избежать переобъявления данного идентификатора в ассемблерном коде.

```
.data
    zfun0 DWORD ?
    tfun0 DWORD ?
    sfun0 DWORD ?
    cfun1 DWORD ?
    xhea DWORD ?
    yhea DWORD ?
    zhea DWORD ?
    sahea DWORD ?
    sbhea DWORD ?
```

Рисунок 7.3 — Пример заполнения поля .data

3) Снова идём по таблице лексем и начинаем описывать функции и конструкции, написанные на языке PAS-2017. Конструкции if-else соответствует шаблон, по которому она и строится на языке ассемблера. В случае, если встречаем лексему “=”, описываем вычисление выражения. Описание алгоритма преобразования выражений представлено в пункте 7.3. Один из вариантов записи функции и конструкций внутри неё представлен на рисунке 7.4.

```
funs PROC bfun1:DWORD, afun1:DWORD
push afun1
call strl
push eax
push bfun1
pop ebx
pop eax
sub eax, ebx
push eax
pop cfun1

    mov eax, cfun1
    ret
funs ENDP
```

Рисунок 7.4 — Пример функции, полученной в результате генерации

7.4. Контрольный пример

Генерируемый код записывается в файл «PASasm.asm». Сгенерированный код можно посмотреть в приложении Е.

Глава 8. Тестирование транслятора

7г8.1 Тестирование фазы проверки на допустимость символов

В языке PAS-2017 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
birth {ы}	111: [IN]: Недопустимый символ в исходном файле (-in);

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

birth
{
oops count Q;
Q=2;
quintessence 0;
}
	120: [LexA]: Ошибка при разборе токена; Строка: 2
	120: [LexA]: Ошибка при разборе токена; Строка: 3

birth
{
x=5;
quintessence 0;
}
	121: [LexA]: Используется необъявленный идентификатор; Строка: 2

destiny count fi(count y)
{
quintessence y;
}
	124: [LexA]: Отсутствует точка входа

birth
birth
{
quintessence 0;
}
	125: [LexA]: Обнаружено несколько точек входа

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

birth
{
oops count q;
q=2
quintessence 0;
}
	Ошибка 609: [SinA]: Обнаружена синтаксическая ошибка (журнал Log)

8.4 Тестирование семантического анализатора

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

Исходный код	Диагностическое сообщение

birth
{
oops count q;
q=2;
oops road w;
w=q;
quintessence 0;
}
	704: [SemA]: Тип данных результата выражения не соответствует идентификатору; Строка 5

birth
{
oops count q;
oops count q;
quintessence 0;
}
	700: [SemA]: Повторное объявление идентификатора; Строка: 13

birth
{
oops count x;
x=5;
quques(x);
quintessence 0;
}
	702: [SemA]: Ошибка в передаваемых значениях в функции; Строка : 4

Приложения

Контрольный пример

```
destiny count fcount(count x, count y)
{
oops count z;
oops count t;
oops count s;
z = isqr(x, 2);
t = isqrt(y, 3);
s = x*(z+t);
quintessence s;
}

destiny count froad(road a, count b)
{
oops count c;
c = strl(a) - b;
quintessence c;
}

birth
{

oops count x;
x = 5;
oops count y;
y = 8;
if (x<y)
[
quques('x is less than y');
]
else
[
quques('y is less than x');
]

oops count z;
z = fcount(x, y);
ququed (z);

oops road sa;
sa = 'PAS-2017';
z = froad(sa, x);
ququed(z);
quintessence 0;
}
```

Приложение А

```
0   : S->fci(F){N}S
4   : F->ci,F
7   : F->ci
11  : N->vci;N
15  : N->vci;N
19  : N->vci;N
23  : N->i=E;N
25  : E->g(i,k)
32  : N->i=E;N
34  : E->a(i,k)
41  : N->i=E;N
43  : E->iM
44  : M->*(E)
46  : E->iM
47  : M->+E
48  : E->i
51  : N->ri;
55  : S->fci(F){N}S
59  : F->si,F
62  : F->ci
66  : N->vci;N
70  : N->i=E;N
72  : E->z(i)M
76  : M->-E
77  : E->i
79  : N->ri;
83  : S->h{N}
85  : N->vci;N
89  : N->i=E;N
91  : E->k
93  : N->vci;N
97  : N->i=E;N
99  : E->k
101 : N->u(i<i)[N]Y
108 : N->y(l);
114 : Y->e[N]N
116 : N->y(l);
122 : N->vci;N
126 : N->i=E;N
128 : E->i(W)
130 : W->i,W
132 : W->i
135 : N->x(i);N
140 : N->vsi;N
144 : N->i=E;N
146 : E->l
148 : N->i=E;N
150 : E->i(W)
152 : W->i,W
154 : W->i
157 : N->x(i);N
162 : N->rk;
```

```
--------Таблица идентификаторов---
№ 0
Имя идентификатора: fcount
Тип данных : count
Тип идентификатора: функция

№ 1
Имя идентификатора: xfco0
Тип данных : count
Тип идентификатора: параметр
Значение: 0

№ 2
Имя идентификатора: yfco0
Тип данных : count
Тип идентификатора: параметр
Значение: 0

№ 3
Имя идентификатора: zfco0
Тип данных : count
Тип идентификатора: переменная
Значение: 0

№ 4
Имя идентификатора: tfco0
Тип данных : count
Тип идентификатора: переменная
Значение: 0

№ 5
Имя идентификатора: sfco0
Тип данных : count
Тип идентификатора: переменная
Значение: 0

№ 12
Имя идентификатора: froad
Тип данных : count
Тип идентификатора: функция

№ 13
Имя идентификатора: afro1
Тип данных : road
Тип идентификатора: параметр
Значение:
Длина: 0

№ 14
Имя идентификатора: bfro1
Тип данных : count
Тип идентификатора: параметр
Значение: 0

№ 15
Имя идентификатора: cfro1
Тип данных : count
Тип идентификатора: переменная
Значение: 0

№ 18
Имя идентификатора: xbir
Тип данных : count
Тип идентификатора: переменная
Значение: 0

№ 20
Имя идентификатора: ybir
Тип данных : count
Тип идентификатора: переменная
Значение: 0

№ 24
Имя идентификатора: zbir
Тип данных : count
Тип идентификатора: переменная
Значение: 0

№ 25
Имя идентификатора: sabir
Тип данных : road
Тип идентификатора: переменная
Значение:
Длина: 0
```

```
FST fstCount("", 'c', TYPE, 6,
    NODE(1, RELATION('c', 1)),
    NODE(1, RELATION('o', 2)),
    NODE(1, RELATION('u', 3)),
    NODE(1, RELATION('n', 4)),
    NODE(1, RELATION('t', 5)),
    NODE());

FST fstRoad("", 's', TYPE, 5,
    NODE(1, RELATION('r', 1)),
    NODE(1, RELATION('o', 2)),
    NODE(1, RELATION('a', 3)),
    NODE(1, RELATION('d', 4)),
    NODE());

FST fstOops("", 'v', LEXEMS, 5,
    NODE(1, RELATION('o', 1)),
    NODE(1, RELATION('o', 2)),
    NODE(1, RELATION('p', 3)),
    NODE(1, RELATION('s', 4)),
    NODE());

FST fstQuintessence("", 'r', LEXEMS, 13,
    NODE(1, RELATION('q', 1)),
    NODE(1, RELATION('u', 2)),
    NODE(1, RELATION('i', 3)),
    NODE(1, RELATION('n', 4)),
    NODE(1, RELATION('t', 5)),
    NODE(1, RELATION('e', 6)),
    NODE(1, RELATION('s', 7)),
    NODE(1, RELATION('s', 8)),
    NODE(1, RELATION('e', 9)),
    NODE(1, RELATION('n', 10)),
    NODE(1, RELATION('c', 11)),
    NODE(1, RELATION('e', 12)),
    NODE());

FST fstDestiny("", 'f', LEXEMS, 8,
    NODE(1, RELATION('d', 1)),
    NODE(1, RELATION('e', 2)),
    NODE(1, RELATION('s', 3)),
    NODE(1, RELATION('t', 4)),
    NODE(1, RELATION('i', 5)),
    NODE(1, RELATION('n', 6)),
    NODE(1, RELATION('y', 7)),
    NODE());

FST fstQuques("", 'y', LIB, 7,
    NODE(1, RELATION('q', 1)),
    NODE(1, RELATION('u', 2)),
    NODE(1, RELATION('q', 3)),
    NODE(1, RELATION('u', 4)),
    NODE(1, RELATION('e', 5)),
    NODE(1, RELATION('s', 6)),
    NODE());

FST fstQuqued("", 'x', LIB, 7,
    NODE(1, RELATION('q', 1)),
    NODE(1, RELATION('u', 2)),
    NODE(1, RELATION('q', 3)),
    NODE(1, RELATION('u', 4)),
    NODE(1, RELATION('e', 5)),
    NODE(1, RELATION('d', 6)),
    NODE());

FST fstBirth("", 'h', HEAD, 6,
    NODE(1, RELATION('b', 1)),
    NODE(1, RELATION('i', 2)),
    NODE(1, RELATION('r', 3)),
    NODE(1, RELATION('t', 4)),
    NODE(1, RELATION('h', 5)),
    NODE());

FST fstIsqrt("", 'a', LIB, 6,
    NODE(1, RELATION('i', 1)),
    NODE(1, RELATION('s', 2)),
    NODE(1, RELATION('q', 3)),
    NODE(1, RELATION('r', 4)),
    NODE(1, RELATION('t', 5)),
    NODE());

FST fstIsqr("", 'g', LIB, 5,
    NODE(1, RELATION('i', 1)),
    NODE(1, RELATION('s', 2)),
    NODE(1, RELATION('q', 3)),
    NODE(1, RELATION('r', 4)),
    NODE());

FST fstRoadLen("", 'z', LIB, 5,
    NODE(1, RELATION('s', 1)),
    NODE(1, RELATION('t', 2)),
    NODE(1, RELATION('r', 3)),
    NODE(1, RELATION('l', 4)),
    NODE());

FST fstElse("", 'e', LEXEMS, 5,
    NODE(1, RELATION('e', 1)),
    NODE(1, RELATION('l', 2)),
    NODE(1, RELATION('s', 3)),
    NODE(1, RELATION('e', 4)),
    NODE());

FST fstIf("", 'u', LEXEMS, 3,
    NODE(1, RELATION('i', 1)),
    NODE(1, RELATION('f', 2)),
    NODE());

FST fstPlus("", '+', OPERATOR, 2,
    NODE(1, RELATION('+', 1)),
    NODE());

FST fstMinus("", '-', OPERATOR, 2,
    NODE(1, RELATION('-', 1)),
    NODE());

FST fstRoadLit("", 'l', LIT, 4,
    NODE(1, RELATION((char)0x27, 1)),
    AllSymb(),
    NODE(1, RELATION((char)0x27, 3)),
    NODE());

FST fstStar("", '*', OPERATOR, 2,
    NODE(1, RELATION('*', 1)),
    NODE());

FST fstLeftIf("", '[', LEXEMS, 2,
    NODE(1, RELATION('[', 1)),
    NODE());

FST fstRighBr("", ')', LEXEMS, 2,
    NODE(1, RELATION(')', 1)),
    NODE());

FST fstRighIf("", ']', LEXEMS, 2,
    NODE(1, RELATION(']', 1)),
    NODE());

FST fstLeftBlock("", '{', LEXEMS, 2,
    NODE(1, RELATION('{', 1)),
    NODE());

FST fstLeftBr("", '(', LEXEMS, 2,
    NODE(1, RELATION('(', 1)),
    NODE());

FST fstRighBlock("", '}', LEXEMS, 2,
    NODE(1, RELATION('}', 1)),
    NODE());
```

```
FST fstSemicolon("", ';', LEXEMS, 2,
    NODE(1, RELATION(';', 1)),
    NODE());
FST fstDot("", '.', LEXEMS, 2,
    NODE(1, RELATION('.', 1)),
    NODE());
FST fstComma("", ',', LEXEMS, 2,
    NODE(1, RELATION(',', 1)),
    NODE());
FST fstRavno("", '=', LEXEMS, 2,
    NODE(1, RELATION('=', 1)),
    NODE());
FST fstLess("", '<', LEXEMS, 2,
    NODE(1, RELATION('<', 1)),
    NODE());
FST fstCountLit("", 'k', LIT, 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("", 'i', 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('f', 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('f', 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());
```

```
struct Entry
{
    char lexema;        //лексема
    int sn;             //номер строки в исходном коде
    int idxTI;          //индекс в таблице идентификаторов
};
struct LexTable
{
    int maxize;                         //емкость таблицы лексем
    int size;                           //текущий размер таблицы лексем
    Entry* table;                       //массив строк табилцы лексем
};
```

```
enum IDDATATYPE { COUNT = 1, ROAD = 2};
enum IDTYPE { V = 1, F = 2, P = 3, L = 4, O = 5 };   //переменная, функция, параметр функции, литерал, оператор

struct Entry
{
    int         idxfirstLE;         //индекс первой строки в таблице лексем
    char        id[ID_MAXSIZE];     //идентификатор (автоматические усекается до ID_MAXSIZE)
    IDDATATYPE  iddatatype;         //тип данных
    IDTYPE      idtype;             //тип идентификатора
    union
    {
        int vint;                   //значение integer
        struct {
            int len;                        //длина string
            char str[TI_STR_MAXSIZE];       //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

```
Rule(NS('S'), GRB_ERROR_SERIES, 3,   //  Структура программы
    Rule::Chain(10, TS('f'), TS('c'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('}'), NS('S')),
    Rule::Chain(10, TS('f'), TS('s'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('}'), NS('S')),
    Rule::Chain(4, TS('h'), TS('{'), NS('N'), TS('}'))
),
                    Rule(NS('F'), GRB_ERROR_SERIES + 1, 4,  // параметры ф-ии
                        Rule::Chain(2, TS('c'), TS('i')),
                        Rule::Chain(4, TS('c'), TS('i'), TS(','), NS('F')),
                        Rule::Chain(2, TS('s'), TS('i')),
                        Rule::Chain(4, TS('s'), TS('i'), TS(','), NS('F'))
                    ),
Rule(NS('N'), GRB_ERROR_SERIES + 2, 22,  // возможные конструкции в ф-иях
    Rule::Chain(5, TS('v'), TS('c'), TS('i'), TS(';'), NS('N')),
    Rule::Chain(5, TS('v'), TS('s'), TS('i'), TS(';'), NS('N')),
    Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),
    Rule::Chain(10, TS('u'), TS('('), TS('i'), TS('<'), TS('i'), TS(')'), TS('['), NS('N'), TS(']'), NS('N')),
    Rule::Chain(10, TS('u'), TS('('), TS('i'), TS('<'), TS('k'), TS(')'), TS('['), NS('N'), TS(']'), NS('N')),
    Rule::Chain(10, TS('u'), TS('('), TS('k'), TS('<'), TS('i'), TS(')'), TS('['), NS('N'), TS(']'), NS('N')),
    Rule::Chain(10, TS('u'), TS('('), TS('k'), TS('<'), TS('k'), TS(')'), TS('['), NS('N'), TS(']'), NS('N')),
    Rule::Chain(10, TS('u'), TS('('), TS('i'), TS('<'), TS('i'), TS(')'), TS('['), NS('N'), TS(']'), NS('Y')),
    Rule::Chain(10, TS('u'), TS('('), TS('i'), TS('<'), TS('k'), TS(')'), TS('['), NS('N'), TS(']'), NS('Y')),
    Rule::Chain(10, TS('u'), TS('('), TS('k'), TS('<'), TS('i'), TS(')'), TS('['), NS('N'), TS(']'), NS('Y')),
    Rule::Chain(10, TS('u'), TS('('), TS('k'), TS('<'), TS('k'), TS(')'), TS('['), NS('N'), TS(']'), NS('Y')),
    Rule::Chain(3, TS('r'), TS('i'), TS(';')),
    Rule::Chain(3, TS('r'), TS('k'), TS(';')),
    Rule::Chain(3, TS('r'), TS('l'), TS(';')),
    Rule::Chain(6, TS('y'), TS('('), TS('i'), TS(')'), TS(';'), NS('N')),
    Rule::Chain(6, TS('y'), TS('('), TS('l'), TS(')'), TS(';'), NS('N')),
    Rule::Chain(5, TS('y'), TS('('), TS('i'), TS(')'), TS(';')),
    Rule::Chain(5, TS('y'), TS('('), TS('l'), TS(')'), TS(';')),
    Rule::Chain(6, TS('x'), TS('('), TS('i'), TS(')'), TS(';'), NS('N')),
    Rule::Chain(6, TS('x'), TS('('), TS('k'), TS(')'), TS(';'), NS('N')),
    Rule::Chain(5, TS('x'), TS('('), TS('i'), TS(')'), TS(';')),
    Rule::Chain(5, TS('x'), TS('('), TS('k'), TS(')'), TS(';'))
),
```

```
Rule(NS('E'), GRB_ERROR_SERIES + 3, 28, // выражения
    Rule::Chain(1, TS('i')),
    Rule::Chain(2, TS('i'), NS('M')),
    Rule::Chain(1, TS('l')),
    Rule::Chain(2, TS('l'), NS('M')),
    Rule::Chain(1, TS('k')),
    Rule::Chain(2, TS('k'), NS('M')),
    Rule::Chain(6, TS('g'), TS('('), TS('i'), TS(','), TS('i'), TS(')')),
    Rule::Chain(7, TS('g'), TS('('), TS('i'), TS(','), TS('i'), TS(')'), TS('M')),
    Rule::Chain(6, TS('g'), TS('('), TS('k'), TS(','), TS('i'), TS(')')),
    Rule::Chain(7, TS('g'), TS('('), TS('k'), TS(','), TS('i'), TS(')'), TS('M')),
    Rule::Chain(6, TS('g'), TS('('), TS('i'), TS(','), TS('k'), TS(')')),
    Rule::Chain(7, TS('g'), TS('('), TS('i'), TS(','), TS('k'), TS(')'), TS('M')),
    Rule::Chain(6, TS('g'), TS('('), TS('k'), TS(','), TS('k'), TS(')')),
    Rule::Chain(7, TS('g'), TS('('), TS('k'), TS(','), TS('k'), TS(')'), TS('M')),
    Rule::Chain(6, TS('a'), TS('('), TS('i'), TS(','), TS('i'), TS(')')),
    Rule::Chain(7, TS('a'), TS('('), TS('i'), TS(','), TS('i'), TS(')'), TS('M')),
    Rule::Chain(6, TS('a'), TS('('), TS('k'), TS(','), TS('i'), TS(')')),
    Rule::Chain(7, TS('a'), TS('('), TS('k'), TS(','), TS('i'), TS(')'), TS('M')),
    Rule::Chain(6, TS('a'), TS('('), TS('i'), TS(','), TS('k'), TS(')')),
    Rule::Chain(7, TS('a'), TS('('), TS('i'), TS(','), TS('k'), TS(')'), TS('M')),
    Rule::Chain(6, TS('a'), TS('('), TS('k'), TS(','), TS('k'), TS(')')),
    Rule::Chain(7, TS('a'), TS('('), TS('k'), TS(','), TS('k'), TS(')'), TS('M')),
    Rule::Chain(4, TS('z'), TS('('), TS('i'), TS(')')),
    Rule::Chain(5, TS('z'), TS('('), TS('i'), TS(')'), NS('M')),
    Rule::Chain(4, TS('z'), TS('('), TS('l'), TS(')')),
    Rule::Chain(5, TS('z'), TS('('), TS('l'), TS(')'), NS('M')),
    Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),
    Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')'), NS('M'))
),
      Rule(NS('W'), GRB_ERROR_SERIES + 4, 6, // принимаемые параметры ф-ии
          Rule::Chain(3, TS('i'), TS(','), NS('W')),
          Rule::Chain(3, TS('l'), TS(','), NS('W')),
          Rule::Chain(3, TS('k'), TS(','), NS('W')),
          Rule::Chain(1, TS('l')),
          Rule::Chain(1, TS('i')),
          Rule::Chain(1, TS('k'))
      ),
      Rule(NS('Y'), GRB_ERROR_SERIES + 5, 2, // else
          Rule::Chain(5, TS('e'), TS('['), NS('N'), TS(']'), NS('N')),
          Rule::Chain(4, TS('e'), TS('['), NS('N'), TS(']'))
      ),
      Rule(NS('M'), GRB_ERROR_SERIES + 6, 6, // знаки
          Rule::Chain(2, TS('+'), NS('E')),
          Rule::Chain(4, TS('+'), TS('('), NS('E'), TS(')')),
          Rule::Chain(2, TS('-'), NS('E')),
          Rule::Chain(4, TS('-'), TS('('), NS('E'), TS(')')),
          Rule::Chain(2, TS('*'), NS('E')),
          Rule::Chain(4, TS('*'), TS('('), NS('E'), TS(')'))
      )
```

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 st;                         // 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* getCSt(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(bool need_trace, Log::LOG log);        // сохранить состояние автомата
    bool Mfst::reststate(bool need_trace, 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, bool need_trace);    // выполнить шаг автомата
    bool Mfst::start(bool need_trace, 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)

    struct Deducation                   // вывод
    {
        short size;                         // количество шагов в выводе
        short* nrules;                      // номера правил грамматики
        short* nrulechains;                 // номера цепочек правил грамматики (nrules)
        Deducation() { size = 0; nrules = 0; nrulechains = 0; };
    } deducation;
    bool savededucation();                  // сохранить дерево вывода (дерево разбора)
};
```

Приложение Г

Начало разбора

```
Шаг :Правило              Входная лента                Стек
0   :S->fci(F){N}S        fci(ci,ci){vci;vci;vci;i=     S$
1   : SAVESTATE:           1
1   :                     fci(ci,ci){vci;vci;vci;i=     fci(F){N}S$
2   :                     ci(ci,ci){vci;vci;vci;i=g     ci(F){N}S$
3   :                     i(ci,ci){vci;vci;vci;i=g(     i(F){N}S$
4   :                     (ci,ci){vci;vci;vci;i=g(i     (F){N}S$
5   :                     ci,ci){vci;vci;vci;i=g(i,     F){N}S$
6   :F->ci                ci,ci){vci;vci;vci;i=g(i,     F){N}S$
7   : SAVESTATE:           2
7   :                     ci,ci){vci;vci;vci;i=g(i,     ci){N}S$
8   :                     i,ci){vci;vci;vci;i=g(i,k     i){N}S$
9   :                     ,ci){vci;vci;vci;i=g(i,k)     ){N}S$
10  : TS_NOK/NS_NORULECHAIN
10  : RESSTATE
10  :                     ci,ci){vci;vci;vci;i=g(i,     F){N}S$
11  :F->ci,F              ci,ci){vci;vci;vci;i=g(i,     F){N}S$
12  : SAVESTATE:           2
12  :                     ci,ci){vci;vci;vci;i=g(i,     ci,F){N}S$
13  :                     i,ci){vci;vci;vci;i=g(i,k     i,F){N}S$
14  :                     ,ci){vci;vci;vci;i=g(i,k)     ,F){N}S$
15  :                     ci){vci;vci;vci;i=g(i,k);     F){N}S$
16  :F->ci                ci){vci;vci;vci;i=g(i,k);     F){N}S$
17  : SAVESTATE:           3
```

Конец разбора

```
437 :N->rk;               rk;}                          N}$
438 : SAVESTATE:           52
438 :                     rk;}                          rk;}$
439 :                     k;}                           k;}$
440 :                     ;}                            ;}$
441 :                     }                             }$
442 :                                                   $
443 : LENTA_END
444 : ------>LENTA_END
```

Правила разбора

```
0   : S->fci(F){N}S
4   : F->ci,F
7   : F->ci
11  : N->vci;N
15  : N->vci;N
19  : N->vci;N
23  : N->i=E;N
25  : E->g(i,k)
32  : N->i=E;N
34  : E->a(i,k)
41  : N->i=E;N
43  : E->iM
44  : M->*(E)
46  : E->iM
47  : M->+E
48  : E->i
51  : N->ri;
55  : S->fci(F){N}S
59  : F->si,F
62  : F->ci
66  : N->vci;N
70  : N->i=E;N
72  : E->z(i)M
76  : M->-E
77  : E->i
79  : N->ri;
83  : S->h{N}
85  : N->vci;N
89  : N->i=E;N
91  : E->k
93  : N->vci;N
97  : N->i=E;N
99  : E->k
101 : N->u(i<i)[N]Y
108 : N->y(l);
114 : Y->e[N]N
116 : N->y(l);
122 : N->vci;N
126 : N->i=E;N
128 : E->i(W)
130 : W->i,W
132 : W->i
135 : N->x(i);N
140 : N->vsi;N
144 : N->i=E;N
146 : E->l
148 : N->i=E;N
150 : E->i(W)
152 : W->i,W
154 : W->i
157 : N->x(i);N
162 : N->rk;
```

Приложение Д

```
void PolishNotation(LT::LexTable* LexT, IT::IdTable* IdT)
{
    for (int i = 0; i < LexT->size; i++)
    {
        if (LexT->table[i].lexema == LEX_RAVNO)
        {
            Conversation(LexT, IdT, ++i);
        }
    }
}
void Conversation(LT::LexTable* lextable, IT::IdTable* idtable, int lextable_pos)
{
    std::stack<LT::Entry> st;
    LT::Entry outstr[200];
    int len = 0,                            //общая длина
        lenout = 0,                         //длина выходной строки
        semicolonid;                        //ид для элемента таблицы с точкой с запятой
    char t, oper, hesis = 0;                //текущий символ/знак оператора/кол-во скобок
    int indoffunk;                          //индекс для замены на элемент с функцией
    for (int i = lextable_pos; lextable->table[i].lexema != LEX_SEMICOLON; i++)
    {
        len = i;
        semicolonid = i + 1;
    }
    len++;
    for (int i = lextable_pos; i < len; i++)
    {
        t = lextable->table[i].lexema;
        if (lextable->table[i].lexema == LEX_PLUS || lextable->table[i].lexema == LEX_MINUS ||
            lextable->table[i].lexema == LEX_STAR)
            oper = idtable->table[lextable->table[i].idxTI].id[0];
        if (t == LEX_RIGHTHESIS)                        //выталкивание всего до другой левой скобки
        {
     while (st.top().lexema != LEX_LEFTHESIS)
     {
         outstr[lenout++] = st.top();           //записываем в выходную строку очередной символ между скобками
         hesis++;
         st.pop();                              //удаляем вершину стека
     }
     st.pop();                                  //удаляем левую скобку в стеке
 }
 if (t == LEX_ID || t == LEX_LITERALSTR || t == LEX_LITERALDIG ||
     t == 'z' || t == 'g' || t == 'a' || t == 'j')
 {
     if (lextable->table[i + 1].lexema == LEX_LEFTHESIS)
     {
         indoffunk = i;
         i += 2;
         while (lextable->table[i].lexema != LEX_RIGHTHESIS)
         {                                  //пока внутри аргументов функции, переписываем их в строку
             if (lextable->table[i].lexema != LEX_COMMA)
             {
                 outstr[lenout++] = lextable->table[i++];
             }
             else
             {
                 hesis++;
                 i++;
             }
         }
         outstr[lenout++] = lextable->table[indoffunk];
         outstr[lenout - 1].lexema = LEX_NEWPROC;
         hesis += 2;
     }
     else
         outstr[lenout++] = lextable->table[i];
 }
```

```
        if (t == LEX_LEFTHESIS)
        {
            st.push(lextable->table[i]);                                //помещаем в стек левую скобку
            hesis++;
        }

        if (oper == '+' || oper == '-' || oper == '*')
        {
            if (!st.size())
                st.push(lextable->table[i]);
            else {
                int pr, id;
                if (st.top().lexema == '(' || st.top().lexema == ')')
                    pr = 1;
                else {
                    id = st.top().idxTI;
                    pr = ArifmPriorities(idtable->table[id].id[0]);
                }
                if (ArifmPriorities(oper) > pr)
                    st.push(lextable->table[i]);
                else
                {
                    while (st.size() && ArifmPriorities(oper) <= ArifmPriorities(idtable->table[id].id[0]))
                    {
                        outstr[lenout] = st.top();
                        st.pop();
                        lenout++;
                    }
                    st.push(lextable->table[i]);
                }
            }
        }
        oper = NULL;                //обнуляем поле знака
    }
        while (st.size())
        {
            outstr[lenout++] = st.top();                                //вывод в строку всех знаков из стека
            st.pop();
        }
        for (int i = lextable_pos, k = 0; i < lextable_pos + lenout; i++, k++)
        {
            lextable->table[i] = outstr[k];                             //запись в таблицу польской записи
        }
        lextable->table[lextable_pos + lenout] = lextable->table[semicolonid];          //вставка элемента с точкой с запятой
        for (int i = 0; i < hesis; i++)
        {
            for (int j = lextable_pos + lenout + 1; j < lextable->size; j++)                //сдвигаем на лишнее место
            {
                lextable->table[j] = lextable->table[j + 1];
            }
            lextable->size--;
        }
    }
    int ArifmPriorities(char symb)
    {
        if (symb == LEX_LEFTHESIS || symb == LEX_RIGHTHESIS)
            return 1;
        if (symb == LEX_PLUS || symb == LEX_MINUS)
            return 2;
        if (symb == LEX_STAR)
            return 3;
    }
}
```

```
------------------------------------------------------------------
-------------------------Таблица идентификаторов------------------
------------------------------------------------------------------
№0: Функция типа count fcount

№1: Параметр типа count xfco0 = 0

№2: Параметр типа count yfco0 = 0

№3: Переменная типа count zfco0 = 0

№4: Переменная типа count tfco0 = 0

№5: Переменная типа count sfco0 = 0

№7: Литерал типа count literal0 = 2

№9: Литерал типа count literal1 = 3

№12: Функция типа count froad

№13: Параметр типа road afro1 = '' (длина = 0)

№14: Параметр типа count bfro1 = 0

№15: Переменная типа count cfro1 = 0

№18: Переменная типа count xbir = 0

№19: Литерал типа count literal2 = 5

№20: Переменная типа count ybir = 0

№21: Литерал типа count literal3 = 8

№22: Литерал типа road literal4 = 'x is less than y' (длина = 16)

№23: Литерал типа road literal5 = 'y is less than x' (длина = 16)

№24: Переменная типа count zbir = 0

№25: Переменная типа road sabir = '' (длина = 0)

№26: Литерал типа road literal6 = 'PAS-2017' (длина = 8)

№27: Литерал типа count literal7 = 0

------------------------------------------------------------------
------------------------------------------------------------------
```

Приложение Е

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

    includelib ../Debug/PASlib.lib
    quques PROTO :DWORD
    qugued PROTO :DWORD
    strl PROTO :DWORD
    isqr PROTO :DWORD, :DWORD
    isqrt PROTO :DWORD, :DWORD

.stack 4096
.const
    literal0 DWORD 2
    literal1 DWORD 3
    literal2 DWORD 5
    literal3 DWORD 8
    literal4 BYTE 'x is less than y', 0
    literal5 BYTE 'y is less than x', 0
    literal6 BYTE 'PAS-2017', 0
    literal7 DWORD 0
.data
    zfco0 SDWORD ?
    tfco0 SDWORD ?
    sfco0 SDWORD ?
    cfro1 SDWORD ?
    xbir SDWORD ?
    ybir SDWORD ?
    zbir SDWORD ?
    sabir DWORD ?

.code
fcount PROC yfco0:DWORD, xfco0:DWORD
push xfco0
push literal0
call isqr
push eax
pop zfco0

push yfco0
push literal1
call isqrt
push eax
pop tfco0
push xfco0
push zfco0
push tfco0
pop eax
pop ebx
add eax, ebx
push eax
pop eax
pop ebx
mul ebx
push eax
pop sfco0

    mov eax, sfco0
    ret
fcount ENDP

froad PROC bfro1:DWORD, afro1:DWORD
push afro1
call strl
push eax
push bfro1
pop ebx
pop eax
sub eax, ebx
push eax
pop cfro1

    mov eax, cfro1
    ret
froad ENDP

main PROC
push literal2
pop xbir

push literal3
pop ybir

mov eax, xbir
cmp eax, ybir
jb less
ja more
less:
    push offset literal4
    call quques
flag:
push xbir
push ybir
call fcount
push eax
pop zbir

    push zbir
    call qugued
push offset literal6
pop sabir

push sabir
push xbir
call froad
push eax
pop zbir

    push zbir
    call qugued
push 0
call ExitProcess
main ENDP
end main
```

Заключение

По окончании выполнения всех пунктов, изложенных ранее, получили рабочий транслятор языка программирования PAS-2017 на язык ассемблера.

Язык PAS-2017 поддерживает 2 типа данных: целочисленный (count), строковый (road).

Для целочисленного типа реализована обработка 3 стандартных арифметических действий, скобок, обозначающих приоритет операций и условных скобок, 3 функций стандартной библиотеки .

Для строковых переменных реализованы 2 функции стандартной библиотеки.

На этапе семантического анализа производится проверка соответствия исходного кода спецификации по 5 правилам.

Реализованы 5 функции стандартной библиотеки.

Итоговое суммарное количество строк исходного кода ~2900.

Литература

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

2. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – М.: Вильямс, 2005. – 912с.